KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Design and Optimization of Hybrid Main Memory Systems using Machine Learning

Emil Karimov

Supervisor:
Prof. dr. ir. F. Catthoor

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

June 2025

Design and Optimization of Hybrid Main Memory Systems using Machine Learning

Emil KARIMOV

Examination committee:
Prof. dr. ir. J. Berlamont, chair
Prof. dr. ir. F. Catthoor, supervisor
Dr. ir. A. Mallik
Prof. dr. ir. M. Verhelst
Dr. ir. D. Preuveneers
Prof. dr. S. Hamdoui
(TU Delft)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

June 2025

© 2025 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Emil Karimov, Kapeldreef 75 - box 4055, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Acknowledgments

This thesis has had a long way to its completion, and I would like to wholeheartedly thank all of my friends and family for simply being my friends and family. Their presence and support makes the PhD journey much more different than if I had done it alone.

In this brief preface, I would like to first and foremost thank my parents and my sister, who have been there for me through all the years - the changes, the uncertainties, the frustrations, and the successes. I am honored and lucky to have them. Çox sağ olun!

I would like express my dearest gratitude to my supervisor, Prof. Francky Catthoor, who despite his busy schedule and the chaos of onboarding and starting a new PhD in the midst of the global pandemic, always patiently and wisely navigated my deviations from the topic and provided a voice of reason. I doubt that this PhD would have been possible without his guidance. Thank you to all the jury members for their valuable feedback that improved my work.

Two people that contributed significantly to my PhD, especially in its early stages, are Saeideh Alinezhad Chamazcoti and Timon Evenblij. I would like to thank Saeideh for the countless hours of discussing trace analysis and bugs in Ramulator. Timon is who first met me at imec and provided the early vision of my PhD trajectory. Hard to say that I followed it, but without Timon laying the groundwork it would have been much more different. I would like to thank the rest of the CSA team for providing the social and working environment and becoming my research group outside of KU Leuven.

I am also incredibly thankful to Dwaipayan Biswas, who amid all the reorganizations followed upon my work and enriched its scope. Many thanks to Khakim for collaboration and friendly discussions.

The highlights of this journey are the friends that I have made. I would like to thank, in no particular order, Kalliopi, Nathan, Giuseppe, Nitish, Giuliano,

Diana, Tommaso, and Leandro. Thanks for the endless debates on what is a soup and the meaning of life, and many other fun activities.

Thanks to Guillaume, Cameron, Akash, Sophie, and the GS guy, yelling "Just go up!" while climbing plastic holds for several hours became a great way to spend time. I thoroughly enjoy our formal delegation trips with Vlad, David, and Ingmar, and the *engaging* discussions that we have. And I would like to thank Marta for being the one who always made it interesting and vibrant (an encompassing description), and just being a good friend. I am lucky to have met such great people over these years in Leuven.

Despite an ocean and several timezones between, I am grateful to have my dear friends Richard and Joo going all the way back to our undergrad. It would be difficult to condense so many years into a few words. Thank you for always being there with the countless hours of games and chats, and for all the trips and memories we have made together.

I also would like to say thanks to Nihat for a friendship pushing two decades, and a strange geographical proximity throughout those years. I am very happy to have a friend like Alice and that our rant sessions have been going on since the RTT days. To have so many great friends is a privilege. There are many others whom I can endlessly list and I am extremely happy to know all of you. PhD is but a chapter and the journey goes on, for which I could not have wished for better companions.

To everyone, without a doubt:
Thank you very much!

E.K.

Abstract

The rising cost of modern large-scale systems stems from workloads with increasing data demands and the associated expenses of the de-facto main memory technology - DRAM. It faces limitations such as high leakage power, refresh frequency, and limited scalability. Emerging alternative memory technologies address these issues of leakage power and density but introduce concerns over higher access latency, endurance, and access energy. This drives the development of Hybrid Main Memory (HMM) systems that complement DRAM with alternative memory technologies.

HMM systems enable larger memory capacities for workloads with expanding working set sizes but add design challenges of selecting memory technologies, their organization hierarchy, and data management. For the latter, the existing literature heavily focuses on complex and costly migration policies based on naive initial data placement. Suboptimal placement strategies lead to increased data movement that reduces the efficiency of the memory system. Moreover, assumptions in access patterns and memory device specifications limit adaptability. The result is increased inefficiencies under dynamic workloads and system configuration variability, highlighting the importance of data placement.

This dissertation proposes an adaptive policymaker for initial data placement in HMM systems by leveraging reinforcement learning. PARL, or Page Arbitration using Reinforcement Learning, operates at the boundary of virtual and physical address spaces, hiding the memory heterogeneity. PARL learns by observing the state of the system and serves the objective to minimize data migration and improve workload runtime. PARL adapts to various workloads and memory configurations, improving runtime by up to 43% and DRAM hitrates by up to 21% compared to existing HMM strategies. By redefining memory management as a resource utilization problem and avoiding the granular analysis of memory accesses and their patterns, this work advances HMM design by proposing a non-biased, adaptive solution that minimizes the burden of data migration.

Beknopte Samenvatting

De stijgende kosten van moderne grootschalige systemen worden veroorzaakt door workloads met toenemende datavereisten en de bijbehorende kosten van de de facto hoofdgeheugentechnologie - DRAM. Deze technologie kampt met beperkingen zoals een hoog lekvermogen, een hoge verversingsfrequentie en beperkte schaalbaarheid. Opkomende alternatieve geheugentechnologieën pakken deze problemen van lekvermogen en dichtheid aan, maar brengen zorgen met zich mee over hogere toegangslatentie, duurzaamheid en toegangsenergie. Dit stimuleert de ontwikkeling van Hybrid Main Memory (HMM)-systemen die DRAM aanvullen met alternatieve geheugentechnologieën.

HMM-systemen maken grotere geheugencapaciteiten mogelijk voor workloads met steeds grotere werksets, maar brengen ontwerpuitdagingen met zich mee, zoals de selectie van geheugentechnologieën, hun organisatiehiërarchie en gegevensbeheer. Wat dit laatste betreft, richt de bestaande literatuur zich sterk op complexe en kostbare migratiebeleidsregels gebaseerd op naïeve initiële gegevensplaatsing. Suboptimale plaatsingsstrategieën leiden tot een verhoogde gegevensverplaatsing, wat de efficiëntie van het geheugensysteem vermindert. Bovendien beperken aannames in toegangspatronen en specificaties van geheugenapparaten de aanpasbaarheid. Het resultaat is een toename van inefficiëntie bij dynamische workloads en variabiliteit in de systeemconfiguratie, wat het belang van gegevensplaatsing benadrukt.

Dit proefschrift stelt een adaptieve beleidsmaker voor voor de initiële plaatsing van gegevens in HMM-systemen door gebruik te maken van reinforcement learning. PARL, of Pagina-Arbitrage met behulp van Reinforcement Learning, opereert op de grens tussen virtuele en fysieke adresruimten en verbergt de geheugenheterogeniteit. PARL leert door kenmerken op systeemniveau te observeren en dient als doel datamigratie te minimaliseren en de looptijd van de werklast te verbeteren. PARL past zich aan verschillende werklasten en geheugenconfiguraties aan en verbetert de looptijd met tot wel 43% en de DRAM-hitrates met tot wel 21% in vergelijking met bestaande HMM-strategieën.

Door geheugenbeheer te herdefiniëren als een resourcegebruikprobleem en de gedetailleerde analyse van geheugentoegang en hun patronen te vermijden, bevordert dit werk het HMM-ontwerp door een onafhankelijke, adaptieve oplossing voor te stellen die de last van datamigratie minimaliseert.

Contents

Abstract iii

Beknopte Samenvatting v

Contents vii

List of Abbreviations xiii

List of Figures xv

List of Tables xix

1 Introduction 1
- 1.1 Motivation . . . 1
  - 1.1.1 System Architecture and The Memory Wall . . . 2
  - 1.1.2 Memory Hierarchy . . . 2
- 1.2 Memory Technologies . . . 4
  - 1.2.1 Existing Memory Technologies . . . 4
  - 1.2.2 Storage Class Memory for Main Memory . . . 7
- 1.3 Memory Systems . . . 11
  - 1.3.1 Heterogeneous Memory Systems . . . 11
  - 1.3.2 Hybrid Memory Systems . . . 12
- 1.4 Design and Exploration of Hybrid Main Memory . . . 14
  - 1.4.1 Physical Design . . . 15
  - 1.4.2 Management Policies . . . 17
  - 1.4.3 Literature Observations . . . 18
  - 1.4.4 Research Goals . . . 19
  - 1.4.5 Research Proposal . . . 19
- 1.5 Thesis Structure . . . 21

2 Background and Related Work **23**
2.1 Overview of Decision-making Approaches 24
2.1.1 Heuristics 24
2.1.2 Machine Learning and Subtypes 24
2.2 Relevant Components of Memory Management 28
2.2.1 Address Spaces and Memory Pages 28
2.2.2 Page Migration 29
2.3 Related Works in HMM Management 30
2.3.1 High-level Classification 30
2.3.2 System-level Proposals 31
2.3.3 Task-level Proposals 31
2.3.4 Object-level Proposals 32
2.3.5 Page-level Proposals 33
2.4 Decision-making in HMM Management 35
2.4.1 Heuristics for HMM 35
2.4.2 (Un)supervised Learning for HMM 36
2.4.3 Reinforcement Learning for HMM 37
2.4.4 Mixture of Decision-making Methods for HMM 40
2.5 Summary 41

3 Design of HMM Arbitration Methods **43**
3.1 Hybrid Main Memory Environment 43
3.1.1 Baseline Environment Implementation 44
3.1.2 Hybrid Main Memory Support 45
3.1.3 Main Memory Interactions 46
3.2 Proposed Method for Page Arbitration: PARL 50
3.2.1 PARL: Design Overview 50
3.2.2 Action Space Definition 51
3.2.3 Rewards System 52
3.2.4 Environment States and Observations 54
3.2.5 Request-level State Definition 56
3.2.6 System-level State Definition 57
3.2.7 Q-value Update Function 59
3.2.8 Practical Implementation Details 61
3.2.9 Estimated Implementation Overhead 62
3.3 Design Overview of Relevant Works 63
3.3.1 Naive Initial Placement Policies 63
3.3.2 APAMS 64
3.3.3 Kleio 65
3.4 Summary 67

4 System and Experiment Setup **69**
4.1 System and Simulator Configuration 69
4.2 Main Memory Configuration 70
4.2.1 Memory Controller Operation 70
4.2.2 Device Timing Specifications 71
4.2.3 Total Memory Capacity 73
4.3 Overview of Workloads 73
4.3.1 List of Workloads Used 74
4.3.2 Workload Classification using Page Access Timeline . . 75
4.4 Evaluation Metrics 77
4.5 Configuration of PARL 78
4.6 Summary 80

5 Evaluation: Initial Page Placement Policies **81**
5.1 Overview 81
5.2 Workloads with one-time page allocation 83
5.3 Workloads with periodic page allocation 87
5.4 Workloads with continuous page allocation 90
5.5 Limitations of Initial Page Placement Policies 92
5.6 Summary 95

6 Evaluation: Page Migration Policies using Machine Learning **97**
6.1 Overview 98
6.2 Comparison of DRAM Hitrates 99
6.3 Comparison of Workload Runtime 102
6.4 Summary 104

7 Conclusion **105**
7.1 Conclusion 105
7.2 Future Work 108

Publications **111**

A Page Fault Mechanism **113**

B Workload Classification using Page Allocation Timeline **115**

C Exploration of Future Work **120**
C.1 Shared Learning Across Workloads 120
C.2 Memory Technology Exploration 122
C.2.1 Improvement potential under different memory technologies and rewards 122
C.2.2 Hybrid Main Memory with Different Technologies . . . 123
C.2.3 Hybrid Main Memory with Homogeneous Technology . 124

D Declaration of the Generative AI Use **125**

Bibliography **127**

List of Abbreviations

APAMS Address Pattern-Aware Memory System. 33, 34, 64, 65, 72, 77, 81–83, 85, 86, 90, 92, 93

CPU Central Processing Unit. 69

DDR Double Data Rate. xix, 69–71

DRAM Dynamic Random Access Memory. xv–xvii, xix, 4–8, 10–13, 15–18, 30, 32, 34, 39, 47, 49, 51, 53, 54, 58, 63–66, 69–74, 76, 77, 80–83, 85–95, 97–103, 109, 120, 122–124

FCFS first-come first-serve. 70

FR-FCFS first-ready first-come first-serve. 70

GPU Graphics Processing Unit. 32, 33

HMM Hybrid Main Memory. xv–xvii, xix, 12–21, 23, 24, 29–37, 39, 41, 43, 45, 48, 50, 52–55, 59, 61–65, 67, 70, 73, 76, 77, 80–83, 95, 97, 104–108, 122–124

IGZO Indium-Gallium-Zinc Oxide. xv, 7, 10, 11, 15, 122, 123

JEDEC Joint Electron Device Engineering Council. 69

LLC last-level cache. 44, 47, 69, 70, 113

LSTM Long Short-term Memory. 37, 38, 66, 98, 102–104

MDP Markov Decision Process. 38, 39, 55, 95

ML Machine Learning. 24–26, 34–37, 40, 62, 63, 65, 77, 95, 101, 102, 104, 106, 107

MRAM Magnetic Random Access Memory. 8, 9, 15, 35

MTJ Magnetic Tunnel Junction. 8, 9

NUMA Non-Uniform Memory Access. xv, 11, 12

NVM Non-Volatile Memory. 6, 7, 32, 64, 70, 72

OS operating system. 31, 32, 108

PA physical address. 44, 45, 47, 124

PARL Page Arbitration using Reinforcement Learning. xv–xvii, xix, 19–21, 33, 34, 41, 45, 50–53, 56–59, 61–63, 66, 67, 69, 72, 77–82, 85–87, 89–95, 97–110, 120–124

PAS Physical Address Space. 28, 36, 45, 50, 56, 84

PCM Phase-Change Memory. 6, 7, 32, 33, 35, 64, 70, 122, 123

PTM Page Translation Mechanism. 44, 45, 50–52, 61

PTT Page Translation Table. xv, 44–48, 51, 61, 113

RL Reinforcement Learning. xv, xvi, 25–27, 34, 35, 37–41, 50, 52, 54–57, 59, 60, 63, 65, 67, 78, 95, 97, 104, 106, 107, 109, 114, 123

RNN Recurrent Neural Network. 37

SCM-M Storage Class Memory for Main Memory. xix, 7, 8, 10, 12, 13, 15–17, 30, 31, 34, 49, 51, 53, 54, 63, 64, 66, 69–73, 77, 80, 83, 85, 86, 92, 98, 122, 124

SL Supervised Learning. 25, 26, 34, 36, 37, 40

SOT Spin-orbit Torque. 7–9

SRAM Static Random Access Memory. xv, 5, 6

STT Spin-transfer Torque. 7–9, 15, 35

UL Unsupervised Learning. 25, 26, 34, 36–38, 40

VA virtual address. 44, 47

VAS Virtual Address Space. 28, 44, 56, 65, 84

VCMA Voltage-controlled Magnetic Anisotropy. xv, 7, 9, 10

List of Figures

1.1 System Hierarchy - Conceptual View. 2
1.2 Memory Hierarchy - Conceptual View. 3
1.3 Overview of existing memory technologies. Reproduced from [14]. 4
1.4 DRAM bitcell design. 5
1.5 6T-SRAM bitcell design. 6
1.6 Comparison of memory technologies. Image source [38]. 7
1.7 Storage Class Memory Technologies for Main Memory. 8
1.8 Spin-transfer (left) and spin-orbit (right) torque mechanisms. . 9
1.9 VCMA bitcell. Source [43]. 10
1.10 Bitcell diagrams for IGZO-based DRAM. 11
1.11 Local and remote memory accesses in a NUMA system. 12
1.12 Different memory devices in a shared hybrid memory space. . . 12
1.13 HMM organization: hierarchical vs. flat. 16
1.14 HMM arbiter placement options. 16

2.1 Classification of Machine Learning Methods. 25
2.2 Interaction of an RL agent with its environment. 26
2.3 Address Translation in Homogeneous and Hybrid Memory. . . . 29

3.1 Conceptual diagram of a system as modeled by Ramulator. . . 44
3.2 Conceptual diagram of HMM and PARL within Ramulator. . . 45
3.3 Conceptual overview of address translation in HMM. 45
3.4 Page fault count (log-scale) for baseline and modified PTT under different DRAM capacities and various workloads. 47
3.5 Impact of page fault delays on workload runtime. 49
3.6 Internal structure of PARL and interaction with its environment. 51
3.7 Impact of scaled vs. shaped rewards on the number of page faults caused by the agent without environment constraints, for a sample workload. DRAM-only baseline and modified page fault count plotted for reference. 54

3.8 Q-values calculated using the standard Q-update Eqn. (2.1) vs. the discounted delta Q-update Eqn. (3.3) for a simple RL agent in an environment with 10 states in 200 episodes and stable rewards. 60
3.9 Q-values calculated using the standard Q-update Eqn. (2.1) vs. the discounted delta Q-update Eqn. (3.3) for a simple RL agent in an environment with 10 states in 200 episodes and random rewards. 60
3.10 Ramulator codebase visualization with PARL and HMM support. 61
3.11 Total access rates (log-scale) of pages sorted chronologically. . . 64
3.12 Design Overview of Kleio LSTM-based Page Scheduler [58]. . . 65

4.1 Ratio of workload pages that fit into a given memory capacity. 74
4.2 Page allocation and reuse timeline (proportional to each workload). 75
4.3 Page access rates for various workloads. 76
4.4 Exploration rate curve for different decay rates. 78
4.5 Learning curve of improving measured time in memory (in black). Blue lines represent if time in memory followed the ϵ decay equation, bound by the measured maximum and minimum time achieved by PARL for that workload. 79

5.1 Average slowdown from the benchmark runtime, across all workloads, per HMM configuration. 82
5.2 Standard deviation of the average slowdown from the benchmark runtime, across all workloads, per HMM configuration reported in Fig. 5.1. 82
5.3 Relative slowdown for workloads with one-time page allocation. 83
5.4 Page access rate for `462.libquantum`. 84
5.5 Distance of consecutive read addresses in `462.libquantum`. . . 84
5.6 Distribution of DRAM accesses - `429.mcf`. 85
5.7 Distribution of DRAM accesses - `462.libquantum`. 86
5.8 Relative slowdown for workloads with periodic page allocation. 87
5.9 Distribution of DRAM accesses - `471.omnetpp` 88
5.10 Page access rate for `471.omnetpp`. 88
5.11 Distribution of DRAM accesses - `433.milc`. 89
5.12 Relative slowdown for workloads with continuous page allocation. 90
5.13 Distribution of DRAM accesses - 436.cactusADM. 91
5.14 Relative slowdown of `473.astar` (continuous profile). 92
5.15 Page access rate for `473.astar`. 93
5.16 Distribution of DRAM accesses - `473.astar`. 93
5.17 Relative slowdown - `grep-map2`. 94
5.18 Distribution of DRAM accesses - `grep-map2`. 94

6.1 DRAM hitrates of Kleio, PARL, and Oracle (left y-axis) with an overlay of the number of migrations performed by Kleio (right y-axis, semi-log) 99
6.2 Relative runtime slowdown for Kleio from its own benchmark DRAM-only configuration. 102
6.3 Relative slowdown improvements by PARL - 434.zeusmp. . . . 103

A.1 Access rate of pages in the order their appearance. 114

B.1 Occurrence of new page allocations and related statistics 116
B.2 Occurrence of new page allocations and related statistics 117
B.3 Average number of requests between new page allocations for all workloads. 117
B.4 Page occurrence timeline for workloads with a one-time profile. 118
B.5 Page occurrence timeline for workloads with a periodic profile. 118
B.6 Page occurrence timeline for workloads with a continuous profile. 119
B.7 Page occurrence timeline for workloads with a continuous & periodic profile. 119

C.1 PARL with a shared Q-table across 12 workloads, showing improvements in IPC achieved for every workload simulated sequentially. 121
C.2 IPC improvements by PARL in `458.sjeng` using different technology combinations and reward functions. 122
C.3 IPC improvements by PARL in sample workloads using different technology combinations in HMM. 123
C.4 IPC improvements by PARL under DRAM-only HMM. 124

List of Tables

2.1 HMM management policies based on the level of abstraction . . 30

3.1 Reported metrics to analyze memory accesses of workloads . . 48
3.2 Overview of observable attributes in the environment. 55
3.3 Request-level state attributes for PARL. 56
3.4 System-level state attributes for PARL. 57

4.1 Simulated system configuration. 69
4.2 A subset of DRAM and SCM-M device timing parameters operating on the DDR4 standard, in memory clock cycles. 71
4.3 Recorded trace files from the referenced benchmark suites [200, 201]. 75

6.1 Recorded trace files from the referenced benchmark suites [200, 201]. 98
6.2 Geometric mean of DRAM hitrates across all evaluated workloads for every memory configuration. 101

Chapter 1

Introduction

...implications of the obvious [1]

1.1 Motivation

The modern day is increasingly dependent on the results of workloads that employ complex computational models and operate on large volumes of data. These include machine learning applications for language processing, weather prediction, genome sequencing, etc. As the size of the workloads grows, so does the scale of the systems to execute them. The associated operating cost rises too, leading system architects and engineers to make difficult design choices to strike a balance in energy consumption and performance of data centers or high performance computing (HPC) clusters.

The two building blocks of a system - computational and storage units - have long been treated separately due to historical differences in performance, size, and cost. Originally proposed by von Neumann [1] in the 1940s, this principle had allowed engineers to focus on scaling down the dimensions of transistors and increase the computational efficiency. Better performance enabled workloads with higher computational complexity and data demands. In the long run, however, improvements in computation far outpaced the improvements in the storage space. As a result, making data available for computations eventually became an issue, as was already predicted by observations on the computing systems and their design [2–4].

[1] Wm. A. Wulf, Sally A. McKee. "Hitting the memory wall: implications of the obvious." SIGARCH Comput. Archit. News 23-1 (March 1995). https://doi.org/10.1145/216585.216588

1.1.1 System Architecture and The Memory Wall

In a traditional system architecture, computational units are often clustered together and the rest of the system is dedicated to various tiers of data storage (see Fig. 1.1). When data is requested, the higher tiers (closest to the computational units) are checked first. If data was not found there, additional time and energy has to be spent on retrieval from lower tiers further away. Due to technology and cost-related limitations, the capacity of lower memory tiers is usually greater than that of the higher ones. Given that modern workloads require large volumes of data that cannot all be stored on the higher memory tiers, more data has to be located and accessed from further away.

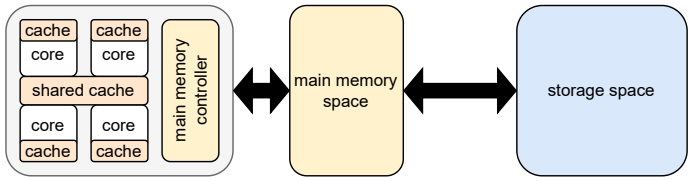


Figure 1.1: System Hierarchy - Conceptual View.

Accessing and moving data across the memory hierarchy will likely result in penalties on performance and energy consumption. This phenomenon is referred to as *the memory wall*, or the von Neumann bottleneck [2–4]. Addressing this issue is an active area of research with potential solutions that involve various memory technologies or modifications to the system architecture. In order to explore and identify a research gap in the existing literature, this work continues with an overview of the memory hierarchy design and memory technologies.

1.1.2 Memory Hierarchy

The memory hierarchy is typically comprised of four levels - on-chip registers, on-chip cache, off-chip main memory, and storage [5–7]. Fig. 1.2 shows the memory hierarchy with typical technologies and the expected access time. This structure could be explained as the result of matching available memory technologies and their characteristics to how workloads are expected to interact with data. The main design parameters for any memory hierarchy level are capacity, performance, energy consumption, and the associated costs of the underlying memory technology such as manufacturing, purchasing, operating, etc. Workloads can access data stored in blocks of different sizes with varying frequency and repetition patterns. It can be seen that designing the memory hierarchy and data placement policy is an intertwined design challenge.

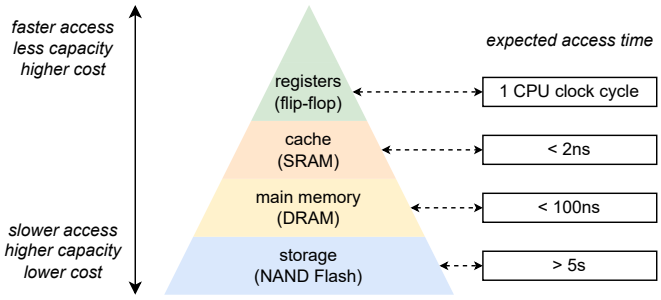


Figure 1.2: Memory Hierarchy - Conceptual View.

As an example, the storage level is often used to store long-term (cold) bulk data, while the most frequently used data is placed in the on-chip memory and physically closer to the computational units [5, 6, 8, 9]. Consequently, the storage level is based on memory technologies that are cost-effective for deployment in larger capacities and their access times are less critical. On-chip memory technologies need to have better performance (lower access latency) and as a result tend to be less cost-effective. This limits their capacity to kilo- or megabytes, enough for only the most recently used data or data predicted to be used by the currently executed workload. Furthermore, each memory hierarchy level can also be heterogeneous within itself. Large-scale systems often divide the storage level into local (internal to the system) and remote (external to the system) access tiers. On-chip memory includes registers that are the fastest memory elements, as well as cache levels that are private to or shared among the computational units (as seen in Fig. 1.1).

The gap in the characteristics of the storage and on-chip memory levels needs to be covered by another level of the memory hierarchy - referred to as the *main memory* [5, 10, 11]. It must act as a middle ground for access latency, capacity, and associated costs, all of which directly impact how often the storage level will be accessed. If the technology used for main memory has large individual memory elements (bitcells), its density and thus capacity will be limited. Access latency must be adequately low to minimize the impact on performance of the workload while data that was requested is transferred from the main memory to the caches. Naturally, writing to main memory must be fast to quickly offload data from the caches when their capacity is exhausted and new data will be fetched from the main memory. In summary, the choices behind the design of memory hierarchy are crucial to system performance and energy efficiency and the main memory is increasingly important [10–13].

1.2 Memory Technologies

The aforementioned latency and capacity requirements constrain the range of technologies that are suitable for main memory. Major existing options are briefly introduced next, followed by an overview of emerging technologies.

1.2.1 Existing Memory Technologies

As shown in Fig. 1.3, memory technologies are generally either volatile or non-volatile according to their data retention ability [5, 12, 14]. Over time, these options were found to match the demands of one of the memory hierarchy levels and as a result, have evolved following those needs.

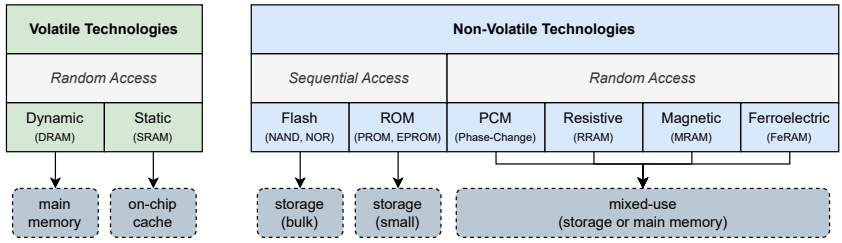


Figure 1.3: Overview of existing memory technologies. Reproduced from [14].

Dynamic Random Access Memory

Since its invention in the 1960s, capacitor-based MOSFET-controlled Dynamic Random Access Memory (DRAM) has been the de-facto choice for the main memory [15–17]. The design of DRAM is based on a capacitor to store a single bit of data, as shown in Fig. 1.4. This places physical and cost-related constrains on DRAM downscaling and they have shown their impact since the early 2000s, with the trend continuing today [17, 18].

As seen in the Fig. 1.4, a single DRAM bitcell has an access transistor and a storage capacitor. The capacitor is electrically charged or discharged to represent a single binary value, i.e. "1" or "0". Multiple bitcells are interconnected into rows and columns via wordlines and bitlines, respectively. A specific bitcell (i.e. the capacitor storing the value) can be addressed via these lines that enable the access transistor. Both the transistor and the capacitor need to be scaled down in size to increase the bitcell density and increase the memory capacity, which is important to meet the demands of modern workloads [17, 19–21].

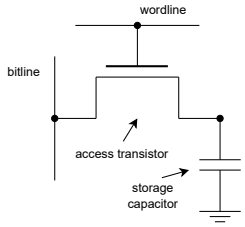


Figure 1.4: DRAM bitcell design.

The limitations in DRAM downscaling are mainly due to its capacitor that dominates the size of the bitcell. Additionally, capacitors tend to discharge over time resulting in leakage current and a refresh mechanism is necessary to prevent data loss [22]. Reducing the size of the capacitor further amplifies these effects - resulting in higher leakage current, increased energy consumption, and increased frequency of invoking the refresh mechanism [23–28]. Downscaling DRAM also increases bit-level errors with security implications such as the RowHammer attack discovered in 2010s [29].

It can be seen that the capacitor-based design of DRAM was a simple and innovative design at first, but it also becomes the leading cause of the main memory bottleneck as modern workloads demand higher memory capacity while maintaining performance and energy consumption efficiency. Though there are potential mitigation strategies for some or most of the technological issues of DRAM, they can be challenging to eliminate entirely and system designers need to look at alternatives suitable for main memory.

Static Random Access Memory

One of the alternatives to DRAM is Static Random Access Memory (SRAM), which is similarly volatile and has low access latency [30]. Built using only transistors (see Fig. 1.5), the capacitor-less design of SRAM also eliminates the need for refresh mechanisms and typical leakage issues of DRAM. Due to these advantages and the compatibility with the regular CMOS manufacturing processes, SRAM has become the standard choice for on-chip memory. While it can be used as main memory technology [31], SRAM has a higher cost per bitcell due to using more transistors than DRAM (e.g. 4-, 6-, and 8-transistor SRAM designs). As a result, SRAM does not meet the density and cost demands placed on the main memory level by modern workloads.

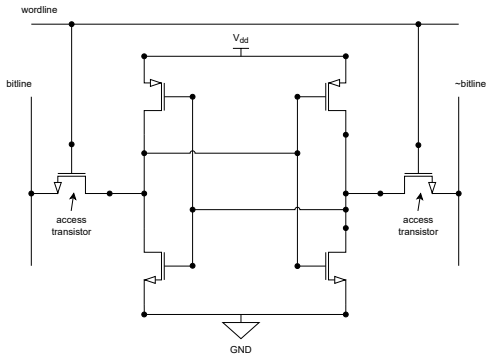


Figure 1.5: 6T-SRAM bitcell design.

Non-Volatile Memory Technologies

In contrast to volatile technologies, Non-Volatile Memory (NVM) is commonly found in storage devices due to indefinitely retaining data after the device is powered down. Various flavors of NVM technologies are available, including but not limited to flash, Phase-Change Memory (PCM), resistive (RRAM), and magnetic (MRAM) devices [32–35]. In addition to no refresh mechanisms and low-to-no leakage current, NVM devices are more cost-effective than DRAM to manufacture in large capacities.

The wide-spread adoption of NVM as main memory devices has been limited due to endurance concerns, access latency higher than DRAM, and potentially higher access energy than DRAM [33, 36–38]. Fig. 1.6 is a comparison plot adopted from the study on challenges of NVM technologies by Kargar et al. [38], which shows how various technologies behave in accordance to these metrics. Additional techniques and management methods have to be developed to account for effects such as non-symmetrical read and write mechanisms (e.g. much higher voltage to write into flash memory [32]). Some technologies also allow for individual addressing - leading to uneven degradation of the bitcells which requires wear-leveling techniques to mitigate the reduced endurance [37, 38].

In summary, when the nuances of each memory technology are taken into account, the perceived advantage of data retention and low leakage current of NVM technologies can disappear under the high access rate expected on the main memory level. As it is an active area of research, emerging alternatives could potentially address the drawbacks of both non-volatile and volatile technologies.

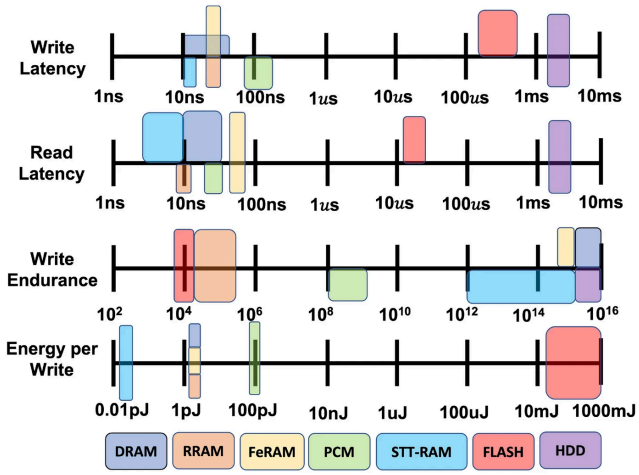


Figure 1.6: Comparison of memory technologies. Image source [38].

1.2.2 Storage Class Memory for Main Memory

The drawbacks of the established NVM devices are expected to be mitigated or eliminated by the Storage Class Memory for Main Memory (SCM-M) [39]. It is an umbrella term for the emerging NVM technologies that exhibit promising characteristics (e.g. lower access latency) and could be used for main memory. SCM-M devices are under active research and development [40–45] and a few selected options are shown in Fig. 1.7.

As it could be seen earlier in Fig. 1.6, Spin-transfer Torque (STT) technology is an example of expected characteristics of SCM-M devices [44]. It has significantly better endurance than traditional NVM devices based on NAND/NOR Flash [32] or PCM [33] technologies, as well as lower access latency. This follows the trend of improving upon the shortcomings of NVM. In summary, the main challenges for SCM-M devices are related to technology maturity, cost, and achieving better-than-DRAM bitcell density. For example, Indium-Gallium-Zinc Oxide (IGZO) DRAM [40, 41] improves upon the shortcomings of DRAM but at the cost of lower density. Spin-orbit Torque (SOT) [45] and Voltage-controlled Magnetic Anisotropy (VCMA) [43] are recent developments in magnetic memory that require further research for deployment in real systems.

Storage Class Memory for Main Memory (SCM-M)			
Volatile Technologies	Non-Volatile Technologies		
IGZO-DRAM	STT-MRAM	SOT-MRAM	VCMA-MRAM

Figure 1.7: Storage Class Memory Technologies for Main Memory.

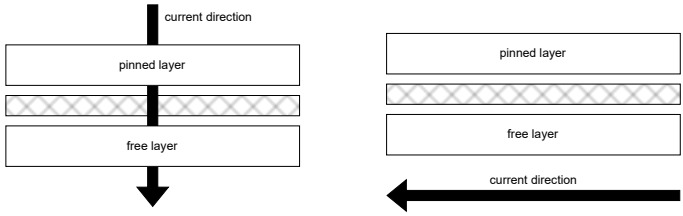
Spin Torque-controlled Magnetic Memory

The phenomenon of magnetism is one of the main alternative methods to store data in contrast to storing electric charge in a capacitor [35]. Magnetic Tunnel Junction (MTJ) is a common mechanism to build Magnetic Random Access Memory (MRAM) devices by storing a value encoded in the magnetic field or the polarization state of the electrons. MTJ-based devices are distinguished by their operation principles and are widely explored in the literature [35, 42–46].

One of the established methods of switching the state of an MTJ device is through the spin torque effect [44, 47]. Spin is defined as the alignment of the angular momentum of electrons to a certain direction (e.g. up or down) [48]. To achieve this, electric current is passed through two ferromagnetic layers separated by a thin non-magnetic layer [47]. As electrons pass the first ferroelectric layer (i.e. pinned layer), they become polarized, which affects their spin direction. The polarization is retained through the non-magnetic layer, and the current (electrons) apply torque to the second ferroelectric layer (i.e. free layer) that is dependent on their spin direction.

This corresponds to the STT effect, where the current is injected into MTJ perpendicularly, as shown in Fig. 1.8a. The direction of the current is the key difference of STT-MRAM to its counterpart - SOT-MRAM, shown in Fig. 1.8b where write current is injected in-plane with the MTJ.

While STT-MRAM offers the expected benefits of magnetic memory over the drawbacks of DRAM, it suffers from issues of its write mechanism [44, 47]. Faster writing speeds require increased current, which results in the material breakdown of the MTJ and reduces the endurance of the memory device. As mentioned, the expected high utilization rate of the main memory is not compatible with endurance concerns and it is one of the main reasons that limits the adoption of some SCM-M technologies. There is also an inherent limit to how fast an STT-MRAM device can switch [47]. SOT-MRAM is posed to address these concerns. In the early iterations of the technology it demonstrated more than



(a) Write current injected perpendicular to the MTJ.

(b) Write current injected in-plane with the MTJ.

Figure 1.8: Spin-transfer (left) and spin-orbit (right) torque mechanisms.

$5x10^{10}$ switching cycles with latency on the order of 10^{-12} seconds [49].

To eliminate the drawbacks of STT-MRAM, the read and write paths are separated in SOT-MRAM. While the read path remains perpendicular, similar to STT-MRAM, the write path is in-plane with the MTJ. The STT approach allows for a single access transistor, but SOT requires two access transistors for two separate mechanisms. It can be seen that tradeoffs are unavoidable - SOT-MRAM trades density for improved endurance and latency.

Voltage-controlled Magnetic Anisotropy

Another recent development in magnetic devices is VCMA. This method uses voltage to write data into the MTJ and enables faster state switching [42, 43, 46]. VCMA-MRAM also allows for higher bitcell density as it eliminates the circuitry to drive current into each MTJ.

The conceptual diagram of a VCMA-based MTJ, as reported in [43], is shown on the left of Fig. 1.9. The right side of Fig. 1.9 depicts the voltage-dependent magnetization state curve of the device. Depending on the voltage applied at the terminals of the MTJ, it switches between the parallel and anti-parallel states ("P" and "AP", respectively). Access to the VCMA-controlled MTJ can be enabled using a transistor (1T-1MTJ) or a diode (1D-1VCMA) [42, 43].

One of the drawbacks of VCMA-MRAM is high read latency due to the high MTJ resistance. Authors of [42] have shown that read circuitry modifications could improve read performance by up to $\sim$260%. Due to this inherent limitation, however, VCMA-based magnetic memory would be more suitable for systems that execute write-intensive workloads.

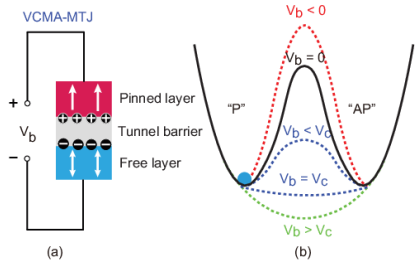


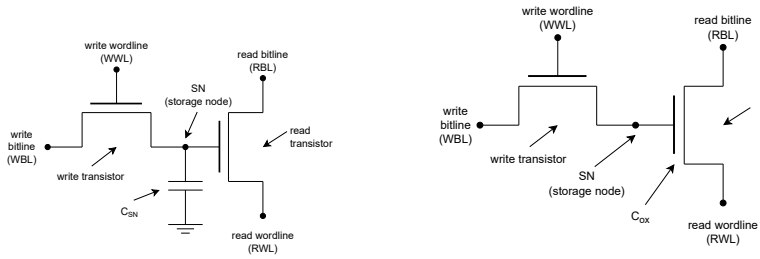
Figure 1.9: VCMA bitcell. Source [43].

Indium-Gallium-Zinc Oxide (IGZO) DRAM

Lastly, as another alternative to resolve the high leakage current drawback of traditional DRAM, researchers have been exploring $InGaZnO_4$ or IGZO thin-film transistors [40, 41]. The resulting device exhibits considerably lower leakage current and higher retention time than DRAM. Although IGZO-based DRAM was listed as an SCM-M device, it is an exception due to its volatile nature and could be considered as an improvement of DRAM technology.

A capacitor-based design with two access transistors (2T1C) [40] is shown in Fig. 1.10a, which is naturally a volatile device. An IGZO-DRAM that does not use capacitors - 2T0C IGZO-DRAM [41], is shown in Fig. 1.10b. Despite not having a separate capacitor, this design is also volatile because it relies on the capacitance of the read transistor gate oxide (i.e. C_{ox}) to store data, which is still susceptible to charge leakage. Both devices significantly outperform the refresh frequency of traditional DRAM and 2T1C IGZO-DRAM can exhibit longer retention time than its counterpart. The need for two separate transistors for read and write operations, however, raises concerns over the density of an IGZO-DRAM memory array to achieve storage capacity equivalent to DRAM.

In conclusion, no single memory technology has yet been identified as a successor to DRAM. Instead of replacing it, an alternative solution could be to reduce the DRAM footprint in the system and deploy a complementary memory technology in parallel. The resulting hybrid main memory is an architectural approach to resolve the technological challenges of the memory wall. This work continues by exploring hybrid main memory design, associated challenges and solutions.



(a) 2T1C IGZO-DRAM bitcell design. (b) 2T0C IGZO-DRAM bitcell design.

Figure 1.10: Bitcell diagrams for IGZO-based DRAM.

1.3 Memory Systems

Given the shortcomings of individual memory technologies, system designers explore architectural solutions to mitigate them and address the memory wall. Heterogeneous and hybrid memory systems can be beneficial by exploiting the tradeoffs of combining and accessing one or another memory region or memory device. Examples of both approaches are briefly introduced here, with a focus on the hybrid approach followed by this work.

1.3.1 Heterogeneous Memory Systems

In general, memory heterogeneity is not a new concept and has well-established examples. Non-Uniform Memory Access (NUMA) is a multiprocessor system architecture with local and remote memory regions [50, 51]. In systems with more than one processor interconnected via a shared data bus, a memory tier such as the cache, is physically local to one processor and remote to the other processors. An example design is shown in Fig. 1.11, however, exact implementations vary. The difference between accessing the local and remote memory is accounted for by software or hardware (e.g. the operating system or the data bus controller). NUMA systems typically benefit workloads that exhibit a high degree of locality and low data contention, when data is already present in the local memory region and it is not accessed by the other processors.

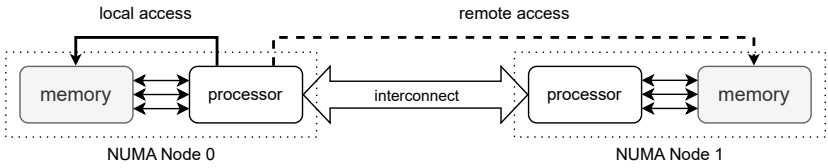


Figure 1.11: Local and remote memory accesses in a NUMA system.

1.3.2 Hybrid Memory Systems

A hybrid memory system would combine different memory technologies within the same memory hierarchy tier, since the advantages of one technology could compensate for the drawbacks of another. The resulting system has two or more different memory technologies on the same access tier, in the same physical location, and possibly on the same electrical interface (see Fig. 1.12). The latter part is not always possible due to emerging memory devices having different peripheral circuitry and physical design, as it could be seen during technology exploration in Section 1.2. Examples of hybrid memory systems can be found in the recent works exploring hybrid on-chip cache and storage tiers [52–55].

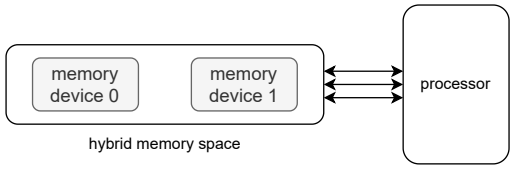


Figure 1.12: Different memory devices in a shared hybrid memory space.

In summary, hybrid memory can be a viable option when a single technology does not satisfy the demands of the system. It can be applied to various memory hierarchy tiers, including the main memory, as will be explored next.

Hybrid Main Memory

The previous discussions on main memory challenges, memory technologies, and memory system design lead up to the introduction of *Hybrid Main Memory (HMM)*. It is a combination of DRAM and SCM-M devices with complementary characteristics - in other words, it promises increased main memory capacity at a cost lower than an equivalent DRAM-only configuration with minimal

performance and energy consumption penalties. The challenge of HMM is two-fold: (1) the physical design and (2) the management policy. A brief summary of these challenges, relevant literature, and the identified research scope are outlined next. Section 1.4 follows and expands on the design and exploration of HMM as a research problem, and proposes the solution presented in this work.

- **Physical Design**

The two challenges are interconnected - the choices of device hierarchy and technologies, communication protocols, electrical interfaces, etc. dictate the potential improvements in energy consumption and performance, as well as the costs [39, 56]. Once the physical design of HMM is defined, it sets boundaries on the most common criteria used for system evaluation - performance and energy consumption. Therefore, it is necessary to develop a management policy that will maximize the chosen evaluation criteria within the set boundaries. The policy should be designed to satisfy an optimization goal by finding the tradeoff point between using one memory device or the other.

- **Management Policy**

The goal of the management policy might be to maximize performance, minimize energy consumption, or to strike a balance between the two. As it has been mentioned earlier, SCM-M and DRAM are mostly complementary in their characteristics, meaning that access latency and access energy for HMM devices are often inversely proportional. More specifically, the goal of the management policy could be expressed as the total time it takes to complete a workload, how much total power the system has drawn, or what the best access latency is for a given limit on energy consumed.

Then, to achieve the desired goal, the policy must execute its functions - that is, to make data available without additional delays and to reduce unnecessary data movement. Recent studies show that data movement could account for up to 60% of total system energy consumption and achieve, on average, 55% reduction in total energy consumed by the system [26]. This means that while increased main memory capacity (both homogeneous and hybrid) can reduce the data movement *across* the memory hierarchy tiers - to and from storage - the hybrid main memory management policy must be optimized to minimize data movement *within* hybrid memory space too.

- **Existing Literature**

As it will be explored further in Chapter 2, a considerable part of the existing literature on HMM focuses on data migration policies between the HMM devices contrary to the observation on data movement (e.g. [57–62]). The goals of these proposals are (1) to identify data access patterns, (2) to relocate time-critical

data to the device with low access latency, and (3) to relegate the least frequently used data to the device with lower energy consumption. These policies are implemented to search for memory access patterns that would best match the characteristics of the memory device per the expectations of the designers.

However, migration and pattern analysis are complex and resource-intensive tasks, exacerbated by the workloads with heavy memory utilization. Data and the correlated access to a memory device are separated by several levels of system hierarchy - such as the application, data objects, compilers, operating system memory management, virtual and physical address spaces, memory pages, etc. [5]. As a result, analyzing memory accesses can be done at various levels with varying degree of granularity, design efforts, and overhead. This will be further discussed in Section 2.3 with the categorization of existing literature into system-, task-, object-, and page-based methods.

- **Identified Research Gap**

It follows that to reduce the burden of runtime memory access analysis and the impact of data migration, it is necessary to consider the initial placement of data. The existing proposals often discount this aspect of HMM management and place data in one of the devices by default or at random, to later analyze for patterns and migrate as needed. In this work, *the lack of optimized initial data placement* is identified as an overlooked aspect of HMM and this thesis is devoted to exploring this research gap and proposing a solution to it. To achieve this, the research scope around HMM design is discussed next.

1.4 Design and Exploration of Hybrid Main Memory

As mentioned, the challenge of HMM is in determining the device hierarchy and deriving a data placement policy. Following this definition and observations from the review of the existing literature on HMM, the following four questions are proposed to determine the scope of HMM design and related research:

- *What memory technologies to use?*
- *What is the internal organization hierarchy?*
- *What is the placement of the HMM arbiter in the system?*
- *What memory device to use for a given memory access?*

The first three questions are briefly discussed in Section 1.4.1, and the last question is introduced in Section 1.4.2. Discussion around it is summarized in Section 1.4.3 and expanded further in Chapter 2 with a literature review.

1.4.1 Physical Design

The first three questions are concerned with the physical design of HMM and are an active research topic [33, 36, 39, 40, 46, 56, 63]. Based on the observations from the literature, a summary of possible answers is presented next.

What memory technologies to use?

First, the choice of memory technologies is subject to availability and budget, and the potential candidates were introduced earlier as SCM-M devices. The motivation to deploy HMM in modern systems is to reduce DRAM footprint while offering sufficiently large main memory capacity [27]. Hence, the options that have challenges in bitcell density (e.g. STT-MRAM or IGZO-DRAM) may not the most suitable candidates as a *complementary* memory technology.

What is the internal organization hierarchy?

Second, the device hierarchy options (hierarchical vs. flat) are shown in Fig. 1.13. It has been explored in the academic literature (e.g. [63–67]) and commercial products such as Intel Optane that can be used in "Memory Mode" and "App Direct Mode" [68]. The first mode corresponds to a hierarchical HMM (Fig. 1.13 left) which is more beneficial to time-critical workloads and when the device performance discrepancy is significant. This design often uses cache-like management policies (e.g. prefetching, evicting old data, etc.) [64, 66, 67]. It eliminates the third question because it does not require the system to make a choice between the HMM devices. It also has an immediate answer to the fourth question by using DRAM as the first point of access. This masks the latency of slower devices and minimizes the impact on performance, since data is moved from DRAM to SCM-M in the background.

A direct access to SCM-M, bypassing DRAM, could be more beneficial than hierarchical HMM in scenarios where the memory devices have similar access latency, the accessed data is not time-critical, or the data has characteristics that correlate to the target memory device, satisfying the expected optimization goals. A flat HMM hierarchy (Fig. 1.13 right) enables such independent access to all memory devices and corresponds to the "App Direct Mode" of Intel Optane mentioned earlier. However, answering the third and the fourth questions is no longer as straightforward as using cache-like management policies.

For both HMM designs, exposing memory heterogeneity to the higher abstraction levels of the system may be beneficial due to better insights on how and when a

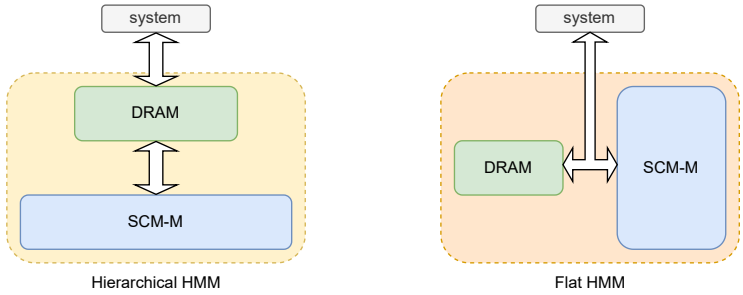


Figure 1.13: HMM organization: hierarchical vs. flat.

workload is accessing a particular block of data. Conversely, abstracting HMM and presenting the memory space as homogeneous enables seamless integration. This discussion leads to the next question on HMM design.

What is the placement of the HMM arbiter in the system?

The third question is mainly concerned with flat HMM. By design, it allows the system to access either device equally, unlike hierarchical HMM that must access DRAM before SCM-M device can be used. Since flat HMM implies that memory devices can be accessed independently, it is necessary to augment the memory management and introduce *an arbiter*. A decision has to be made on where data will be placed, and what characteristics of that data will be used to make that decision. The arbiter is the decision-making agent, the design of which is one of the main contributions of this work. The placement of the arbiter for flat HMM will expose or abstract the memory heterogeneity from higher levels of abstraction in the system - i.e. the workloads, operating system, etc. Fig. 1.14 visualizes the potential arbiter locations.

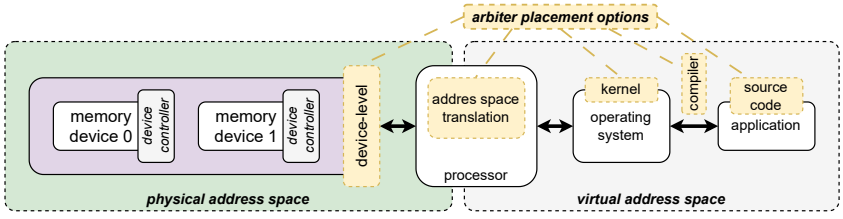


Figure 1.14: HMM arbiter placement options.

If flat HMM is abstracted from the system, then arbitration has to happen within the domain of the main memory rather than within the domain of the rest of the system. The choice still remains whether to place the arbitration logic together with the memory devices (e.g. adding a unified memory controller) or to place it within the system boundaries. As it could be seen in Fig. 1.14, the system is divided between the physical and virtual address spaces. The physical memory space covers the memory devices and the virtual memory space is where the workloads and the operating system reside [69–71]. As it will be discussed in detail later in Chapter 3, the border between these two regions could serve as a convenient placement for the arbiter (e.g. processor-level memory management [70]). The HMM is exposed if arbitration becomes the burden of the workloads (e.g. explicitly specifying the target for each data object) or the compilers and the operating system (e.g. memory allocators). As an example, in the "App Direct Mode" of Intel Optane, the workloads and the operating system are explicitly aware of the hybrid main memory space.

In the case of a hierarchical HMM, the main memory appears to the system as homogeneous already, just as the DRAM-only configuration. This allows the system to retain the existing main memory management methods and policies. Although, the choice between software or hardware cache-like management methods still remains [67]. While it is possible to expose the heterogeneity and explicitly instruct the system to place or move data to the SCM-M device, this is not possible without accessing DRAM first. This could be considered as the main shortcoming of hierarchical HMM.

Following the discussion on the placement of the arbiter, its decision-making logic must be defined based on available insights and is explored next.

1.4.2 Management Policies

Based on the previous discussion, the remaining aspect to consider is the design of the decision-making logic of the HMM arbiter. Main memory management is a broad function of the system: it is responsible for allocating dedicated regions of the memory space to workloads, managing the translation from the virtual space in which workloads operate to the physical space in which the memory devices are located, and other tasks such as queuing and scheduling the memory accesses, etc. [5, 70–76]. HMM arbiter is an extra function for memory management and the earlier question is rephrased as follows:

- *When to use one device or another?*
- *What criteria should be used to make that decision?*

As an example, if the goal is to maximize performance, then a common assumption is that frequently accessed data should be placed in a memory device with low access latency. If data is accessed infrequently, it should be moved to the device with higher energy efficiency for long-term storage. Overall, HMM management mechanisms are often complex and limited in scope to maintain the design and implementation feasibility. Analyzed attributes commonly include access frequency, type, priority, data object size, and device specifications. The scope of various attributes of memory accesses that could be selected for analysis is reflected by the number of works referenced and reviewed later in Chapter 2 [27, 58–63, 65, 66, 77–107].

1.4.3 Literature Observations

Two main observations derived from the analysis of the existing literature on HMM management in Chapter 2 are presented below as L.O.1 and L.O.2:

L.O.1: *the existing policies designed for HMM management exhibit a lack of non-biased and flexible decision-making*

L.O.2: *the existing policies designed for HMM management use naïve initial allocation policies and rely on data migration*

- Device arbitration in the proposed works follows certain expectations on how access characteristics of a data block would relate to the memory device characteristics. Following a rigid logic approach in this process leads to biased policies, where DRAM is expected to be the better performing device, even though this might not always be true [108, 109] and will be explored in the discussion of the results of this work.
- Flexibility of the management policies refers to the ease of reconfiguring the decision-making logic and optimization targets, either offline or in runtime. The design and implementation of these policies is often tailored to the chosen decision-making logic with its biases, and the resulting management layer is not as flexible anymore.
- Migration of data between the HMM devices is largely unavoidable, however, the decision on where data was placed *initially* plays an important role too. Existing works in the literature tend to use simple (naïve) policies that place data exclusively in one device or the other, without considering the impact of this choice and potentially increasing the number of data migration events.

1.4.4 Research Goals

As a conclusion to the discussion on HMM design, its challenges in arbitration, and to address the related literature observations, it becomes possible to formulate the sought (envisioned) solution. Considering the identified gap in the existing literature, this work envisions an *adaptive policymaker guided by an optimization target* to tackle the challenge of initial data placement in hybrid main memory systems.

This policymaker should learn what memory device to use, rather than operate on assumptions. Hence, its decision-making logic should be more flexible to avoid following a rigid rule, such as matching a memory access with an identifiable pattern to a memory device with particular characteristics. Instead, it would follow an optimization target that guides the learning process and can be redefined as needed. The impact of the optimized initial data placement should be reflected in the reduced burden of data migration and improved runtime for workloads executed on an HMM system.

The following research goals (R.G.) summarize these expectations:

> **R.G.1:** *to design an adaptive policymaker guided by an optimization target to decide on which memory device will be used*

> **R.G.2:** *to optimize initial data placement, reducing the burden of subsequent data migration during the execution of a workload*

> **R.G.3:** *to improve workload runtime in a system with hybrid main memory compared to the existing management policies*

Stated research goals #1 and #2 correspond to literature observations #1 and #2, respectively. Research goal #3 is a criterion to evaluate the proposal to address the observed literature shortcomings and the research gap.

1.4.5 Research Proposal

To achieve the listed research goals, this thesis proposes, designs, and evaluates **Page Arbitration using Reinforcement Learning (PARL)** for Hybrid Main Memory systems. It is a reinforcement learning (RL) agent designed to meet the set research goals. It exhibits characteristics which will be shown to be

advantageous over the existing solutions that are based on rigid decision-making logic. The observed characteristics of PARL are as follows:

1. *agnostic of the memory configuration and devices*
2. *placed at the border of address spaces for transparent integration*
3. *reduced RL state space for implementation feasibility*
4. *configurable optimization target using reward functions*
5. *adaptive to various workloads and memory system configurations*

The first three points are discussed in Chapter 3. The remaining two are derived from the evaluation in Chapters 5 and 6. A brief summary and observed correlation to the research goals is given below:

- PARL is agnostic of the workloads and available memory devices. As a reinforcement learning agent PARL chooses a device and receives feedback based on how it impacted the optimization target specified by the reward function. Through an iterative learning process, PARL builds up experience and finds the optimal initial placement for data in the hybrid memory space that leads to the lowest runtime for a given workload. *This characteristic of PARL corresponds to R.G.1 and R.G.3.*
- The location of the arbiter, as discussed earlier, is chosen such that its impact on the system is minimized and HMM is abstracted from the workloads. This corresponds to the mechanism that translates addresses from the infinite virtual space to the limited physical memory.
- To reduce the complexity of the arbiter, the initial data placement in HMM is approached as a resource utilization problem, rather than a developing another method of memory access analysis. PARL is designed to observe the state of system resources (e.g. available memory capacity, queued requests, etc.) instead of tracking the individual patterns and characteristics of the memory accesses (e.g. repetitions, locality, etc.). As a result, the number of the environment states is reduced significantly.
- PARL does not always choose the same device for the same type of identifiable memory access pattern as fixed-logic policies. Its goal is to use the memory device readily available, rather than the one with nominally lower access latency. This mitigates the scenario where the "faster" device has too many pending requests to process and becomes a bottleneck, even though the "slower" device was available. *This characteristic of PARL corresponds to R.G.1 and R.G.3.*

- PARL is designed to learn a better initial data placement strategy without involving data migration. While doing so, it delivers improved runtime over existing fixed-logic policies for initial placement, as well as matches and outperforms proposals that are based on migration policies, showing that there is potential to reduce the frequency of data migration. *This characteristic of PARL corresponds to R.G.2 and R.G.3.*

In summary, it can be seen that PARL answers the research goals set, and addresses the shortcomings of the literature. PARL has a non-biased and flexible decision-making logic, achieving improvements in workload runtime by up to 43% better than fixed-logic policies. It does so without matching memory accesses to devices following a predetermined logic, and instead responds to the state of the system and improves the utilization of memory resources.

1.5 Thesis Structure

This work, titled "PARL: Page Allocation in Hybrid Main Memory using Reinforcement Learning", has been published in the Journal of Systems Architecture (JSA) [110], and the manuscript has been incorporated into this thesis throughout Chapters 3, 4, 5, and 6.

Next, Chapter 2 presents an overview on machine learning and its application in memory management, as well as a summary of the relevant literature on HMM management. Chapter 3 outlines the design principles behind the proposed reinforcement learning agent for initial data placement. It also includes design overviews of the relevant works that will be evaluated together with PARL. Chapter 4 introduces the configuration of the experiments, the agent, and the system, as well as the workloads and how they are evaluated. Chapter 5 and Chapter 6 present the results of the experiments and derive conclusions based on the observations. Chapter 7 concludes this work and outlines possible improvements and extensions. Supporting material is presented in the appendices, including preliminary exploration of two potential follow-ups.

Chapter 2

Background and Related Work

"The real question is not whether machines think, but whether we do." [2]

This chapter discusses the literature on HMM management and decision-making approaches to highlight the importance of matching the control agent to the nature of the task. First, two major types of decision-making are briefly introduced, i.e. *heuristics* and *machine learning* with its subtypes.

Then, Section 2.2 expands on previously mentioned components of memory management: address spaces and pages. The proposed categorization of the literature based on system abstraction levels is introduced here too. Section 2.3 follows up with descriptions of works in each category. These two sections provide the background information on the HMM management task and highlight the focus on page migration policies with rigid rule-based logic.

Section 2.4 combines the decision-making approaches with the related works. Advantages and disadvantages are discussed, highlighting the flexibility of machine learning but with the consequences of mismatching the chosen method to the intended task. Applicability of reinforcement learning is discussed as an alternative approach to decision-making in the HMM context.

A summary section closes the discussion on the design of control agents for HMM management so far, combining the goals from Chapter 1 and the supporting material from this chapter, and, ultimately, leading to Chapter 3 on the design of the proposed page-level HMM arbiter using reinforcement learning.

[2] This quote is commonly paraphrased from the works of B.F. Skinner on applying and developing formal techniques for reinforcement learning in education [111, 112].

2.1 Overview of Decision-making Approaches

To design the control agent for HMM as it was proposed previously, it is necessary to define its decision-making logic. In computing systems, and in general, decision-making can be based on intuitive rules using fewer attributes (heuristics) or use algorithmic methods that consider several attributes simultaneously (machine learning) [113, 114]. The concepts and implementations differ, but they all strive to achieve the desired optimization metric.

2.1.1 Heuristics

Decision-making is extensively studied in various contexts ranging from human psychology to computers [113, 115, 116]. One of the methods to arrive at a decision is to use intuitive shortcuts or generalizations derived from the experience of a well-studied task. This is referred to as heuristic(s) and based on their survey of literature on this topic, psychologists Gigerenzer et al. provide the following definition: "*strategies that ignore part of information to make decisions more quickly than complex methods*" [115]. In the context of computing systems, heuristics-based decision-making could be summarized as a set of rules based on selected attributes and the intuitive expectation of their impact (or assumptions) [113, 116, 117]. The decision-making logic of such control agents is more interpretable, as it follows a much shorter cognitive path from the input data to the output decision [115–117]. Moreover, their design and implementation can be optimized to minimize resource utilization [117]. On the other hand, assumptions may not always hold true and the rule-based design would be less flexible to adapt to different workloads or HMM configurations.

2.1.2 Machine Learning and Subtypes

Other methods to make decisions, according to [115], are based in formal logic and statistics. In the current context this corresponds to algorithmically derived and programmable logic, such as Machine Learning (ML) [114, 118, 119].

ML is broadly defined as data-driven mathematical models that can be trained, or taught, to operate on data sets larger than control agents designed by hand [114, 118–120]. A large number of (interdependent) parameters in memory systems may be better suited for an ML-based control agent than to a heuristics-based one. An ML-based HMM arbiter could make its decisions by finding correlations between data access patterns and the characteristics of the devices that might have been overlooked or omitted for simplicity of manual design.

A shortcoming of ML methods is that their implementation could require more resources, especially if the chosen method is mismatched to the task [114, 121, 122]. Therefore, it is important to understand the nature of the task, the desired properties of the control agent, and the nature of the chosen ML method.

Fig. 2.1 shows the main ML subtypes: Supervised Learning (SL), Unsupervised Learning (UL), and Reinforcement Learning (RL), as introduced next.

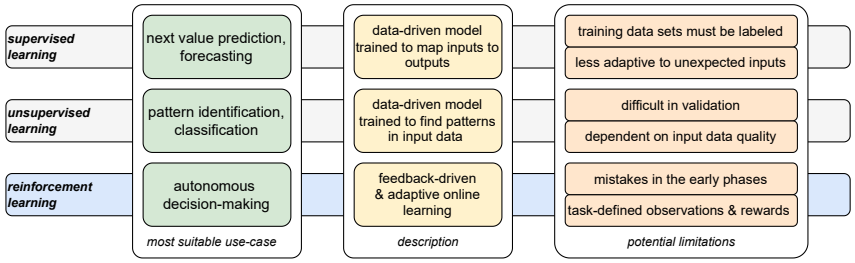


Figure 2.1: Classification of Machine Learning Methods.

(Un)supervised Learning

SL and UL are the two methods of ML that work with datasets, which is conceptually different from RL that interacts with its environment. The goal of these two methods is to develop a model, represented as a weights matrix, which will map the input data to the output. The tuning of the weights to achieve the desired output is referred to as training. During this process the generated output is mathematically compared to the expected correct output to quantify its deviation (error). This value is then used to adjust the weights in the matrix until the error is sufficiently minimized [114, 118, 123, 124].

The main difference of SL to UL is that the desired output is known, which means that the input dataset is labeled (often by humans) [114, 118, 123]. Due to this, SL is used for identifying a previously learned object or a feature of an object in a new unlabeled dataset - known as inference. By training a model on various pictures labeled as a dog, for example, that model will be able to predict, with a degree of certainty, whether a new input image is a dog or not. Generally, the greater the variety of the training dataset, the more accurate will be its response to an unlabeled image. It will not, however, be able to identify what the non-dog image represents - i.e. to label a new input image as a cat, since it has not been trained on data labeled as cats.

In short, SL models are predictors and the goal is to achieve high prediction accuracy [123]. After training an SL model has to be validated with a new dataset before it can be deployed for its intended application. To update the model with new labels or datasets, it needs to be trained and validated again.

On the other hand, UL, by definition, is trained without supervision. Therefore, it is more useful in clustering entries with common features in an unlabeled dataset [114, 118, 124]. The main challenge is that there is no measure of its correctness, as is the case with SL. An outside observer has to judge on the relevance, quality, and accuracy of the clusters found [124]. UL is applicable in the fields that generate large volumes of unsorted data and it is beneficial to automate the initial analysis. Common examples include healthcare [125], network traffic analysis [126], urban planning [127], etc.

Reinforcement Learning

Reinforcement learning is conceptually different from the other ML methods. Instead of learning by example, RL is a goal-driven agent that acts to gather experience and learn. Its goal is to derive the most optimal behavior (actions) towards a specified goal by interacting with its environment and receiving feedback [119, 128–130].

RL is commonly applied to control systems where the impact of a decision is deduced from observations on the state of that system [130, 131]. It is useful when action-impact correlation is not a rule, as in the heuristics-based controllers. An overview of an RL agent and its environment is shown in Fig. 2.2.

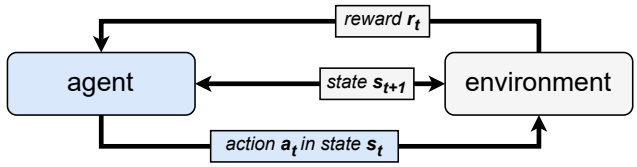


Figure 2.2: Interaction of an RL agent with its environment.

The agent interacts with its environment in discrete time steps t and receives a reward r_t in the next time step $[t+1]$. The rewards establish a correlation between the action taken a_t and the observed environment state s_t. While the state-action pair belongs to the same time step - $[s_t, a_t]$, the next state s_{t+1} is also determined when the reward for the previous $[s_t, a_t]$ pair is received. This allows the agent to calculate the impact of its actions using the equation (2.1).

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma[Q_{max}(s_{t+1})] - Q(s_t, a_t)] \qquad (2.1)$$

The individual terms of this equation, referred to as the Q-value update function, are explained below [119, 130–138]:

- The main term Q is the quality value calculated for every $[s_t, a_t]$ encountered. The Q-value tells the agent which action, in a particular state, is more favorable to its long-term goals.
- Reward r_t describes these goals or the metric of the environment to be optimized. The agent aims to maximize its cumulative rewards - implying that in a given state an action with a lower reward could lead to greater rewards in the following states. This perceived benefit of an action is what the Q-value represents.
- γ (gamma) is a design parameter of the agent, known as the discount factor. It is used to bias towards immediate or future rewards by scaling down the maximum future Q-value. Allowed values are $\gamma \in [0, 1]$, and the smaller values prioritize immediate rewards for short-sighted planning. Increased γ considers the impact of the future possible Q-values more for a far-sighted strategy. In practice, $\gamma \approx 1$ to consider short-term effects but with an emphasis on long-term planning.
- By subtracting the Q-value of the occurred state-action pair, the agent calculates its deviation from the maximum and guarantees that there will be convergence to that maximum. The total reward is thus the sum of the received reward r_t and the discounted future reward, error-corrected by $Q(s_t, a_t)$ due to the concept of temporal difference learning [119, 133].
- Lastly, α (alpha) is the learning rate of the agent. It scales the total reward to specify how quickly the new Q-values should be adopted. Higher α causes the agent to strongly influence its existing experience, potentially causing instability in the Q-values, while lower α ensures a smoother, albeit slower learning process. $\alpha \in [0, 1]$ is allowed but can be often constrained to $\alpha \in [0.01, 0.1]$ or set to 0.1.

The remaining design parameter is the exploration rate ϵ (epsilon). Without pre-programmed knowledge, experience is gathered by exploring the environment. The question is whether the agent should explore and gather experience, or exploit it. An exploration rate aims to resolve this *exploration-exploitation dilemma* of decision-making, also found in various contexts [119, 139, 140].

A common strategy to resolve this dilemma is called decaying ϵ-greedy, where $\epsilon \in [0, 1]$. It gradually decays from active exploration (high ϵ) towards "greedy" exploitation (low ϵ) of gathered experience [119]. To define ϵ and its decay, it is necessary to establish the lifetime of the agent. RL is applied to processes that are episodic (repeatable) in nature, where the agent can learn iteratively.

ϵ is fixed during an episode and decays for every episode following a defined function - e.g. exponentially as shown in Equation (2.2).

$$\epsilon_n = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-n\lambda} \tag{2.2}$$

This function calculates the exploration rate ϵ for episode n, bounded by the minimum and maximum values – e.g. $\epsilon \in [0.01, 1]$ to ensure that the rate is never zero. For every episode, ϵ is scaled by the chosen decay rate λ. During an episode, newly calculated ϵ is compared against the randomly generated exploration rate threshold $\epsilon_{threshold} \in [0, 1]$. This threshold can also be deterministic and controlled. If $\epsilon_{threshold} > \epsilon_n$, the agent will exploit its experience. Thus, the decay of ϵ ensures a gradual shift from exploration to exploitation.

2.2 Relevant Components of Memory Management

This section introduces the fundamental memory management components that are the most relevant to the current discussion.

2.2.1 Address Spaces and Memory Pages

Paging has been a common method to manage computer memory since the early 1960s [69]. They partition the memory space and represent a range of contiguous memory locations, both in virtual and physical address spaces. This allows the system to manage memory more efficiently – instead of locating and accessing an individual address, several are accessed and manipulated at once.

Pages are also useful in matching the unlimited Virtual Address Space (VAS) to the limited Physical Address Space (PAS) of a memory device (i.e. its capacity). Retaining the division of address spaces allows for changes in the memory subsystem without exposing them to the workloads, as shown in Fig. 2.3.

As an example, two workloads are concurrently running in the VAS (purposefully depicted borderless). On the left, their pages are matched to a single memory device by the existing memory management mechanisms of the system. On the right, the arbiter stands in front of the hybrid memory space, thus retaining similarly "homogeneous" address space from the perspective of the workload, while redirecting the requests to the corresponding devices.

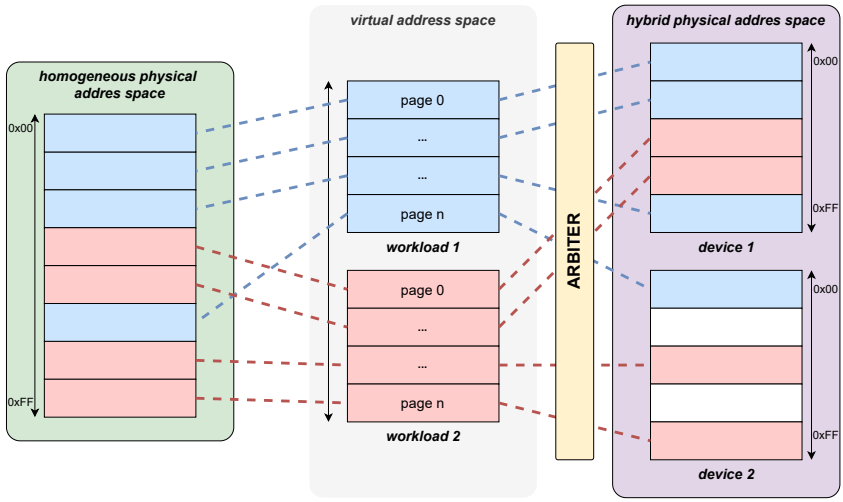


Figure 2.3: Address Translation in Homogeneous and Hybrid Memory.

2.2.2 Page Migration

As it was mentioned in Chapter 1, reducing data movement across and within memory hierarchy levels is important. Page migration refers to the relocation of pages from one HMM device to another. This mechanism is complex and time-consuming, affecting system performance and energy efficiency.

To migrate a page, the system must identify the candidate (i.e. following a migration policy), schedule its transfer (i.e. read from one and write to another device), and update all records on the connected virtual and physical locations. Both memory devices could be unavailable in the process, and the affected workload could be stalled until its data becomes accessible. If some data needs to be evicted to the storage level before migration (e.g. a device exhausting its capacity), then the penalty is exacerbated further.

The impact of migration is significant, and grows even further if data belongs to different workloads. Some studies report that improving the migration mechanism *implementation* by clustering or increasing the size of pages, creating superpages, etc., could reduce the runtime by up to 40% before the migration *policy* is improved [62, 104].

2.3 Related Works in HMM Management

Given that this thesis presents initial data placement as an overlooked aspect of HMM management, the goal of this section is to highlight that most works in this field are migration-driven proposals. Benefits and downsides of various approaches are also discussed, where applicable.

2.3.1 High-level Classification

First, the existing literature on HMM management is categorized from the system perspective, based on what level is used to tackle this task.

The main observation is that HMM and its control mechanisms are designed in accordance with the envisioned position of HMM in the system and memory hierarchy. This choice influences the abilities of the arbiter and how it would achieve its objectives. Implementations of the proposed works include theoretical models, model-based or cycle-accurate simulations, hardware-based emulators, and full system designs. The latter two are not always possible for new technologies, and most works experiment with simulators [141–145]. Four categories are listed in Table 2.1, citing the respective works to be discussed.

Category	**Description**	**Related Works**
system-level	*global, operating system-level*	[65, 77–79]
task-level	*algorithm or application-level*	[80–82]
object-level	*data object, library, compiler-level*	[27, 59, 83–89]
page-level	*memory page-level*	[58, 60–63, 66, 85, 90–107]

Table 2.1: HMM management policies based on the level of abstraction

Another observation is that existing works are similar in their design methodology. First, most works follow a simple initial distribution of data within HMM. Then, a method to analyze memory accesses is proposed (offline or runtime workload profiling), such that a policy for data migration can be derived from it with the goal to minimize migration events themselves. Overall, then, this policy serves the optimization objective set in the beginning.

Naturally, migration depends on and follows the initial placement. They are parts of the memory management mechanism serving the same goals of improving performance, minimizing energy consumption, maximizing device endurance, etc. Among all considered works, only four initial page placement strategies have been identified: *DRAM-first*, *SCM-M-first*, *random*, and *round-robin*. These policies are intuitive in their definition, without considering the impact on the

subsequent migration even though, the goal is to minimize its occurrence and impacts. However, if the data was placed (offline or in runtime) such that it has to be moved later, it defeats the purpose of improving migration policies by increasing migration frequency. If it can be shown that it is possible to place data and avoid its movement later on, whether by observing the states of the system or the data itself, this potential needs to be explored.

A brief overview of the first three categories is given next. The current work falls under the page-based category which is introduced in greater detail.

2.3.2 System-level Proposals

In this category, HMM management is considered as a problem of the entire system [65, 77–79]. In one of the early works in this space, Mogul et al. consider potential design avenues for HMM and propose to manage migration as an operating system (OS) task [79]. The authors of [77] enable virtualized guest and host OS in datacenters to exchange information on the memory accesses. They reduce data migration and distribute the burden of workload profiling across the systems. In [65], the authors focus on the question of HMM hierarchy and dynamically reconfigure the physically flat memory space to logically hierarchical as needed. In [78], the authors present an OS-level hybrid memory management framework covering both the cache and the main memory. Their approach makes decisions based on page hotness and data reuse patterns collected via a runtime monitoring tool.

Despite promising results, these approaches are either tailored to a specific use-case or require substantial design efforts that may inhibit their adoption.

2.3.3 Task-level Proposals

The next category considers scheduling individual applications to a region of the hybrid memory space [80–82]. As a result, the burden of runtime access pattern analysis is either reduced or eliminated. The overhead of profiling new applications, however, is not well-explored. Some proposals are based on analytical models, such as Integer Linear Programming (ILP) [146] in [80]. The scheduling task can be defined as a linear equation with defined constraints that is solved for a chosen target value. While analytical models offer quick insights for evaluation, the gap to implementation may be substantial.

In [81], the authors base their work on a real system and avoid the challenges of a theoretical approach. This method proposes a fixed strategy to minimize the number of write accesses in the SCM-M device and maximize its utilization for

reading data. The authors consider PCM technology as the DRAM counterpart [33], which is susceptible to write-induced device failure [37]. This reflects the technology bias in policy-making also found in other works.

Overall, these coarse-grained task-based proposals could miss access patterns variations within a workload and might not consider the scenarios where workloads outgrow the size of the allocated memory region.

2.3.4 Object-level Proposals

Instead of considering the workload as a whole, its individual data objects and data structures can offer more fine-grained understanding of memory access patterns. This advantage is reflected by a greater number of related proposals [27, 59, 83–89]. The implementations, however, require modifications to the source code of an application, programming libraries, compilers, or memory allocation mechanisms within the OS (as was shown in Fig. 1.14).

A common method is to tag data structures to hint at or to explicitly specify which memory device should be used. In [83], the authors employ static analysis and runtime tagging. They develop an access time and energy model for each memory device and use it to estimate the potential benefits of DRAM against NVM placements. In [85], the authors propose an application-aware method for page allocation.[3] Workloads are first profiled at the compiler level to understand the page hotness and the size of accessed data objects. Then, using a hint-based approach, the data is accordingly placed within the HMM to better serve Graphics Processing Units (GPUs). In [87], the authors develop a runtime programming library for runtime profiling of data object sensitivity to bandwidth and latency. Authors of [89] propose a software-level hint-driven HMM management leveraging various arbiter placements that aligns to the third question on HMM design proposed earlier.

Despite their fine-grained approach to help with data placement, these proposals reduce the transparency of HMM integration, with potential design efforts from individual developers and across the stack.

In summary, the presented categories are active research areas in HMM. However, the chosen system abstraction levels in these proposals may require considerable design efforts and do not satisfy the goals of reducing migration and seamlessly integrating HMM and its control mechanisms. Since the traditional system design already offers a convenient memory space division and operates on memory pages, then designing HMM and its arbiter around them could be less intrusive than within the workloads or across the system.

[3] This method is derived from a page-based method that will also be discussed next.

2.3.5 Page-level Proposals

The benefit of the page-based methods corresponds to the number of proposals that approach HMM management from this perspective, including PARL [58, 60–63, 66, 85, 90–107, 110]. The major point of distinction of PARL is that it focuses on the *initial page placement* to exploit its potential for improvement and to minimize migration, as also mentioned in some works [63, 85].

Overall, these works retain similarities to the previous categories in their choices of tracked attributes and derived migration policies. The CLOCK page replacement algorithm and its derivatives are often used in both homogeneous and heterogeneous memory systems, e.g. [60, 90, 94, 147]. In [98], AC-CLOCK is proposed as a migration policy to mitigate the endurance limitations of the PCM technology based on the intensity of write accesses – similarly to [81]. Thermostat [96] is a method proposed to separate data objects belonging to the workload or to the system through page-level analysis – showing that pages are also viable to derive fine-grained insights similarly to data objects. By tracking workload page hotness and migrating pages to and from the device with lower access latency, the authors strive to improve system performance.

Previously mentioned in the object-based category, the authors of [85] have also proposed a method for page placement for GPUs in HMM systems. Here, pages are allocated at a ratio that matches the aggregate bandwidth ratio across all memory devices in the system. The goal is to saturate the memory bandwidth to better serve the highly parallelized nature of GPUs. This fixed-logic bandwidth-aware strategy is in contrast to their application-aware approach, as it does not consider if pages are placed in devices that severely impact latency or energy consumption due to their access intensity. Additionally, this placement benefits workloads specific to GPUs, while general-purpose processors execute a variety of workloads that are not necessarily parallelized in their computations and data accesses. Although, the authors do report a reduction in the need for data migration and runtime memory access analysis – this aligns with the observations and expectations of this thesis.

UH-MEM [97] considers the utility of migrating pages between memory devices based on access latency, energy consumption, and capacity. The authors derive a model to estimate the impact of these factors based on their assumptions about the correlation of these attributes to the memory device characteristics. Despite this heuristic-driven model, the similarity of *estimating* the impact of decisions and PARL *learning* the impact of its decisions is notable.

The authors of Address Pattern-Aware Memory System (APAMS) [63] approach HMM design as a whole, following the four questions proposed in this thesis. They cover the HMM design space and explore the choices of the architecture,

including the impact of emerging memory technologies. They propose a page placement policy based on analyzing memory accesses for spatial locality. The fixed logic of decision-making in APAMS assumes a match between random or sequential access patterns and device specifications. However, there are various possible logical definitions of what is "random" or "sequential", based on the distance between the accessed locations. Moreover, the obstructed view into the internal organization of memory devices means that these definitions are not guaranteed to work in all cases. Nonetheless, APAMS is reproduced and included in the evaluation of PARL as one of the only dedicated works to consider initial page placement.

Lastly, only few works have been identified as applicable to flat HMM and using ML. Kleio and its follow-up study [58, 107] train a supervised ML model to be used for page scheduling and making decisions on subsequent migration [148–150]. They leverage the history of access rates of a selected high-priority page, and train a model to predict the hotness in the next scheduling interval. Frequently accessed (hot) pages will be migrated to DRAM for faster access and the cold ones will be demoted to SCM-M for energy efficiency. This shows how SL or UL need to be complemented by heuristics-driven decision-making logic, even though ML was used to generate the insights.

Q-prefetcher [66] is another work combining ML and HMM, albeit in its hierarchical version. Here, the authors propose an RL agent to select one of the implemented data prefetchers instead of analyzing accesses and learning a new prefetch policy. Prefetching is a mechanism to preemptively move data, e.g. from SCM-M to DRAM in a hierarchical HMM, and is often used in cached systems [151]. Q-prefetcher is based on the observation that one type of a prefetcher does not suit all phases of a workload. They correlate these phases to the intensity of memory accesses using two observable attributes: last-level cache misses per thousand instructions (MPKI) and the hitrate in the DRAM device. A wrong prefetcher will cause an increase in both and negatively impact the performance of the workload. The agent can then learn to select a prefetcher that will minimize the spikes in these two metrics.

In summary, page-level mechanisms are well-established in memory management, and their relevance to HMM is equally high. Page-level management satisfies the goal of transparently integrating HMM and fits with the address space separation, however, referenced works still tend to focus on migration.

Overall, it can be seen how across all categories proposals are mainly rule-based control agents and therefore, the decision-making logic needs further exploration and discussion to match it with the intended task of HMM management.

2.4 Decision-making in HMM Management

Heuristics are the dominant method for the decision-making logic of the control agents in HMM and are summarized in Section 2.4.1. Furthermore, ML is underrepresented in this space, and using RL in HMM has not been extensively studied either. It will be shown in Section 2.4.3 that one of the reasons for this is the mismatch of the task definition to the underlying principles RL.

2.4.1 Heuristics for HMM

Reviewed works show that their logic is often based on the *expected* (assumed) conditions under which a memory device would maximize its nominal characteristics. These could be the access latency, bandwidth, endurance, energy consumption, etc. It follows that data accesses could be analyzed for access patterns that correspond to these characteristics - e.g. locality, hotness, type, repetitiveness, etc. Finding a match between the two would simplify the choice for the placement of that data. Then, the remaining challenge is to develop efficient (e.g. faster or less resource-intensive) methods to identify these patterns. While the implementation details are specific to the individual proposals, the current discussion is on their reliance on heuristics.

- **Access Intensity**

It is intuitive to place a frequently (repetitively) accessed data block in a memory device with low access latency. This can be achieved if that data block can be profiled as frequently accessed, either by static (offline) analysis, runtime profiling, or through a hint provided by the application developer. Examples of using access frequency or recency as the basis of arbitration logic can be found in [78, 83, 85, 87, 96]. Other works are based on a clock-like rotating pointer to the most recently used location and the data block stored there, to be replaced when they are outdated [90, 98, 147]. Time-dependency can also be tracked by first-in, first-out (FIFO), least recently used (LRU), and similar methods.

- **Access Type**

Another heuristic for data placement is its physical impact on the device. Some memory technologies, e.g. PCM or STT-MRAM, could exhibit endurance issues related to the intensity (frequency) of write operations in a specific location range in the memory space (i.e. temporal and spatial locality, respectively). In [37], the authors track write accesses to minimize last-level cache writeback to PCM-based homogeneous main memory. Example proposals for HMM using this heuristic can be found in [81, 90, 98].

- **Access Locality**

Furthermore, policies that use memory locations (addresses) as a heuristic for decision-making face the challenge of determining what is considered as spatially local. While there is a uniform PAS for the hybrid memory system, as referenced in Fig. 1.14 and Fig. 2.3, it is still a logical convention. The information on the internal organization (i.e. arrangement of the memory arrays) of the memory devices is proprietary to the manufacturers [152–155]. The true spatial locality of data is not known to any other system component. Several works explore how addresses are mapped to the real physical space, both explicitly and implicitly [152, 154–159]. Some propose circuit-level changes to the memory devices, while others approach it from higher levels of system hierarchy. As a result, HMM arbitration policies based on spatial locality in the logical PAS may have limitations in their effectiveness and accuracy. An example of spatial locality heuristics not holding true under different workloads is found in [63].

In summary, heuristics are effective when the presumed correlation of tracked data attributes to memory device characteristics holds true, as well as when the workloads and memory configurations are unchanged. This may not always be guaranteed for large-scale general-purpose systems executing various workloads. Thus, more nuanced methods of decision-making need to be applied to HMM.

2.4.2 (Un)supervised Learning for HMM

The presented overview of SL and UL shows that they are methods of processing data rather than control agents. The output of either approach needs to be supplemented by additional logic to make a decision corresponding to the intended task. This leads to a state where the benefit of employing an ML model is superseded by a human-designed, rule-based control agent with the shortcomings and biases as described earlier. However, if these rules are based in conclusions derived from an attribute-rich dataset, the resulting heuristics-ML control agent could potentially outweigh the heuristics-only approach.

A dataset representative of the memory management task could be a series of memory accesses of a workload. A UL model could find patterns in the given series and those patterns can be associated to a memory device with the help of rule-based decision-making logic [160]. An SL model can be trained on the given series and predict the next memory access or its attributes (e.g. read or write, location, repetition, etc.) [58, 107]. When deployed during the execution of that workload, this model would predict the next memory access - e.g. a recently repeated read request to a memory location L. This would be a point of information for the arbiter to make a preemptive decision on the memory

device to be used. If the prediction was wrong and the memory device chosen was not the best-suited, it will likely negatively impact the desired optimization metric. Thus, the training accuracy and the decision-making logic influence the effectiveness of the control agent.

The proposals using SL in HMM based on Long Short-term Memory (LSTM), a type of Recurrent Neural Network (RNN) [148–150], have been already mentioned. Despite the improvements over other the existing heuristics methods, they highlight the training challenges of ML. To achieve higher levels of accuracy, larger datasets are necessary, and consequently, higher resource utilization [161, 162]. The underlying principle is to predict the access intensity of a memory page by training an LSTM model. Naturally, the demands for the training and inference phases will be different. [58] was reproduced for this thesis and found to consume 60-80 GB of working memory space during training and afterwards stored in ∼1 MB *per selected memory page*. Inference latency on the order of seconds is also incompatible with nanoseconds-fast main memory. This shows the impact of mismatching the given task and the chosen method to solve it.

As it was introduced in the motivation behind this work, the expected beneficiaries of HMM are workloads that use a lot of data and thus, a lot of memory pages. [58] trains a separate LSTM model for each page selected as high priority for that workload. This selection in itself is heuristic-driven, and was improved by a separate study [107] and also by the original authors [163]. Regardless of the selection method, every model requires training time with significant computational and storage resources. Given that many modern workloads use a number of memory pages on the order of $10^3 - 10^5$ or more [26, 58, 164–166], training itself becomes a challenge before the model could be deployed. This is also supported by the abundance of academic and commercial research on optimizing ML design stages [167–172].

In summary, SL and UL offer benefits for data processing but cannot necessarily escape heuristics in making a decision. Therefore, methods other than those presented are necessary. As online learning, flexibility, lower resource utilization, and non-biased approach are important to achieve the research goals set earlier, RL may be better-suited to designing the proposed HMM arbiter.

2.4.3 Reinforcement Learning for HMM

The overview on RL shows that it is a decision-making agent, unlike the other ML methods. The remaining challenge is to determine whether it is applicable to the HMM context. One aspect is the action space - in the context of this work and for HMM page placement, actions are defined as choosing a memory device in the next state of the system, i.e. device d among D devices. The other

aspect is the state definition. The complexity of the RL agent is often evaluated based on its state-action space.

The authors of Kleio argue against RL due to the complexity and the size of the design space based on their definition of the task. Their proposed approach, however, (1) creates an LSTM model per each page selected after heuristics-driven analysis, (2) is fully application- and configuration-dependent as models are bound to their training, and (3) requires significant resources for training with considerable latency for inference. This approach is potentially more restrictive in its design and does not necessarily minimize resource utilization.

To ensure the applicability of RL, its underlying principle of state transitions, or Markov Decision Process (MDP), should be upheld [119]. These are processes where state transition is probabilistic given the action applied by the agent and the current state of the environment [173]. The predictability is important to derive a correlation between the actions and their impact. Hence why both s_t and s_{t+1} needed to be known for the Q-value update function in Eqn. (2.1). Thus, the challenge is to determine a set of observable attributes and satisfy the MDP conditions.

- **Memory Access Sequence**

Using a memory access as a state is possible, but a series of memory accesses does not satisfy MDP conditions since it is a sequence of states with no (low) transition predictability. In a list of memory locations without an observable cause behind this sequence, the probability of transitioning from one location to the next is random to the agent. It observes environment-driven transitions with a weak correlation to its actions. The system appears stochastic, and the agent learns a decision per state but not an impact-driven long-term strategy.

Nonetheless, by this definition the agent must choose from a total of D^N possible placements for the requested data. Here, N is the number of memory pages to be placed and D is the number of devices. The decision-making space becomes too large even for a modest workload with e.g. 1000 pages, as there are now 2^{1000} possibilities to consider in a two-device system. While it is possible to replace the large state space using advanced RL techniques [174–176], it will still share the consequences of training on memory accesses as LSTM.

- **Memory Access Patterns**

Instead of individual locations, the access patterns could also be used as states. This also aligns with the logic of selecting attributes for heuristics-based decision-making. It reduces the number of potential observations, as several pages will fall under the same profile. However, patterns must be identified first - either as defined by humans or through the use of UL to automatically find them. If

irrelevant attributes are used to identify patterns, or a hypothetical workload causes unique accesses with no patterns, then the issue of resource utilization will be the same as before.

Moreover, patterns are generally observed over a period of time, and the insights could be compromised based on the chosen history window. However, the utility of access patterns is notable [160]. They can identify phases of a workload, for which an RL agent could learn to select one of the available policies or mechanisms, rather than learning its own policy [53, 66].

While memory accesses are undoubtedly central to memory management, the low state transition predictability is an issue that creates challenges for RL and other decision-making methods. From the perspective of the agent a memory access is not a representative observation of its impact on the environment. It chooses a location for an access but does not observe a change in its environment related to its action. Rewards are important to establish this state-action correlation [137, 138], but the state space cannot be unpredictable [119, 173].

Applicability of RL in (Hybrid) Main Memory

Q-prefetcher [66] has already shown that RL can be applied to HMM. It proposes a state definition that satisfies the MDP condition, as choosing a prefetcher will be reflected in the state attributes observed by the agent. Another study [108] is also notable to further show the applicability of RL in main memory.

The authors in [108] used RL to improve the efficiency of a DRAM device controller in scheduling incoming memory requests with significant performance gains - up to 30% for some simulated workloads. The agent reorders the queue of requests to be scheduled such that the internal buffer hitrate is maximized. The agent learns from observable attributes that are direct properties of the environment (e.g. next request to schedule). This eliminates heuristics in defining the states, and they are correlated with actions and rewards received. The result is that the agent implicitly discovers the hidden internal device organization - it helps to lower latency by improving locality of consecutive accesses in the device itself. These two works show that it is possible define a (hybrid) memory management task as an RL problem.

In summary, the HMM arbitration task must be defined such that there is a correlation of states to the actions and rewards. In HMM, choosing one of the devices has a quantifiable impact on the state of the system, such as free memory capacity, number of pending requests, etc. Chapter 3 will present the proposed approach to HMM arbitration as a system resource utilization problem.

2.4.4 Mixture of Decision-making Methods for HMM

Lastly, given the discussion so far, a combination of these decision-making methods could be considered to design a more "complete" control agent. It was shown that for SL and UL, their data processing capabilities need to be complemented by rule-based decision-making logic. RL with derived attributes (e.g. observing MPKI in [66]) is possible, but data-driven ML models could also be used for the benefit of RL.

While analyzing memory access sequences and large volumes of series-based data was deemed not suitable for traditional Q-learning as described earlier, a large state-action space can be replaced by a (deep) neural network (DNN) trained to predict the Q-values [174–176]. This can address the resource utilization concerns, but creates additional design challenges. The DNN is an SL model and needs offline training with runtime updates to ensure that it captures the feedback received by the agent. Additional mechanisms, such as experience buffers, can help to re-train the model and update its predictions. The resulting agent is potentially more sophisticated than classic Q-learning RL due to covering a larger state-action space. A relevant example can be found in hybrid storage systems [55], however, translating this approach to the main memory may not be immediately viable.

Memory hierarchy, as introduced earlier, makes data available to workloads on a timely basis. Different levels complement the limitations of each other in latency and capacity [5]. Hence, storage is not the first point of access for the workloads, unlike on-chip cache and main memory. The total response delay for storage is less critical than in the higher memory hierarchy levels, because the system architecture is designed to compensate for that. Data flow should not stall the system – typically achieved by prefetching or fetching several data blocks at once, and managing memory in regions [5, 7, 10, 11, 13, 69, 72–76].

Storage devices have access times on the order of microseconds and are not significantly influenced by ML models that could respond in micro- to milliseconds [55, 58, 177]. In contrast, main memory and cache operate in nanoseconds, with limited spare resources, and the introduced agent must be compatible with these characteristics of its environment. This is reflected in main memory and cache-level RL agents that strive for smaller state-action spaces and lower complexity [53, 54, 66, 108].

In summary, a control agent in time-critical environments should have negligible overhead in latency and resources utilization. While mixing ML methods may offer deeper insights for decision-making, in the main memory context it is important to preserve the simplicity and flexibility of the agent.

2.5 Summary

Mechanisms and policies for page migration in HMM could be improved, but a complementary and not mutually exclusive method is to distribute the pages more optimally in the first place. Considering the discussions on the details of the underlying principles of HMM design, its management policies, and various decision-making methods, the following key insights and conclusions about designing an HMM arbiter using RL could be made:

- An arbiter placed on the border of virtual and physical address spaces retains the abstraction of the HMM space to the rest of the system.
- Memory pages are used by memory management mechanisms such as the address translation, and align with the preferred placement of the arbiter.
- The initial placement of memory pages will influence the need for subsequent page migration and can reduce its frequency.
- RL is a goal-driven decision-making framework to learn its impact, unlike other rule-based methods.
- An RL arbiter that chooses a memory device in the HMM space should observe environment states that are influenced by its actions and have predictable transitions.
- System resources are influenced by the actions of an HMM arbiter more than observing a series of memory accesses generated by a workload.
- Attributes that describe the utilization state of the system are a direct property of the environment, rather than inferred from other parameters.

Lastly, the search for a non-biased and flexible decision-making agent benefits the general exploration and design of HMM systems too. Observations from all referenced works and their evaluation methodologies show that if workloads or memory devices are changed, then the policies designed and optimized for the original configuration might not be optimal anymore. The effort to design and optimize a new policy could be substantial, and the quality is not guaranteed to be the same across all evaluated configurations. Therefore, designing a flexible control agent with few to no assumptions in its decision-making could allow for a fair comparison and evaluation of various workloads and memory technologies.

PARL combines and builds upon these insights from the literature, improving the initial data placement in a flat HMM reflected in workload runtime reduction. The implementation details of PARL are presented next, in Chapter 3.

Chapter 3

Design of HMM Arbitration Methods

This chapter outlines the design of the environment in which the proposed HMM arbiter is integrated, as well as the design of the arbiter itself. It also includes the overview on the arbitration proposals from the literature that will be used for evaluation in this thesis.

It should be noted that in this work, the terms "arbitration" and "allocation" are sometimes used interchangeably due to their interconnected nature. More specifically, *arbitration* refers to selecting a memory device for *allocation* of a page from its physical space. Therefore, the proposed agent learns to arbitrate which device in HMM will be responsible for page allocation.

3.1 Hybrid Main Memory Environment

This section introduces the environment to simulate an HMM. First, the baseline simulator design is introduced, followed by modifications to support HMM and extend the models of memory to storage interactions. The proposed and evaluated management methods are integrated in this environment, with the exception of Kleio [58] that was originally implemented as a standalone solution [178] and reproduced from its source in this work.

3.1.1 Baseline Environment Implementation

The baseline Ramulator is designed for homogeneous main memory, as depicted in Fig. 3.1. It implements a simplified processor model to enable cycle-accurate simulation of various memory standards and technologies according to provided timing specifications, e.g. [179]. Additionally, the default implementation of the memory device controller can be extended with custom scheduling algorithms, or other mechanisms necessary for the target memory device.

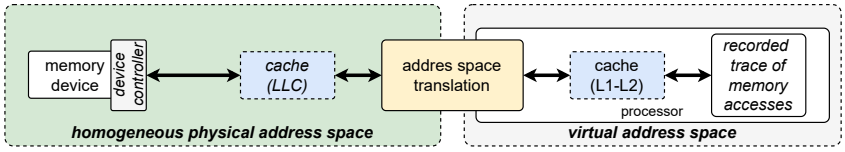


Figure 3.1: Conceptual diagram of a system as modeled by Ramulator.

Ramulator simulates workloads that are represented as traces (history) of memory accesses recorded from a real system. Each line of a trace file specifies the number of non-memory instructions, imitating the computational intensity of a workload and its impact on the memory accesses. The processor clock is incremented until these instructions are issued in 4-wide windows. Then, the memory requests can be ingested and issued to the memory. Each line has one read address and an optional write address. After the memory requests are issued, the processor receives the next line.

The simulated system specifies the main memory configuration and an optional cache hierarchy. The processor resides in the virtual address (VA) space, including higher cache levels (i.e. L1-L2). The main memory is located in the physical address (PA) space, together with the last-level cache (LLC). The Page Translation Mechanism (PTM) is a function of the memory but it is available to and invoked by the processor. This function allocates a physical page for an incoming virtual page, keeping the records in a Page Translation Table (PTT).

A memory request originates in the VAS. Then, the PTM is invoked if it has to be serviced by the LLC or by the main memory. The target PA is either known already and retrieved from the PTT, or it is a new request to be translated. In the second case, the PTM follows its algorithm to find a free page or to evict and replace one if none are available. The chosen PA is returned to the processor which can forward the request to the LLC or the main memory.

3.1.2 Hybrid Main Memory Support

Ramulator is first extended to support an HMM space, to later integrate and place PARL or other arbitration agents on the border of the address spaces. As shown in Fig 3.2, the model of Ramulator is preserved with minimal changes, and a similarly non-invasive implementation is intended for a full system.

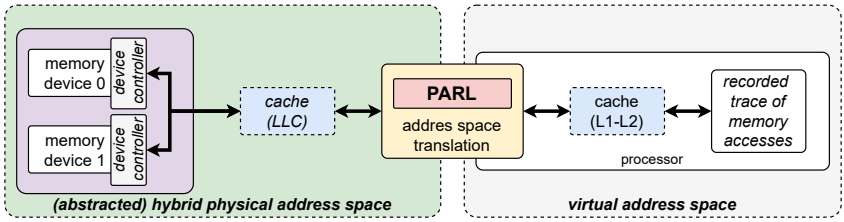


Figure 3.2: Conceptual diagram of HMM and PARL within Ramulator.

The "homogeneity" of the hybrid PAS is achieved by retaining a single PTT, as shown in Fig. 3.3. The memory devices reside in a contiguous address space and the second device is offset by the capacity of the first one. That is, if the first device has a capacity of three locations, i.e. PA#0-2, then, from the system perspective, PA#0 of the second device is PA#3.

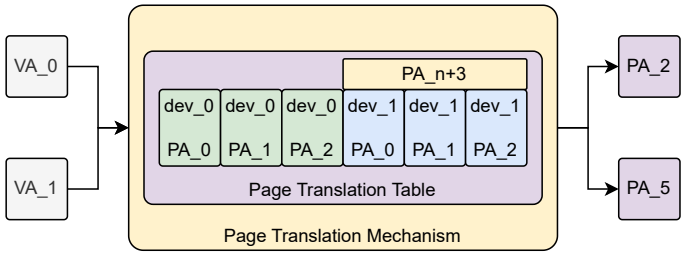


Figure 3.3: Conceptual overview of address translation in HMM.

Each device allocates and manages its pages without any changes, while the arbiter (e.g. PARL) invokes their respective PTMs. From the system or the workload perspective, the PTT is a homogeneous space. This design achieves the seamless and transparent HMM integration envisioned in the motivation behind this work. However, further modifications to the memory model of Ramulator are necessary, as described next.

3.1.3 Main Memory Interactions

Besides accurately modeling the main memory itself, the interactions across the memory hierarchy are also important. As it was introduced earlier, the main memory is positioned as a middle ground within the system. Unlike the simplified model of the processor with high-level approximations and an implementation of the cache hierarchy, the storage level is not considered in the baseline Ramulator. Standalone and full system simulators could provide accuracy and completeness within the storage level [180, 181], but integration with Ramulator might not be as feasible as expected. Additional mechanisms would be necessary to extend and support storage management from the processor perspective, as well as connecting and synchronizing events in simulators with potentially mismatching underlying principles (e.g. cycle-accurate, model-based, event-based, etc.).

In the current context, a high-level approximation of storage, similar to the processor model in Ramulator, could be more applicable. This is achieved by improving the page fault model and adding associated main memory delays.

Page Fault Mechanism

A page fault is one of the main mechanisms between the main memory and storage. It is a memory-critical event, undesired due to its significant impact on the system [182, 183]. Faults occur when the system has run out of free pages in the main memory or the accessed data is not present there. The system has to find a "victim" page (e.g. coldest or lowest priority) and transfer it to storage (i.e. page swapping), before the "offending" request can be serviced.

Ramulator also omits the initialization of main memory as part of the memory to storage interactions. Naturally, memory has to be populated with data for it to be accessed. This means that every new request to main memory should result in a page fault, which is often resolved by prefetching in real systems [5, 151]. The initialization is compulsory and unavoidable, hence in the baseline Ramulator it is presumed to be completed beforehand and this assumption is upheld in this work too. Thus, the PTT is modified to model page faults only due to exhausted main memory capacity.

In the baseline implementation, a physical page is randomly chosen and assigned to the offending request, and the page fault counter is incremented for tracking statistics. Then, the original translated pair remains in the PTT and the follow-up accesses to it do not trigger the counter. However, the original page (i.e. data stored there) has been swapped out to storage, and therefore, it is not available before it is fetched again. The associated delay of swapping is unaccounted too, and will be discussed in second half of this section.

In the extended Ramulator, the PTT is modified to remove the original VA-PA pair upon a page fault and trigger another fault when the original VA is accessed again. As a result, the number of page faults increases, especially for workloads with higher memory access rates. Fig. 3.4 shows the baseline and the modified number of page faults (log-scale) for different main memory capacities.

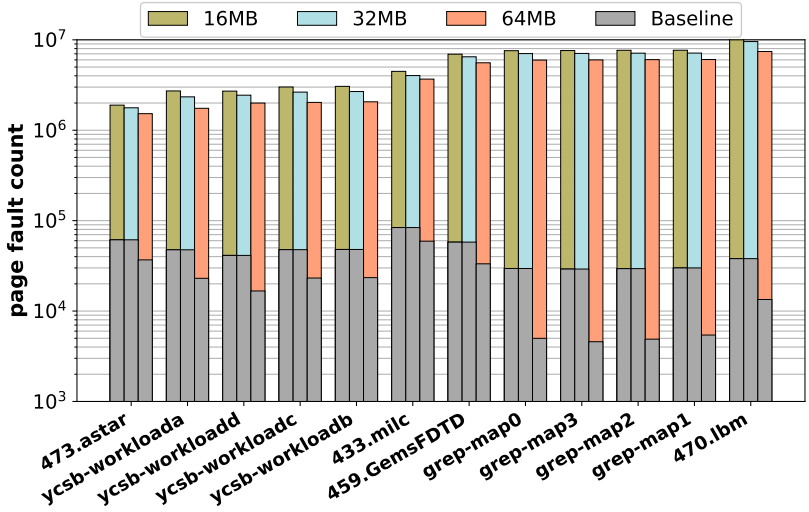


Figure 3.4: Page fault count (log-scale) for baseline and modified PTT under different DRAM capacities and various workloads.

These experiments were conducted using a homogeneous DRAM-only configuration and the capacity is restrained to a smaller range (16-64MB) due to the scale of trace files used for simulations. Throughout this discussion, memory pages are defined as a range of 4,096 addresses, referred to as 4KB pages. Only 4,096 pages can be stored in 16MB of memory or 16,384 pages in 64MB, respectively. All evaluated workloads and system configurations are introduced in Chapter 4.

Workloads with the most page faults are presented in Fig. 3.4, such as "grep-map0" and "470.lbm". The increase from the baseline page fault count is considerable, and to consider whether the modified mechanism is more realistic, it is necessary to analyze the memory accesses of these workloads. Table 3.1 provides an overview of the metrics used in this analysis and further explanations are in Appendix A. Cache related metrics include the miss status holding register (MSHR), which indicates poor cache locality if it is often unavailable, and reflects outstanding requests to be serviced by the memory. Subtracting the number of MSHR unavailable events from (all) LLC accesses should correspond to the total number of requests in the trace file.

Analysis Metric	**grep-map0**	**470.lbm**	**sort-map0**
total number of requests in a trace	9,402,018	9,988,010	275,783
unique pages used for all requests	37,760	46,189	4,891
LLC accesses	96,943,337	17,762,237	324,787
LLC misses	91,529,870	15,175,986	131,815
MSHR unavailable events	87,541,319	7,774,227	49,004
main memory accesses	7,755,353	11,664,712	105,348
page faults (baseline)	4,993 (64MB) 29,569 (16MB)	13,422 (64MB) 37,998 (16MB)	0 (64MB) 0 (16MB)
page faults (modified)	5,969,131 (64MB) 7,580,364 (16MB)	7,424,085 (64MB) 10,629,249 (16MB)	0 (64MB) 25,706 (16MB)

Table 3.1: Reported metrics to analyze memory accesses of workloads

The "grep-map0" and "470.lbm" are representative of large workloads with higher volumes of accesses in the trace file. "grep-map0" has a low cache locality reflected in high MSHR unavailability and should result in many main memory accesses. "470.lbm" has better but still poor cache locality, confirmed by observation that all unique addresses in the trace file are accessed repeatedly but only a few times ($\sim$ 3.3x) on average. Combined with the high volume of accesses, it means that these types of workloads switch between memory locations often. Thus, they should trigger frequent page swaps in the order of the number of requests, unlike the number of page faults reported in the baseline. Lastly, "sort-map0" is a smaller workload but it unexpectedly does not generate any page faults in the baseline implementation for any memory capacity - even though it accesses more pages than can be stored in 16MB.

Naturally, for workloads larger than memory capacity and with high access rates, page faults should be a regular occurrence. The reported number in the baseline is not representative of the demand that such workloads could place on the main memory. After the modifications, page faults more closely follow the total access rate of those different pages, as expected.

In summary, the modified PTT reflects a more realistic scenario, and highlights how limited main memory capacity is detrimental to the system, further supporting the need for extended main memory capacity in general, and by using HMM due to technological issues mentioned in the opening chapters. The remaining aspect of modeling the page faults is their impact on the total access latency of the offending request.

Modeling Page Fault Delays

Storage devices can be three to six orders of magnitude slower to access than main memory [32, 33, 38], not accounting for the system-level processes of encountering and managing a fault. Accessing storage to service a page fault would significantly degrade the runtime of a workload, especially if the workload has to be stalled until the data becomes available.

To mimic the delay of accessing storage, the latency to service an offending request must be increased. To achieve this, first, each offending request is marked accordingly. Then, when the memory controller is scheduling and issuing device commands, the scheduled completion time of an offending request will be incremented by a specified number of clock cycles. This delay would approximately reflect the time it takes to evict a victim page to the storage device before the offending request can actually be serviced.

It is implemented as a configurable parameter to observe the impact of various storage access delays on the workload runtime, as reported in Fig. 3.5. Since the processor and the memory devices operate on different clock frequencies, but the computational part of the system is untouched, evaluation of this delay and workload runtime is reported using memory clock cycles (cc.).

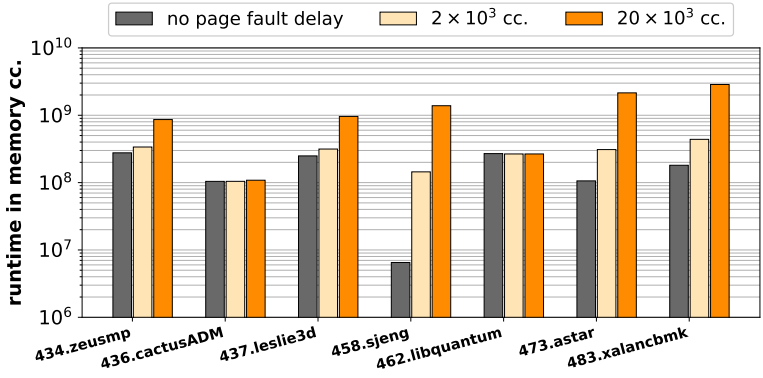


Figure 3.5: Impact of page fault delays on workload runtime.

While the goal was to sweep across potential delay values, the simulation time quickly became unfeasible. If the average DRAM access latency is 50 cc. representing a nominal DRAM latency of 50ns at 1200MHz clock frequency, then a storage device at only two orders of magnitude slower than DRAM (e.g. 2,000 cc.) is a highly optimistic case. Such a device would be closer to the SCM-M devices discussed previously than to traditional NAND/NOR

Flash storage technology. This delay results in a consistent but manageable degradation of the runtime - longest simulations could be completed in a few hours. It takes significantly more time (up to a day or more) when the page fault delay is increased by only one order of magnitude (e.g. to 20,000 cc.) and simulation times are unfeasible at any higher values.

As mentioned earlier, an RL agent needs several episodes to learn - a complete simulation of a workload is an episode for PARL. Therefore, setting the delay to 2,000 cycles keeps the experiments feasible while (optimistically) approximating the time cost of a page fault.

In summary, the environment has been extended to support and improve the modeling of (hybrid) main memory, and to highlight the impact of memory-critical events on the workloads. HMM has been motivated as a method to resolve the issues of the main memory limitations, and to mitigate the consequences of these events. The system is set up with HMM of sufficiently large capacity to avoid page faults, and configured to have a considerable delay in the case that a page fault is triggered. It is the responsibility of the deployed management method to ensure that the capabilities of the main memory space are maximized and the design of these methods is the further topic of this chapter.

3.2 Proposed Method for Page Arbitration: PARL

This section introduces PARL (also "the arbiter" or "the agent") using the constituent components of an RL agent, including considerations on its implementation complexity bound to its environment constraints.

3.2.1 PARL: Design Overview

PARL is integrated in Ramulator as part of extending the simulator to support HMM. The internal structure of the agent is shown in Fig. 3.6.

When the system receives a new virtual address from the trace file and it will be serviced in the PAS and thus translated, the agent will be invoked. PARL first observes the environment to determine its current state, as defined next. Then, if the request has been encountered before, the simulator invokes the already known target PTM. Otherwise, PARL makes a decision given its current exploration rate - either randomly in order to learn, or by exploiting its Q-values. Then, the chosen target PTM is invoked and the simulator follows its baseline implementation. In all cases, the agent will receive feedback for its decisions to evaluate the long-term impact of the initial placement.

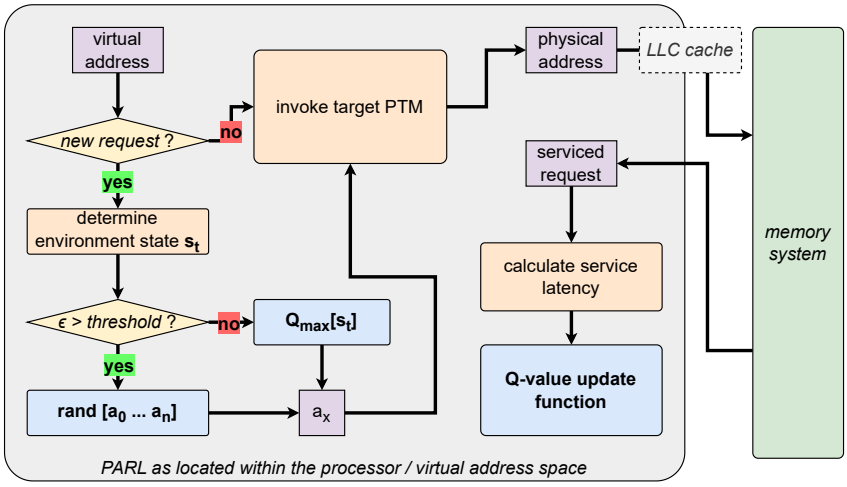


Figure 3.6: Internal structure of PARL and interaction with its environment.

Once the memory controller has serviced a memory request, the effective latency is calculated as the difference between the completion and the issued device command timestamps. This information is part of the internal simulator statistics tracking (akin to performance counters in real systems) but it needs to be made available to the agent. The latency is placed in a rewards buffer that enables sharing information between the memory and the processor as per the design constraints of the simulator. The Q-value is then calculated by matching the received reward to the action and in which state it was taken, as well as the state that followed it. Definitions of the actions, rewards, and environment states are provided next.

3.2.2 Action Space Definition

The action space **A** of PARL includes as many actions as there are main memory devices. For a two-device system, PARL chooses one of them to invoke the corresponding PTM – therefore, it has two actions. The agent chooses between two numbers, i.e. 0 or 1, which would indicate if the PTM of DRAM (i.e. "0") or SCM-M (i.e. "1") should be invoked. Then, the follow-up processes such as executing the PTM function and device-level operations (i.e. scheduling and issuing commands) remain as the responsibilities of the rest of the system.

For previously encountered requests that are already in the PTT, the processor

can forward the request to the target device. For the new requests, PARL *selects* a device to allocate a page instead of *allocating a page*. Hence, PARL is an arbiter, rather than a page allocator. This eliminates the decision-making space with D^N-many choices corresponding to the total number of potential placements in the available memory space. At any given point in time, PARL has to choose from a decision-making space as large as the number of memory devices in the system, addressing the objection presented in [58] against RL.

Additionally, the environment does not allow PARL to choose a device with no free space (pages) remaining. In RL, the environment can limit the actions available based on the current state of it - which would be a reflection on the previous decisions of the agent. If it has made decisions that exhausted the capacity of one of the devices, this will lead to a page fault. As it was mentioned in the discussion on page faults, HMM offers greater main memory capacity to eliminate page faults as much as possible. Therefore, the environment places constraints on the actions available, and the agent will also receive strict penalties for causing a page fault as described next.

Lastly, a brief discussion on using PARL for page migration is presented. Since this agent is designed as an arbiter of the next action, it is better suited to select whether a migration event should be triggered - rather than deriving a migration policy itself. If a page migration mechanism is present in the system, then invoking it can be another action, similarly to how PTMs are invoked by the agent and prefetchers in [66]. There could also be multiple migration mechanisms or policies that benefit different phases of a workload. PARL will learn when migration should be invoked, but not how or which pages to migrate.

However, the implementation of migration mechanisms is dependent on system processes for accurately injecting additional memory requests, invalidating any pending references to the evicted or migrated page, ensuring the data dependency in the workload is not violated, as well as external simulators to provide accurate models of storage-level processes, etc. Given the limited extent to which main memory and storage interactions can be modeled in this simulator, the agent is designed only for the initial device arbitration. Considering the constrained access to devices and the lack of migration, the agent should be expected to maximize the potential benefits from the initial placements.

3.2.3 Rewards System

The reward function describes the impact of the actions on the evaluation metric to be optimized. It is one of the most important aspects of RL agents that directly impact the success of learning [119, 138].

HMM will, by definition, have an imbalance between its "better performing" device (e.g. DRAM) and the "complementary" one (e.g. SCM-M). As a result, some performance degradation is unavoidable. The goal is to define it in a measurable metric and minimize the loss. Given that all other components of a system are unchanged and only the main memory is modified, the time spent in the memory would directly reflect the impact of the agent. Therefore, the objective for PARL is to minimize the runtime of a workload that it can achieve by improving the initial page placement in the absence of migration.

Rewards that are relevant to PARL can be defined by the time that the chosen device took to service a memory request, as defined in Eqn. (3.1). To avoid the previously mentioned bias towards the technology-defined nominal (expected) access latency of a device, the memory controller-reported effective (total) latency of servicing a request is used. This latency can be tracked by the processor to include the time for processes beyond the memory domain – however, the environment outside of memory is modeled less accurately, is unchanged by integrating HMM transparently, and lastly, is less relevant to PARL as it interacts only with the main memory.

$$reward = 1/latency \tag{3.1}$$

This reward is agnostic of a given memory technology, includes the impact of device-level processes, and will follow the changes in the memory system. The inverse of the access latency provides a greater reward for less time spent, indicating that the action was desired. In case of a page fault, this latency will be greater due to introduced page fault delay modification. However, this definition of the reward is not the most effective.

Page faults are prohibited in the action space and the total capacity is set to be large enough to avoid them, such that the inaccuracies of modeling storage-level events are mitigated (explained further in Chapter 4). Yet, if a page fault is unavoidable - i.e. when capacity of all memory devices is exhausted, the reward in Eqn. (3.1) is not as effective as if it was shaped to a negative value [184, 185]. A negative reward, as in Eqn. (3.2), rather than a smaller reward, provides a clear signal to the agent that this action was undesirable.

$$reward = \begin{cases} \frac{1}{latency} - 1 & \text{if a page fault has occurred} \\ \frac{1}{latency} & \text{otherwise} \end{cases} \tag{3.2}$$

The effect of this can be observed in Fig. 3.7. It shows how feedback provided to PARL impacts its learning, reflected in the number of page faults occurring in a sample workload ("458.sjeng"). In the baseline simulator, this workload uses

26,294 pages but had no page faults in DRAM-only memory of 64MB (16,384 pages), and 21,649 faults after the modifications. This figure was generated using HMM with 96MB of total capacity that fits 24,576 pages (32MB of DRAM and 64MB of SCM-M) and should reduce the fault count further, as during this experiment the actions of the agent were not yet restricted by the environment.

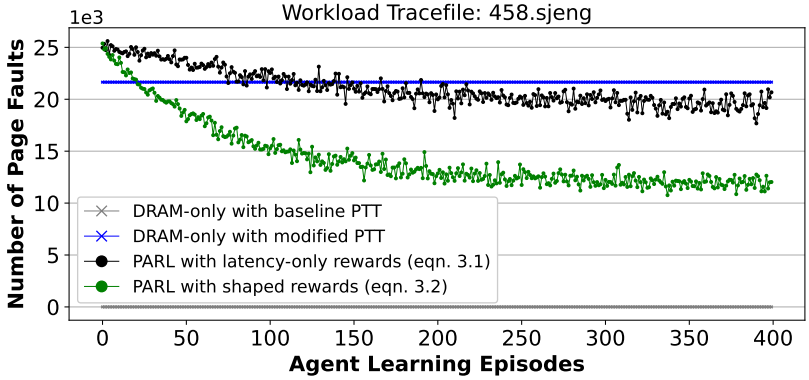


Figure 3.7: Impact of scaled vs. shaped rewards on the number of page faults caused by the agent without environment constraints, for a sample workload. DRAM-only baseline and modified page fault count plotted for reference.

At first, a higher fault count is observed due to starting in the exploration phase, when the agent does not have any knowledge of its environment. At an equal exploration rate decay, the negative rewards force the agent to learn undesirable events more effectively (i.e. faster) than a small, but positive value. There was no noticeable difference when using only the effective latency in Eqn. (3.1) or if the reward was downscaled by a large factor (e.g. 10^6) to make it even smaller, since zero or near zero-value rewards do not provide usable feedback.

3.2.4 Environment States and Observations

To apply actions and receive rewards, the agent must distinguish between the states of its environment. These are defined by the attributes or the properties of the environment that can be quantified into discrete states, and then provided to or followed (observed) by the agent. While rewards are speculated to be "enough" in teaching an agent the desired behavior, the states are equally important in correlating them to the actions of the agent.[4]

[4]From the seminal paper "Reward is Enough" (2021) by Silver et al. and including Richard Sutton - one of the founders of the computational RL [138]. Several follow-up works have revisited this statement, but that discussion is outside of the scope of this work.

In the current context, the observable attributes are listed in Table 3.2. These are derived from following and summarizing the previous discussions around the literature on HMM design and decision-making applicable to it.

System Resource Attributes	**Memory Request Attributes**
available space in a memory device	*page/address distance (spatial locality)*
number of non-memory CPU instructions	*page/cacheline hit (spatial & temporal locality)*
memory scheduling queue depth	*repetitions (spatial & temporal locality)*
occurrence of memory-critical events (e.g. page faults)	*memory access recency (time-intensity)*
current memory operation (read/write)	*current memory operation (read/write)*

Table 3.2: Overview of observable attributes in the environment.

Here, attributes on resource availability are contrasted to attributes of memory accesses, based on the conclusions on the reviewed proposals and considerations for MDP properties of RL states. Attributes such as access locality expressed in repetitions (temporal) or distance between two locations (spatial) are used by the existing proposals, but will have state transition probability issues.

On the other hand, available free space in the memory devices and the depth of the scheduling queue will change if they were used to service a memory request. If an agent made a decision to use one device in one state, then in the next state that device will have one fewer page available, and more requests pending to be serviced. There is a correlation from the action applied to the change in the environment. These changes can be quantified and represent discrete states of the utilization of memory devices. Additionally, such attributes can be tracked as absolute and immediate values, rather than locality attributes that need to be derived from analyzing access history or profiling workloads.

Attributes that are intended to describe the current availability of the memory devices and reflect the state of the system will be referred to as *system-level* states or observations. Combined with the proposed actions for device selection, the complexity of the state-action space should be less than the *request-level* attributes are used with the action space intended for page allocation. The next two sections discuss the details of these state definitions.

3.2.5 Request-level State Definition

While request-level state definition is not sufficiently applicable to an RL agent, it was used in the early design experiments. The goal is to support the previous discussions on using memory accesses in the decision-making logic for memory management. The full composition is listed in Table 3.3.

Attributes	Values	Size	Notes
request type	[rd, wr]	1 bit	*device impact*
zero non-memory instr.	[yes, no]	1 bit	*time-intensity*
address repetition	[yes, no]	1 bit	*spatial & temporal locality*
virtual page hit	[yes, no]	1 bit	*spatial & temporal locality*
cache line hit	[yes, no]	1 bit	*spatial & temporal locality*
distance between page numbers	absolute	52 bits	*spatial locality*

Table 3.3: Request-level state attributes for PARL.

This definition attempts to reflect the common parameters used in the literature, as well as the expected application of an RL agent for page allocation. Thus, the distance between two consecutively accessed pages (in the VAS) is included in this request-level state definition. It should generate fewer combinations than individual page numbers. If the internal memory device organization is known beforehand, it might be possible to calculate distances based on it and find a match between VAS and PAS locality. But any clustering besides individual page numbers implies more complex patterns based on heuristics which have been argued against in the earlier discussions.

The agent also observes the access type (read or write), whether it is a repeated access, or whether it was recently accessed. These attributes are, by definition, properties of a memory request. The latter two reflect temporal locality, while page distance reflects the spatial locality. The number of non-memory instructions that the processor will execute before issuing a memory request is not a direct property of it, but closely associated. It is included to reflect the temporal locality of memory accesses, even though its definition is closer to the system-level rather than the request itself.

This state definition could generate as many as ∼300,000 states encountered by the agent in a single episode of simulating a workload in Ramulator. The main contributor is the page distance based on the maximum range of the entire VAS. It takes 52 bits to cover the range of individual virtual page numbers in systems with a 64-bit address space, as 12 of those are necessary to represent 4,096 locations within a 4KB page.

A recorded history of previous requests is also necessary to track these state attributes. A sliding window of the past 16 requests was used in these

experiments (compared to 64, 128, and more). Larger windows would increase the horizon of analysis but would also incur greater penalties in time and resources required. Nonetheless, the applicability of this state definition is not sufficiently justifiable to consider its implementation and evaluation further.

In summary, the request-level definition of an environment state depends on the expected patterns of memory accesses. It also does not fully satisfy the state transition predictability requirement. Furthermore, evaluated qualitatively, it does not describe a state in which an agent is expected to choose a currently available device and receive feedback to optimize resource utilization. Lastly, given that the RL design space is proportional to the state definition, request-level attributes are not viable in the current context.

3.2.6 System-level State Definition

Attributes that were described as system-level are listed in Table 3.4. This definition mitigates the concerns around large state spaces in RL and also reflects how the environment would change after an action applied by the agent.

Attributes	**Values**	**Size**	**Notes**
request type	[rd, wr]	1 bit	*device impact*
# non-memory instr.	4 bins	2 bits	*system utilization*
free memory capacity	6 bins	3 bits per device	*memory utilization*
scheduling queue depth	3 bins	2 bits per device	*memory utilization*
page fault occurrence	[yes, no]	1 bit	*system/memory state*

Table 3.4: System-level state attributes for PARL.

These attributes are accessible within the system, describe a direct property of the environment, and do not require additional mechanisms to determine their values. Accordingly, derived attributes are not considered to maintain these characteristics. As an example, MPKI, as used for an RL agent in [66], is a logical convention and can be influenced by the main memory, but it is driven by the cache hierarchy and how the executed workload exploits cache locality.

A brief explanation of each attribute is given next:

- *request type:* read or write access to memory
- *non-memory instructions:* the number of non-memory instructions between two consecutive memory accesses
- *memory capacity:* number of currently free pages in each memory device

- *scheduling queue depth:* number of currently pending requests to be scheduled in each memory device
- *page fault:* whether the system has exhausted the main memory capacity

The number of non-memory instructions is available per the definition of workload trace files in Ramulator, but in real systems could be reflected by instruction counters or time tracking. Fewer of them indicates higher intensity of memory accesses and can implicitly differentiate the phases of a workload. If the system has been executing many non-memory instructions, it would imply that the memory system has had time to process and empty its scheduling queues. As mentioned, this attribute is rather system-level than a request-level one, and its granularity is improved from a binary definition in request-level observations to binned values to provide more information to the agent.

The queue depth reflects the load on scheduling in the device controller. The controller must schedule and issue commands - both for device operation (opening or closing a memory location for access) and accesses themselves (read/write) [179]. But if the queue is organized such that it frequently requires operational commands to issue an access command, then the memory device would not be utilized efficiently.

This effect can be observed in [108] that explicitly reorders the queue itself to issue more access commands consecutively, until operational commands cannot be avoided. By observing the load on the device, the expectation for PARL is that it will learn a distribution of pages that also implicitly improves device-level access locality by receiving greater rewards as it minimizes the effective access latency. Combining these two approaches could lead to further improvements in memory device operation.

The page fault flag indicates that the system has had a critical event and memory devices have exhausted their capacity. However, it could be redundant since this condition is also covered by the remaining free space attribute. PARL can use these attributes for preemptive decision-making, as it did not learn to use a single (e.g. faster) device until its capacity was exhausted, due to the considerations of effective access latency. This is in contrast to other policies, such as DRAM-first that invariably sends all requests to the same device due to its nominal access latency - but will inevitably overload the scheduler.

Lastly, each state s_t in the state space $\mathbf{S}$ is constructed by concatenating the observed attributes into a 14-bit vector. The state space is limited to the number of valid combinations of the attributes: 5,184= $[2 * 4 * 6 * 6 * 3 * 3 * 2]$, and not the full range of 2^{14} combinations of the vector. The granularity of these attributes ranges from a binary choice to binned values to minimize the implementation overhead and impact on the simulator, while maintaining their

functionality. The full range of optimizations and tuning the definitions is planned for future studies on the physical implementation of PARL outside of simulated environments.

In summary, system-level states satisfy the requirements of RL and the goals set for an HMM arbiter better than request-level definition. The correlation between the states in which PARL applies its actions and receives feedback for them allows for calculation of the Q-values, as defined next.

3.2.7 Q-value Update Function

As covered in Section 2.1.2, after receiving a reward the agent calculates the perceived impact of its action. In classic Q-learning, the Eqn. (2.1) is used to calculate the Q-value for the action a_t chosen in the state s_t. A slight variation was introduced in the implementation of PARL, shown in Eqn. (3.3):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma[Q_{max}(s_{t+1}) - Q(s_t, a_t)]] \tag{3.3}$$

Instead of discounting the target Q-value and subtracting the full current Q-value, this modified version discounts the delta between the two. Subtracting the Q-value calibrates the influence of the target that the agent strives towards. The greater this "error" signal, the lesser will be the impact of the target as in the Eqn. (2.1). In contrast, the Eqn. (3.3) allows more optimistic Q-values because the entire delta is discounted instead of only the target.

Fig. 3.8 visualizes the impact of these two variations of the Q-update function using a simple RL agent in a sandbox environment with 10 states (s=0 to s=9). It has only one action and receives a reward of "1" or "0" in every other state. Discount factor and the learning rate are the same as for PARL.

This figure highlights that the Q-values calculated in each state for either of the functions are very similar. In later episodes, as the agent accumulates rewards and builds up the Q-values, the discounted delta version exhibits slightly greater Q-values. The potential shortcoming of the discounted delta is that the agent might not converge to a stable Q-value. For this to happen, the rewards must be random or not stable (less predictable), visualized in Fig. 3.9.

As it can be seen, random rewards influence both versions of the Q-update function, e.g. in states s=8 and s=9. However, the instability caused by the discounted delta or the randomness of the rewards can be mitigated by using a larger discount factor γ and low learning rate α - as is the case for PARL and the sandbox agent. These factors ensure that the newly calculate Q-value does not strongly impact the existing one. A higher α or lower γ would lead to faster application of the new Q-values and destabilize the learning process.

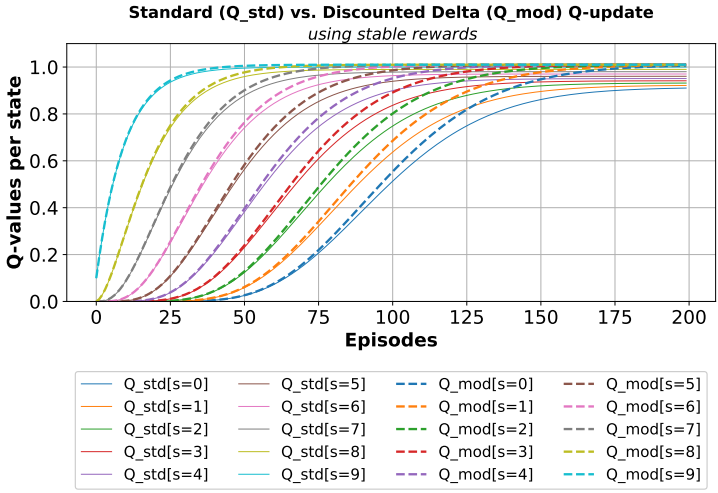


Figure 3.8: Q-values calculated using the standard Q-update Eqn. (2.1) vs. the discounted delta Q-update Eqn. (3.3) for a simple RL agent in an environment with 10 states in 200 episodes and stable rewards.

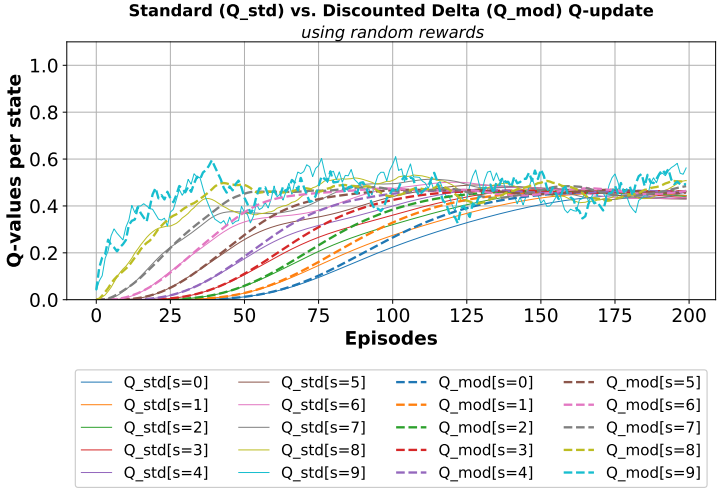


Figure 3.9: Q-values calculated using the standard Q-update Eqn. (2.1) vs. the discounted delta Q-update Eqn. (3.3) for a simple RL agent in an environment with 10 states in 200 episodes and random rewards.

Lastly, a data structure, often referred to as the Q-table, stores the Q-values for each state-action pair. The agent uses it to select the action with the highest Q-value for the given state when it exploits its "experience".

3.2.8 Practical Implementation Details

This section, visualized in Fig. 3.10, gives a brief overview of the simulator codebase, and how HMM and PARL integrate into it (highlighted in yellow).

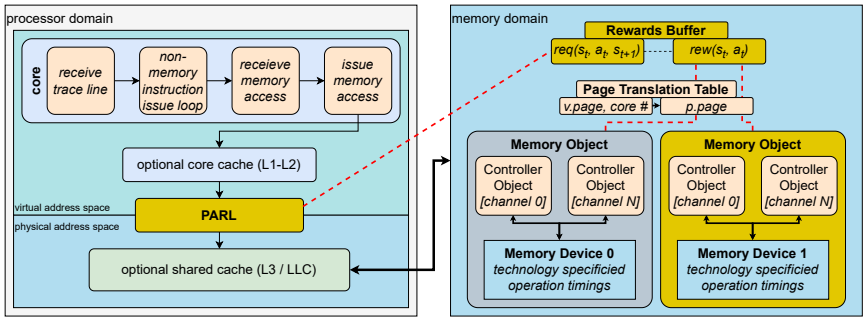


Figure 3.10: Ramulator codebase visualization with PARL and HMM support.

Overall, there are two domains in the simulator - the processor and the memory. The processor includes all computational elements (i.e. cores) and levels of the cache hierarchy enabled for the simulated system, with its own clock frequency.

The memory domain has a separate clock frequency and includes the memory objects, multiple of which are instantiated for HMM. Memory objects, but not the underlying structures, can interact with the processor. Each memory object includes the memory device represented as standardized timings of device-level commands [179]. Conceptually, there is a single controller for each memory device, but since devices can have parallel access channels, controller objects are instantiated per channel and invoked in the same clock cycle.

PARL utilizes the existing structures to retrieve memory-related information (i.e. PTM and PTT), only adding a rewards buffer and its Q-table. The observed attributes are provided to PARL by the environment using the existing parameters in the codebase - e.g. number of free pages, number of pending requests, current request type, etc. The code for PARL is a separate function invoked by the processor for every memory instruction. It returns the chosen action as a number (i.e. "0" or "1") that refers to the memory device to be used.

A rewards buffer for PARL is instantiated within the memory domain to be accessible to in the processor domain. This buffer is necessary due to the abstraction of the device controllers from the processor. A memory controller attempts to optimize the scheduling queue [108, 179, 186–188] and does not service the memory requests in the order they were received. Therefore, the reward generated for PARL must be matched to its state-action pair. To achieve this, a temporary triplet of $[s_t, a_t, s_{t+1}]$ is stored until the corresponding r_t becomes available. As soon as the Q-update function is completed, the stored triplet is removed to keep its storage overhead low.

3.2.9 Estimated Implementation Overhead

PARL is as a global agent – the resources necessary for it are bound by the definition of its state-action space and are not affected by the rest of the system (e.g. single or multiple cores, cache hierarchy, etc.). Only expanding the HMM with more devices can make the state-action space larger by adding more actions and additional attributes to the state definition. The expected overhead of the decision-making traffic is minimal [189, 190], but in-depth analysis is part of a future study on full system implementation.

Since PARL exists in a simulated environment, its complexity can be evaluated in the context of data structures using the big-O notation [191, 192]. The Q-table is implemented with access time complexity of $O(logN)$ to insert or look up entries, where N is the space size. The are 5,184 states or 10,364 Q-values possible for a two-device system. On average, only ∼100 states were encountered across evaluated workloads, in contrast to the request-level observations. States not encountered are not instantiated to reduce the storage overhead.

In all cases, though, the maximum storage for the Q-table depends on what number representation is chosen for the Q-values. For example, if decimal point numbers are represented using 32 bits, as is the case in modern systems, then all possible Q-values would occupy 81KB and another 9KB would be necessary to represent all 5,184 14-bit states. The Q-table is written into a file for statistics and its size was in the order of a couple hundreds of kilobytes, without any optimizations to the storage format. For a real system, number representations can be pruned to as few bits or bytes as possible, which would reduce the overhead even further. It should be noted that this is a process with tradeoffs, as number representation and quantization of stored values in ML models is an active research area to minimize the accuracy loss [193–196].

Given the scope and exploratory nature of this work, the design of PARL is feasible for integration in real systems, as also proven by hardware-optimized

implementations of RL agents with high-level performance under limited resources [189, 190].

In summary, PARL builds upon the insights from the literature on hybrid and homogeneous main memory management using RL. It proposes a novel way to optimize memory utilization using system-level observations and improve the initial page placement in HMM. The design complexity of PARL is as low as possible to incur negligible implementation overhead.

3.3 Design Overview of Relevant Works

Having discussed the design of PARL, this section presents the methodology behind other HMM proposals that will be used in this work for further evaluation. The focus is on the initial page placement of existing works. Three common policies extracted from the reviewed literature, and two more works based on heuristics and ML are included too. All works have been briefly introduced in Section 2.3 but implementation details are presented here.

3.3.1 Naive Initial Placement Policies

Based on the literature reviewed in Chapter 2, three common policies to place pages can be distinguished: (1) *DRAM-first*, (2) *Random*, and (3) *Round-Robin* and all reviewed works proposed migration policies around them. These policies are simple and intuitive in their logic, and thus, are referred to as naive. They have also been implemented in Ramulator for evaluation in this work.

DRAM-first policy allocates memory pages to the DRAM device in the order of their occurrence (chronologically). Once DRAM capacity is exhausted, new requests must use the SCM-M device.

This policy has two major caveats: first, if the workload has a higher access intensity for memory pages in the later part of its runtime but the DRAM device is full, then the time-critical pages must be placed in the SCM-M device or a victim page must be migrated. Both cases would incur considerable performance and energy consumption penalties. An example of such a workload can be seen in Fig. 3.11. This figure visualizes the page accesses throughout the lifetime of the workload plotted with the order of appearance for each unique page.

As it can be seen, pages that appear later, on the right-side of the x-axis, have much more varied (and higher) access rates that the earlier ones. If the better performing device has no free space to place these pages, it could lead

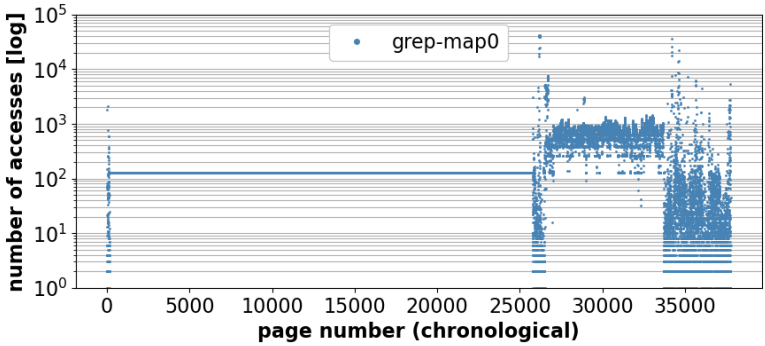


Figure 3.11: Total access rates (log-scale) of pages sorted chronologically.

to using the slower device or unnecessary migration events. Second caveat for the DRAM-first policy is sending all requests to the same device. With no regard to their priority or the load of the scheduler, this will cause the effective latency to increase. As such, DRAM-first may be a simple starting point for migration-driven proposals, but it does not necessarily reduce the burden of migration or maximize its benefits.

Random and round-robin policies choose a memory device unpredictably or by alternating between them. The latter can have variations that switch every N number of requests, but only the $N=1$ version is considered. These policies could achieve balanced memory utilization as they effectively parallelize accesses to the devices. However, they do so without considering their impact, and potentially miss the nuances about the current state of the system.

There are also proposals that use an SCM-M-first policy. But SCM-M devices considered there are older NVM technologies, such as PCM, that have historically been slower than DRAM [33, 68]. Predictably, this policy would have the most severe performance degradation. If a better performing technology is used for an SCM-M device, then this policy would be the inverse of DRAM-first. Therefore, this policy is excluded from the current discussion.

3.3.2 APAMS

The authors of Address Pattern-Aware Memory System (APAMS) originally explore HMM design in general, and as part of it propose a locality-driven approach to the initial page placement, also implemented in Ramulator [63]. APAMS allocates memory accesses with low locality (random accesses) to the DRAM device, and sequential ones to the SCM-M device. This rule is derived

from the observations that DRAM can have better performance when accessing random locations in the device, compared to other technologies considered in their study. However, this is not necessarily true as DRAM devices have physical data buffers that fetch the entire region that hosts the particular accessed location. Memory devices are commonly organized as two-dimensional arrays of rows and columns, and this physical structure is known as the row buffer. Several rows could be fetched into these buffers, and as a result, consecutive (sequential) accesses in DRAM would also have low latency.

This policy can be successful for workloads that exhibit patterns aligned with the assumptions and as long the memory devices satisfy those expected access patterns. In APAMS, the spatial locality is based on the distance of consecutively requested VAS memory locations and the fixed block of data to be fetched from the main memory (i.e. a cacheline of 64 bytes). Requests to memory addresses with a difference greater than 64 bytes are considered random.

APAMS highlights how heuristics-based proposals can be simple in design with intuitive decision-making, but not always adapt to changing conditions of the workloads or the system. However, the authors conclude that HMM systems would benefit from more flexible management policies, including the use of ML.

3.3.3 Kleio

Kleio has already been mentioned several times as part of Section 2.3 and in the discussion around RL. Its design structure is shown in Fig. 3.12 and details on its decision-making process and operating logic are presented next.

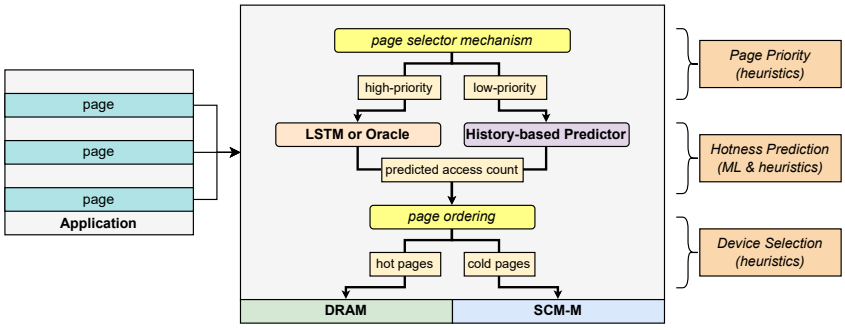


Figure 3.12: Design Overview of Kleio LSTM-based Page Scheduler [58].

Kleio approaches HMM page management as a periodic scheduling job. It migrates pages if the predicted access rate in the next interval mismatches the

current placement - i.e. hot pages in DRAM and cold ones in SCM-M. The design behind an LSTM model per every "important" or high-priority page and its potential drawbacks have been discussed already.

The remaining low-priority pages are managed by a history-based predictor which is based on the real access intensity from the previous interval. If a page was hot, it is predicted to become hot in the next scheduling interval, then it will be migrated to DRAM and vice versa. The workload runtime is incremented by a specified time cost of migration and every access has a fixed average latency due to model-based simulation. Scheduling intervals are defined by the desired number of requests per interval (e.g. 100 by default) and the workload length.

In the simulated system, the memory capacity is specified as a ratio of the total number of pages of the workload. If this ratio is 1, then all pages are allocated to DRAM, but the time costs of scheduling intervals and migration still apply. For all other ratios, Kleio uses the round-robin policy for the initial placement - e.g. for the ratio of 0.2, only 20% of pages can be placed in DRAM. Once all pages are placed, Kleio continues with its prediction and migration process.

Every workload is profiled offline to identify the pages that are often misplaced using the history-based predictor and have high access frequency. Then, LSTM models are trained for each of those pages to be deployed during the simulated execution of the workload. These models will predict the hotness of the page for every scheduling interval with accuracy higher than the history-based predictor. Kleio, as a result, is constrained to the configuration on which it was trained and every model is exclusive to a given page of a workload.

The publicly available source code for Kleio replaces these LSTM models with an "oracle" [178]. This oracle is a list of important pages and their access frequency for every scheduling interval, created beforehand during the offline workload profiling. The oracle then returns the real number of accesses instead of the LSTM model prediction. This removes the overhead of deploying LSTM models, but also eliminates the inaccuracy of pages misplaced by the models.

Kleio is integrated into a model-based simulator (i.e. using a fixed average access latency) which limits the extent of comparison with PARL due to the cycle-accurate nature of Ramulator. The hitrate in DRAM is the main quantitative and time-independent metric available for both Kleio and PARL during evaluation. If PARL, without migration, can achieve similar or higher hitrate than Kleio, it would indicate the potential of initial page placement.

3.4 Summary

This chapter has discussed the design and implementation of various policies for initial page placement in HMM, including the proposed method. It was also shown that HMM can be transparently integrated and mask the heterogeneity of main memory within the address space translation. PARL is differentiated from the existing works as an RL agent with a smaller state-action space, feasible for implementation by considering device utilization to approach the HMM management problem.

Using RL also allows PARL to be flexible and driven by its rewards, rather than follow a rule-based algorithm for its decision-making logic. PARL is evaluated against the set research goals and relevant works after Chapter 4 briefly introduces the system and memory configuration, workloads used in the experiments, and the evaluation criteria.

Chapter 4

System and Experiment Setup

This chapter describes the system configuration, main memory configuration, the workloads used in simulations, and metrics to be used for evaluation. The configuration and learning behavior of PARL are discussed here.

4.1 System and Simulator Configuration

The simulated system is configured as shown in Table 4.1. Ramulator, by design, follows the Double Data Rate (DDR) standard protocol to interface with synchronous DRAM (SDRAM) devices, as specified and maintained by the Joint Electron Device Engineering Council (JEDEC) [179].

CPU		*4-wide issue, single-core, 3.2 GHz*
Cache	LLC	*128KB, 8-way set-associative, 64B cacheline*
Main Memory	DRAM-only	*dual-channel DRAM on DDR4, 2400 MT/s*
	Hybrid	*DRAM + SCM-M on DDR4, 2400 MT/s*

Table 4.1: Simulated system configuration.

The Central Processing Unit (CPU) in Ramulator is a high-level model, as mentioned, and its main configuration parameters include the width of issue windows for non-memory instructions, its operating clock frequency, and the cache hierarchy. Since workloads are represented as traces of memory accesses filtered out up to the LLC in a real system, the simulated system is not configured with any core-level caches (i.e. L1 or L2). The trace files have been generated

outside of this work and were not reconsidered within the current scope. The description of workloads used in this work is given later in Section 4.3.

The scale of the system is adjusted to reflect that the trace files do not necessarily capture the entire workload. Thus, the LLC is set to 128KB and follows a standard cache policy [5, 197]. Smaller LLC also causes a greater number of accesses to the main memory, stressing it more for the purposes of this work. Data is fetched from the main memory in blocks of 64 bytes (i.e. cacheline), which also follows the standard main memory design.

4.2 Main Memory Configuration

This section introduces the timing specifications of the commands to control memory devices with cycle-level accuracy, including a brief overview of those commands. The memory capacity configurations used to evaluate HMM are also introduced here. These detailed fine-grained insights into the devices are not accessible across the abstractions in a real system.

A two-device HMM is instantiated in Ramulator, with both memory devices operating on the DDR4 protocol [179]. The baseline model for DRAM provided in Ramulator has been used in the literature and is adapted here without modifications [63, 142, 164]. The SCM-M device is a custom set of parameters representative of expected improvements to NVM devices. Its specifications were generated based on a PCM device and also used in [63], as part of the shared research scope at imec [198].

4.2.1 Memory Controller Operation

Each device has a dedicated memory controller operating under the default command scheduling and row access policies of Ramulator, i.e. first-ready first-come first-serve (FR-FCFS) and open-row. FR-FCFS prioritizes requests that are ready to be scheduled when all of the timing specifications are fulfilled. A request to a currently open row will be the first in the scheduling queue, before others requests to rows that are not yet activated. Otherwise, all commands follow the chronological order of the regular first-come first-serve (FCFS) policy.

The row access policy specifies whether a memory array row remains open indefinitely, unless another row must be opened - i.e. open-row. Closed-row policy would close a row as soon as no more access commands to it are pending. The open-row policy benefits the FR-FCFS scheduler, since consecutive requests to the same row will be prioritized to achieve lower effective access latency.

Operational and access commands to be scheduled and issued by a memory controller, as specified by the DDR standard, are listed below [142, 179].

- *ACT (activate)* - activate a row at the specified address in the memory array and transfer it into the row buffer
- *PRE (precharge)* - deactivate the memory array row currently open in the row buffer, rewriting the values back
- *REF (refresh)* - refresh the values stored in the memory array, precharging all open rows first
- *RD (read)* - fetch the data from a specified column address in the row buffer onto the data bus
- *WR (write)* - opposite of the read command

The controller issues these commands according to the timing specifications to model the memory device and its underlying technology characteristics.

4.2.2 Device Timing Specifications

Since both devices operate on the same protocol and interface, their specifications are expressed using the same terms too, as listed in Table 4.2 [179, 199].

Name	DRAM	SCM-M	Definition
t_{CL}	18	66	*Delay from an RD command to the first output bit*
t_{RCD}	18	66	*Delay from the row to the column address*
t_{RP}	18	18	*Delay after a PRE command*
t_{RAS}	39	66	*Delay from an ACT to a PRE command*
t_{WR}	18	180	*Delay from the last WR to a PRE command*
t_{CWL}	12	12	*Delay from a WR command to the first input bit*
t_{FAW}	26	104	*Delay to the fifth consecutive ACT command*

Table 4.2: A subset of DRAM and SCM-M device timing parameters operating on the DDR4 standard, in memory clock cycles.

The abbreviations for the mentioned timing parameters are expanded as follows:

- *CAS* - column access strobe
- *CL* - CAS read latency
- *CWL* - CAS write latency
- *RCD* - row address to column address delay
- *RP* - row precharge
- *RAS* - row access strobe
- *WR* - write recovery
- *FAW* - four active window or fifth activate window

A memory device is operated based on the timing and functional dependency among these parameters, impacting its effective access latency used as rewards for PARL. Depending on the accessed locations and their sequence, different command constraints and operational prerequisites must be upheld.

As an example, the latency for a read request is a sum of CL, RCD, and RP. In the best case, if an already open location is accessed (i.e. memory array row buffer hit), the latency would be equal to CL or as low as 18 memory clock cycles in DRAM. In the worst case, i.e. row buffer conflict, an open location must be closed and the requested one opened first, before that data can be accessed. This would result in a 3x higher latency of 54 clock cycles (i.e. RP+RCD+CL). Issuing an RP command must also satisfy the RAS condition to have the row open for a minimum number of clock cycles before it can be closed.

Furthermore, since DRAM requires refreshing, refresh commands (REF) will be injected by the memory controller, whenever necessary, based on the time elapsed since the last refresh. This would trigger additional commands forcing memory locations to be closed first, and to be reopened after the refresh. This would negatively impact the effective latency and these nuances are not easily captured by rule-based page placement policies. For example, accesses to different rows are deemed "random" and assigned to DRAM by APAMS. As it can be seen, this might not be as beneficial given the device specifications, despite the heavy influence from the chosen interface and protocols.

Row buffer conflicts in the SCM-M device used here are ∼2.5x slower than its hits (150 cc. vs 66 cc.). SCM-M row hits are also ∼3.6x slower than row hits of DRAM. Since NVM memory does not require a refresh, that can be advantageous over the momentarily unavailable DRAM during its refresh cycle. This highlights how "sequential" accesses are not necessarily suitable for one device or another, and highly depend on the commands to be issued.

In summary, if there are too many conflicting pending requests in the scheduling queue, then even a device with low access latency could perform poorly. Responsibilities of a memory controller are to optimize device-level management and command scheduling, but the internal structure of the devices is abstracted from the rest of the system. As a result, responding to access analysis insights does not directly influence device management. Therefore, PARL uses system-level observations to optimize device utilization without direct insights into the device inner workings.

4.2.3 Total Memory Capacity

HMM is based on the intent of reducing the DRAM presence due to its costs of operation [27] but offering at least the same or greater memory capacity. Three cases of main memory could be considered to reflect this, as listed below:

- *limited DRAM-only* - smallest capacity causing page faults
- *unlimited DRAM-only* - largest capacity avoiding page faults
- *HMM* - limited capacity DRAM extended by SCM-M to avoid page faults

The "limited" capacity is defined such that it is smaller than the working set size of most or all workloads [165, 166]. Due to frequent accesses to the storage level this configuration will negatively impact performance, but it should have lower associated costs (e.g. purchasing). DRAM of "unlimited" capacity would be the best for performance as it would eliminate all page faults. Main memory, besides the capacity and technology-dependent access latencies, also benefit from parallelized access channels to transfer more data at a time. As such, DRAM-only system is always in a dual-channel configuration.

The flat hierarchy of HMM makes it effectively dual-channel, with two independent single-channel DRAM and SCM-M devices in parallel. The total capacity of HMM matches the "unlimited" DRAM. To achieve this, a limited capacity DRAM is extended by an SCM-M device of greater capacity. This also mitigates the inaccuracies in modeling main memory to storage interactions.

Considering the recorded traces of workloads to be presented next and their high-level analysis - at least 512MB of the "unlimited" main memory capacity is necessary to eliminate all page faults. To stress the evaluated management policies, four configurations ranging from 8MB to 64MB are used to represent the limited capacity DRAM. The capacity of the SCM-M device only needs to be at least 512MB, and was set to 4GB for all experiments.

4.3 Overview of Workloads

This section introduces the workloads used in evaluation, as well as a proposed classification of these workloads based on their timeline of page allocation, that will be later used to organize discussions within Chapter 5 and Chapter 6.

4.3.1 List of Workloads Used

Overall, workloads are taken from the SPEC CPU2006 and Yahoo! Cloud Service benchmark suites [200, 201], also used in [63, 164]. As mentioned, the workloads had been previously recorded as trace files and no modifications were applied to them. Fig. 4.1 shows all workloads considered under various memory capacities, represented as the ratio of pages that can fit within the given memory capacity. This also corresponds to how Kleio defines its DRAM capacity to determine the initial page placement. These trace files were also adapted for simulations in Kleio, retaining the contained information.

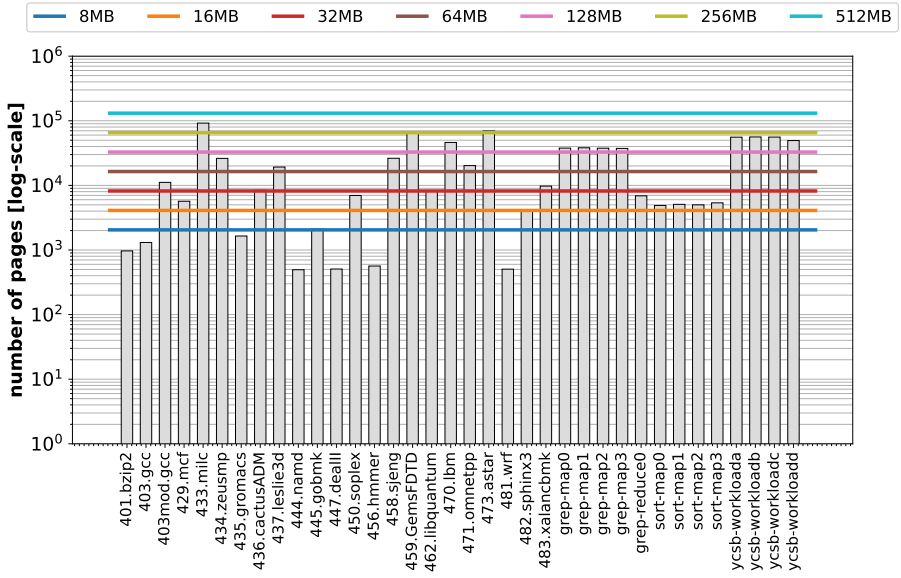


Figure 4.1: Ratio of workload pages that fit into a given memory capacity.

From this figure it can be seen that some workloads fit within 8MB of capacity - such as 401.bzip2 or 456.hmmer, while most require 16MB to 128MB. The last three workloads to require 512MB of memory are 433.milc, 473.astar, and 459.gemsFDTD with 71%, 94%, and 99% pages fitting in 256MB, respectively.

Table 4.3 lists the workloads used in the discussion of the results. The table includes the number of 4KB memory pages used by these workloads and the minimum memory capacity necessary to avoid page faults.

Category	Workload	Pages Used	Min. Capacity	Description
one-time	429.mcf	5,681	32MB	*combinatorial optimization*
	456.hmmer	565	8MB	*gene database search*
	462.libquantum	8,196	32MB	*quantum computer simulation*
periodic	433.milc	92,056	512MB	*quantum chromodynamics (QCD)*
	450.soplex	6,979	32MB	*simplex linear program (LP) solver*
	471.omnetpp	20,239	128MB	*discrete network event simulation*
	473.astar	69,587	256MB	*path finding in video games*
	grep-map2	37,657	128MB	*list search (variation)*
continuous	436.cactus	8,400	64MB	*physics equations solver*
	grep-reduce0	6,891	32MB	*list search (variation)*
	sort-map3	5,369	32MB	*list sort (variation)*

Table 4.3: Recorded trace files from the referenced benchmark suites [200, 201].

4.3.2 Workload Classification using Page Access Timeline

There are various criteria for workload profiling, but the chronological occurrence of new pages is of higher interest for initial page placement. Fig. 4.2 shows the timeline of accessing new pages based on the recorded timestamps in processor clock cycles. Three distinct profiles can be identified visually: *once*, *periodic*, and *continuous*. The timescale on the x-axis is proportional to each workload.

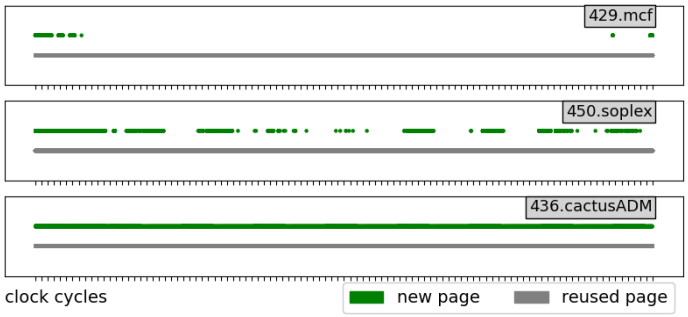


Figure 4.2: Page allocation and reuse timeline (proportional to each workload).

As it can be seen for 429.mcf, new pages (green) are allocated in the beginning of the workload lifetime, or *once*, and only reused (grey) for the rest of its runtime. In 450.soplex the new pages appear *periodically* throughout the runtime, while for 436.cactusADM new pages appear *continuously*. A list of all workloads is presented in Appendix B. This view highlights when new pages occur and aligns with the scope of this work, however, it does not capture when and how intensely each individual page will be reused.

To show how the chosen perspective or evaluation criteria influence conclusions about a workload, Fig. 4.3 shows the page access rates of the mentioned workloads in each category, similarly to Fig. 3.11.

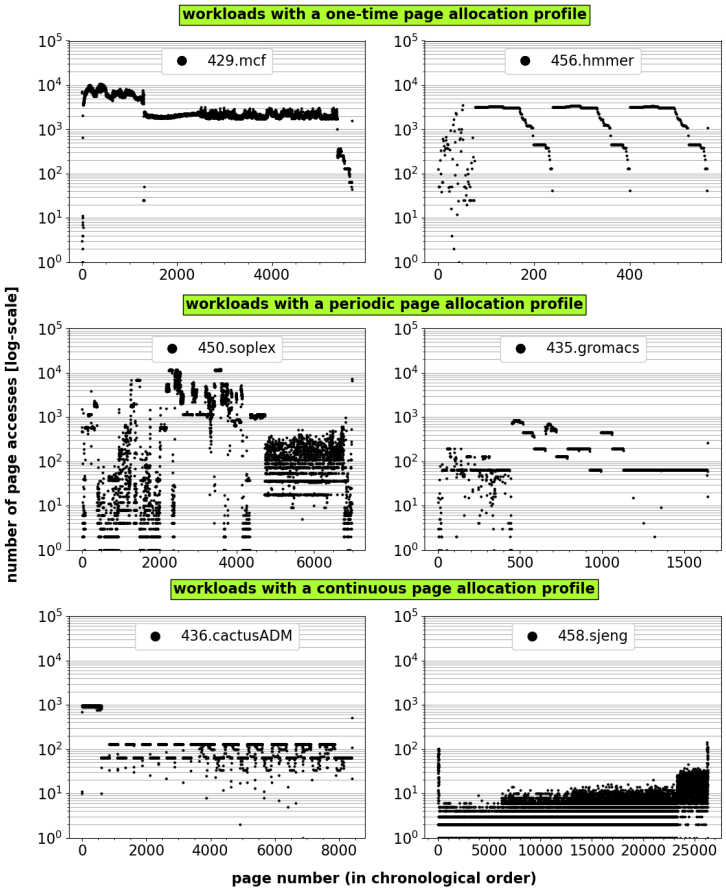


Figure 4.3: Page access rates for various workloads.

One observation from Fig. 4.2 and 4.3 is that access rates for workloads within the same category are not similar. Another observation is that early pages are not necessarily the most accessed, which could be an argument against DRAM-first and similar policies. For example, early pages in 435.gromacs have fewer and a more diverse range of accesses than later ones, or grep-map0 in Fig. 3.11, where later pages are accessed significantly more.

Isolated insights from these analyses are not likely to capture all the nuances of informed decision-making for HMM arbitration. It is necessary to combine qualitative coarse- and fine-grained approaches, as well as quantifiable statistics (e.g. Table 3.1) to formulate conclusions about the evaluated policies.

4.4 Evaluation Metrics

To evaluate the proposed arbitration method for HMM against the existing literature, it is compared against naive page arbitration policies and APAMS as policies for initial page placement based on heuristics. Then, PARL is compared with Kleio as an ML-based method for page scheduling.

If PARL satisfies the set research goals, then it should learn a page placement strategy that would improve the runtime of a workload compared to the other HMM management methods. As a consequence of this improvement, the learned placement should achieve higher utilization of the memory device than a migration-based approach. The comparison is based on runtime, DRAM hitrates, and DRAM access distribution, extracted from the output statistics from Ramulator.

Since the main function of the SCM-M device is to offset the reduced DRAM capacity, it is complementary to DRAM and would likely exhibit higher access latency. In this case, performance degradation is inevitable. Thus, the relative slowdown of runtime is used to compare the initial page arbitration methods, as they are all implemented in a cycle-accurate simulator. The slowdown is measured from the benchmark runtime of an unlimited DRAM-only main memory configuration. Here, runtime is the number of memory clock cycles to complete a full simulation episode. It is the only changing part in the total number of processor clock cycles, because in the given simulation environment the non-memory instructions and the time spent on them are independent of memory. For Kleio, the lack of fine-grained device-level modeling and its average latency modeling approach does not allow for a quantitative comparison of time.

Hitrate can be accurately measured in both simulators and used for evaluation of PARL and Kleio, including the initial placement policies. This metric is time-independent and reflects the access rates in a memory device. It is calculated as the ratio of requests serviced from the better performing device to all requests made by a workload. To improve the hitrate, Kleio can migrate pages, while PARL does not specifically target the hitrate, but as it learns to achieve the lowest access latency, the hitrate is also influenced. Therefore, if an improved initial page placement policy can achieve higher or similar hitrates, migration could have been avoided in those cases.

Lastly, unrestricted access to the source code of Ramulator enables insights into fine-grained statistics of the device utilization. These are expressed in row access statistics and memory access patterns in the original trace file. This comparison is used to reflect how a workload-generated data locality is not efficiently represented in the locality in the memory device. The improvements in runtime are achieved by improving this match.

4.5 Configuration of PARL

Due to the nature of RL agents, several episodes are necessary to learn and transition from exploring to the exploitation phase. As mentioned, the goal of PARL is to optimize the initial page placement and for the purposes of this work, all workloads and memory configurations are evaluated separately. The configurable parameters of an RL agent are adjusted to the values below:

$$exploration\ rate\ (\epsilon) = [0, 1] \qquad discount\ factor\ (\gamma) = 0.99$$

$$exploration\ decay\ rate\ (\lambda) = 0.01 \qquad learning\ rate\ (\alpha) = 0.1$$

These parameters are used in Eqn. (2.2) to calculate the exploration rate for every episode and Eqn. (3.3) for Q-value updates. In-depth analysis and tuning is planned for future work, however, these values were adjusted to make observing the behavior of PARL feasible, given the range of trace files, their sizes, and the time for a single simulation episode. As shown in Fig. 4.4, at the chosen decay rate of 0.01, the agent has the smoothest transition from exploration to exploitation compared to other λ values. The exploration rate in the later episodes also stays above zero to ensure continuous learning. Based on these decay plots, the agent is given a lifetime of 400 episodes or complete simulations to capture its entire learning behavior.

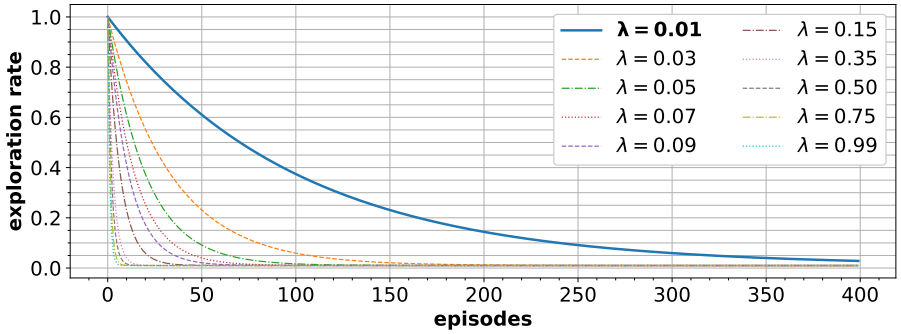


Figure 4.4: Exploration rate curve for different decay rates.

This is reflected in the observed learning curves for PARL, shown in Fig. 4.5 with workloads previously shown in Fig. 4.3. The time spent servicing memory accesses during the execution of a workload is steadily reduced, and converges to a stable value before the exploration rate of the agent is fully decayed. This can be seen from the "ϵ-scaled time" showing how the time in memory would have decreased if it followed the decay rate of ϵ. This blue line follows the Eqn. (2.2) but is bounded by the measured maximum and minimum time in

memory achieved by PARL for that workload. The measured time in memory is bound to the actual devices and how the workload performs under different initial page placements applied by PARL. While the exploration rate continues to decay with every episode, the lowest time in memory influenced by the initial page placement is achieved earlier.

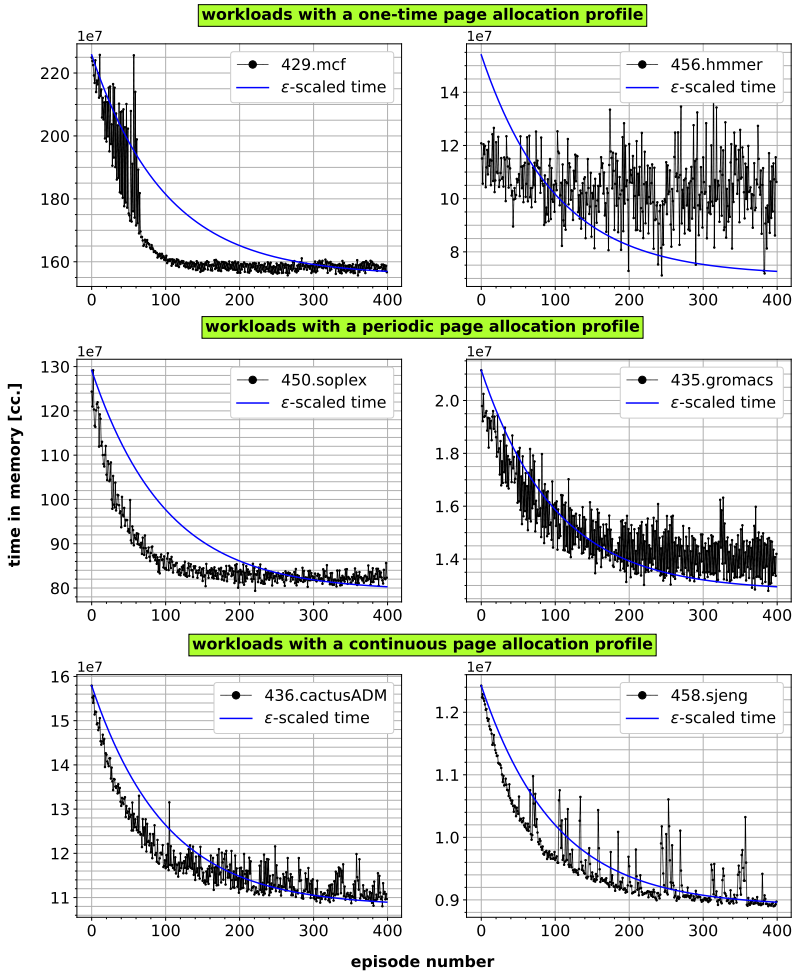


Figure 4.5: Learning curve of improving measured time in memory (in black). Blue lines represent if time in memory followed the ϵ decay equation, bound by the measured maximum and minimum time achieved by PARL for that workload.

The variance in the later episodes is due to the agent continuing to explore other placements and the non-zero ϵ, including how sensitive a workload can be to minor changes in the initial page placement due to its page access profile. This implies that if there was a change in either the workload or the system, the agent should adapt to it.

Lastly, workload 456.hmmer is notable as it does not show a clear reduction of time in memory. This workload has a small working set size of only 565 4KB pages, and therefore, it shows how the HMM approach benefits workloads that are bigger than the main memory capacity. Since this workload can be placed in a memory device of any evaluated capacity (as low as 8MB), there is no improvement to be learned by PARL for an HMM system with a fast DRAM and slightly slower SCM-M device.

4.6 Summary

In summary, this section presented the system configurations to be used, including the metrics to be used for evaluation (runtime in memory cycles, DRAM hitrates, etc.) and the configuration parameters of PARL. The learning behavior of the agent is influenced by the exploration rate decay, workload profile, and the physically achievable lowest time in memory. Next, improvements across various workloads and memory configurations will be presented and compared to policies from the existing literature.

Chapter 5

Evaluation: Initial Page Placement Policies

This chapter presents the results of experiments using PARL, compared against the arbitration policies from the existing literature that use heuristics for decision-making. As mentioned, these are *DRAM-first*, *Random*, *Round-Robin*, and APAMS. The evaluation is based on the runtime slowdown from the benchmark DRAM-only system, as defined previously, with supporting metrics based on the statistics of DRAM accesses - row buffer hits, misses, and conflicts.

The experiments are based on four HMM configurations, and all workloads simulated for each of them. First, results are discussed across all workloads averaged for each memory configuration. For a detailed view on the achieved performance and nuances of each policy under different configurations, the results are discussed in three sections that correspond to the three workload categories proposed earlier.

5.1 Overview

Fig. 5.1 reports the average improvement in runtime across all workloads for each HMM configuration. Based on the geometric mean across these averages, PARL achieves 110.21% slowdown from the benchmark, and outperforms APAMS by 39% and naive policies by 16% (DRAM-first), or 43% (random) and 42% (round-robin).

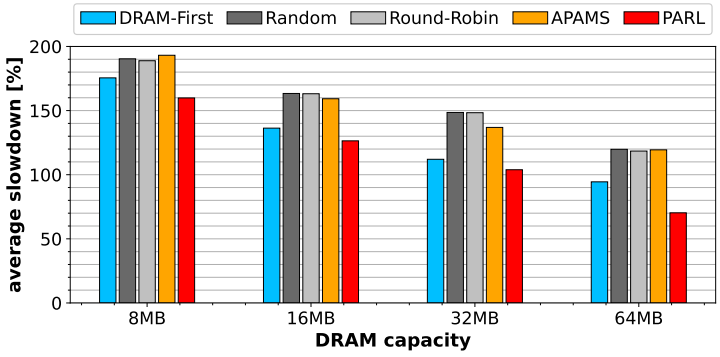


Figure 5.1: Average slowdown from the benchmark runtime, across all workloads, per HMM configuration.

The standard deviation for PARL across all workloads is 122% for 8MB and 16MB configurations, 114% for 32MB, and 73.8% for 64MB. The comparison to the other policies is shown in Fig. 5.2.

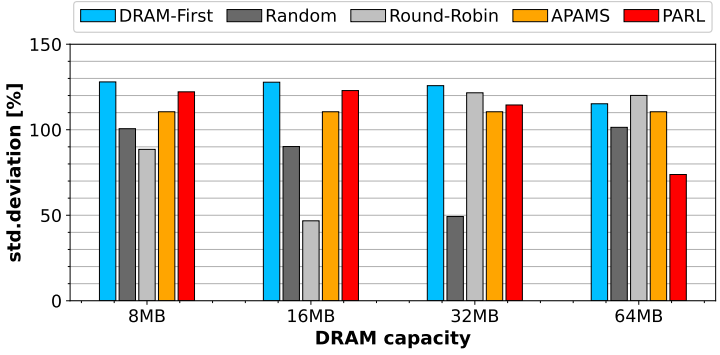


Figure 5.2: Standard deviation of the average slowdown from the benchmark runtime, across all workloads, per HMM configuration reported in Fig. 5.1.

These figures show that random and round-robin policies achieved worse runtime consistently across all workloads, since these policies are both rigid in their logic and degrade performance for all workloads in the same manner. Despite greater deviation, in this average view it can be seen that PARL achieves its goals, and that DRAM-first benefits from the DRAM bias. APAMS cannot show its strengths, since workloads vary in their size and profiles, influencing how much improvement is possible under each memory configuration. To capture more fine-grained insights, individual cases are analyzed next.

5.2 Workloads with one-time page allocation

Fig. 5.3 shows the relative slowdown from the DRAM-only baseline for workloads with one-time page allocation profile.

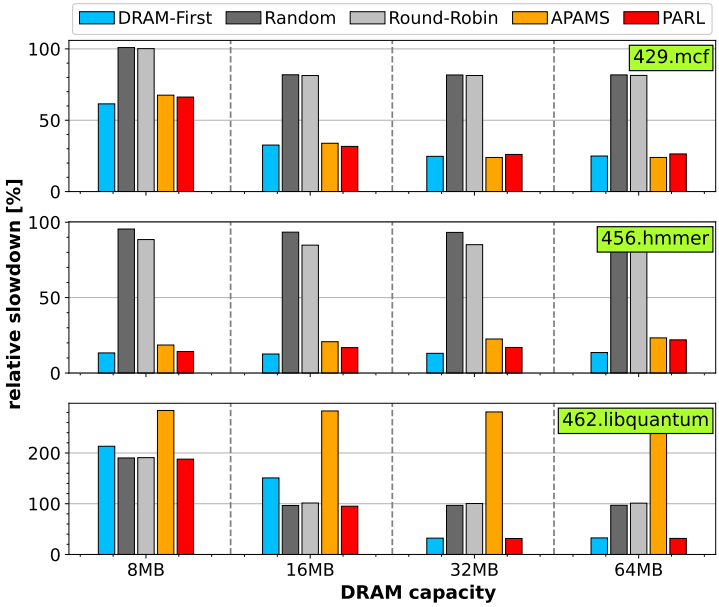


Figure 5.3: Relative slowdown for workloads with one-time page allocation.

Firstly, `456.hmmer` has been already mentioned in Section 4.5. All of its pages fit in the smallest DRAM capacity and under policies similar to DRAM-first, the runtime is only constrained by memory technology characteristics and device access parallelism. All other policies will use both devices and the higher latency of SCM-M would inevitably degrade the runtime. However, as noted earlier, such small workloads are not within the scope of HMM systems.

`429.mcf` and `462.libquantum`, on the other hand, exhibit contrasting access patterns, as seen in Fig. 4.3 and Fig. 5.4, respectively.

APAMS highlights this contrast as seen from the runtime it achieved for these two workloads. Fig. 5.5 shows that close to all of read requests are accessed at an interval of 64 bytes, which corresponds to the size of cachelines fetched from the main memory. In APAMS, this distance is considered sequential and it is assigned to SCM-M. Many requests to a device with higher access latency lead to a significant runtime slowdown.

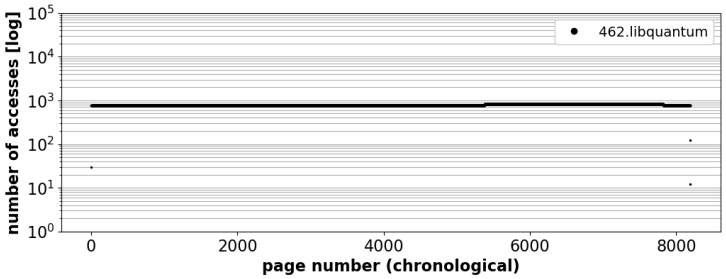


Figure 5.4: Page access rate for `462.libquantum`.

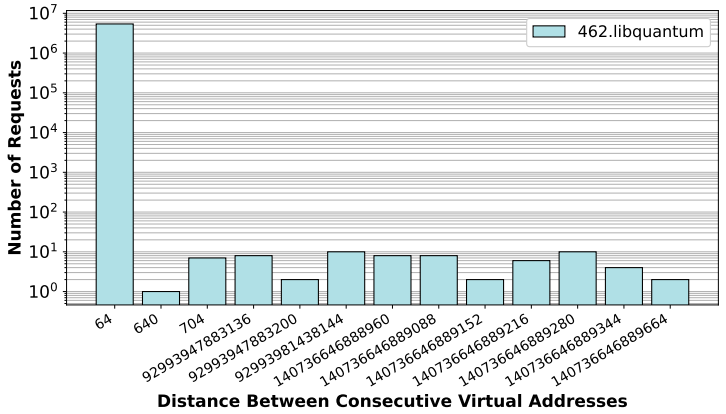


Figure 5.5: Distance of consecutive read addresses in `462.libquantum`.

This plot shows the distance between consecutive memory addresses accessed by a workload. The number of different distances and their ratio to the number of accessed addresses indicate the address regularity of a workload. Observed ratios of evaluated workloads range from <1% up to 58%. Low ratio corresponds to a highly regular, sequential profile, while the higher ratio reflects an irregular (random) access cadence. Memory management, in general, should improve the regularity of accesses in both VAS and the PAS, or translate addresses such that it benefits the device characteristics.

`429.mcf` has 357,817 different address distances, which is 2% of all accessed addresses. `462.libquantum` has 38 different distances (including both read and write locations) at a ratio of 0.0006% - the lowest ratio among all workloads. The next closest workload is `470.lbm`, with a "regular" access cadence ratio of 0.003% and 329 different distances.

If the decision-making logic of APAMS was reversed to forward these "sequential" requests to DRAM instead of SCM-M, then the slowdown of `462.libquantum` could be reduced to 23.3%. This is calculated by multiplying the number of regularly spaced requests with the average access latency of DRAM, without accounting for command scheduling impact. In practice, the slowdown was improved to 33.7%, aligning with this prediction.

Random and round-robin policies do not lead to any notable insights, as they lack a nuanced decision-making strategy. After considering the other policies, results achieved by PARL are discussed next.

PARL improves the slowdown for these workloads, except for those similar to `456.hmmer` as mentioned. To explain the improvements, it is necessary to analyze DRAM accesses. As mentioned, if memory accesses could exhibit improved locality prior to the device command scheduler, then the runtime could also be optimized by improving the effective access latency. Rewards based on it allow PARL to achieve this goal.

Fig. 5.6 shows the distribution of hits, misses, and conflicts in the row buffers of the DRAM device. The total height of each bar represents the total number of accesses to the device. It can be seen that both DRAM-first and APAMS resulted in slightly more DRAM accesses. PARL, however, learned to reduce the number of time-consuming row conflicts and misses while retaining a high hitrate. This leads to a lower effective access latency and runtime improvements.

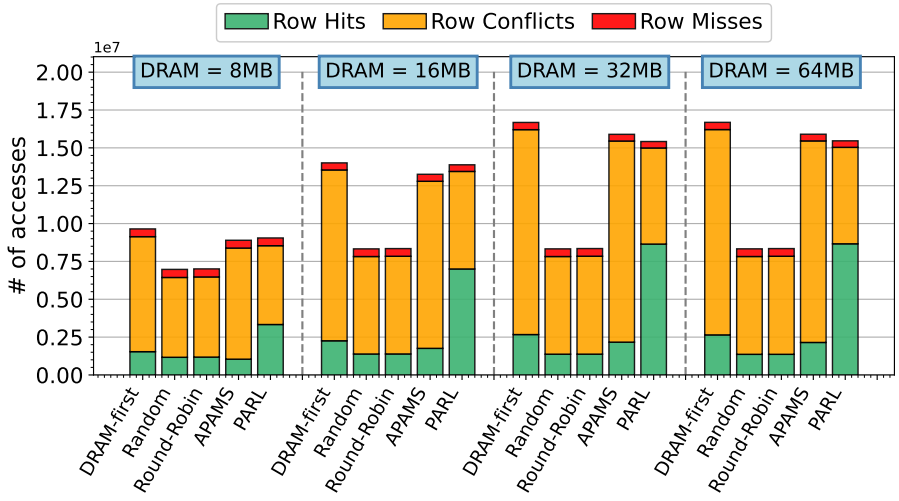


Figure 5.6: Distribution of DRAM accesses - `429.mcf`.

When DRAM capacity is large enough for `429.mcf` (i.e. 32MB), the average access latency is 92 cc. by following the DRAM-first policy. This is approximately double of the expected average DRAM latency (i.e. 35-50 cc.), which is a sign of high access intensity and lower access regularity in the device. PARL is able to reduce the average effective DRAM latency to 89 cc. which is significant given 16 million requests to only 6,000 pages. However, this improvement is mitigated by the SCM-M device that serviced 9% of all requests (i.e. 1.5 million requests) at an average latency of 145 cc. Some constraints of the memory technologies cannot be masked by the management policy and better timing specifications would enable further reduction of the slowdown.

Fig. 5.7 shows that for `462.libquantum` the number of accesses is similar between DRAM-first and PARL, despite the runtime difference of almost 50%. There are more row conflicts under DRAM-first and 29% higher latency than PARL (49 cc. vs 38 cc.). At 3.8 million accesses to DRAM (50% of this workload), PARL results in ∼850,000 fewer memory clock cycles.

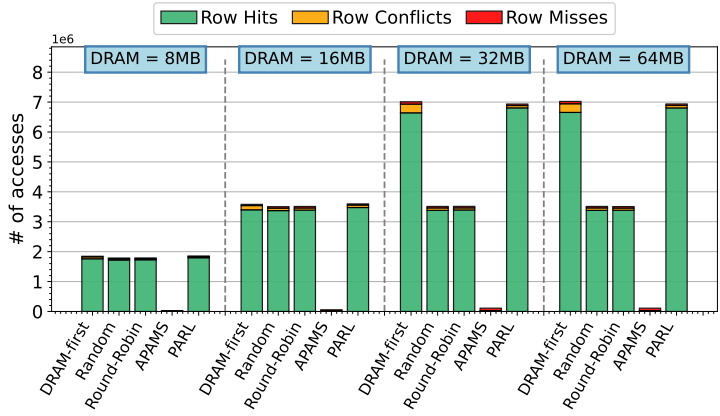


Figure 5.7: Distribution of DRAM accesses - `462.libquantum`.

In summary, DRAM-first policy is beneficial if the memory capacity is large enough for the given workload. APAMS and other policies with heuristics-based logic may perform better if the workload fits their assumptions. PARL, on the other hand, adapts to the constraints and optimizes the placement of pages.

O1: *PARL can learn an initial page placement that minimizes effective access latency by implicitly improving the row buffer locality for workloads with one-time page allocation profile.*

5.3 Workloads with periodic page allocation

Workloads with periodic page allocation could appear to have more complex page access profile than those within the one-time category. The runtime achieved by the considered policies for these workloads is shown in Fig. 5.8.

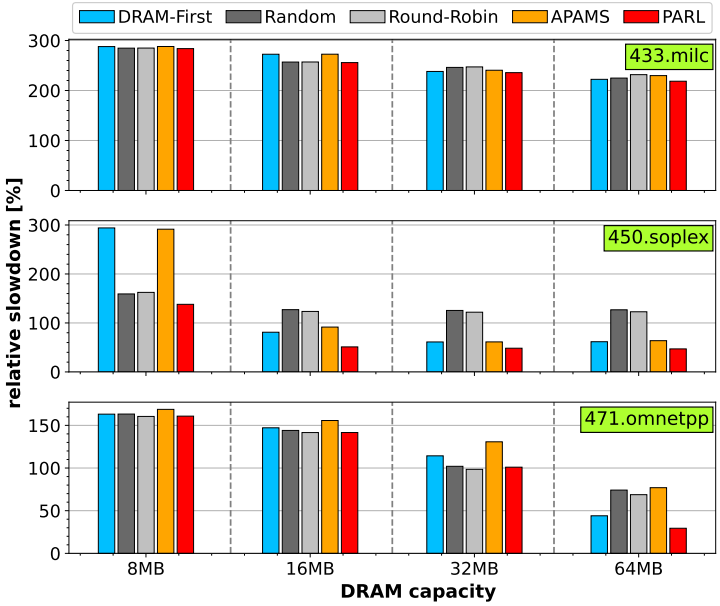


Figure 5.8: Relative slowdown for workloads with periodic page allocation.

Given that in this category pages can be allocated at later stages of the workload lifetime, it can be more difficult to account for that by only initial page placement. Aditionally, constrained memory capacities may also cause further limitations considering the pattern of page occurrences. The results show that PARL is able to adapt and learn an initial page placement strategy that improves the runtime even for these workloads with more complex page access patterns.

PARL learns to arbitrate between memory devices such that the slowdown is substantially improved, as seen in `450.soplex` or `471.omnetpp`. The former one requires 32MB of memory capacity, but at 8MB PARL achieves 2.5x more row hits than other policies and a significantly lower runtime than DRAM-first. At 16MB, PARL achieves the minimum runtime for this workload that is even lower than other policies at 32MB and greater capacities.

`471.omnetpp` is a workload with an address distance ratio of 5% (30th percentile among all workloads) and a higher reuse rate. Out of its 20,000 pages, a few are accessed up to 11,000 times, for a total of 4 million requests. Given this, and that 128MB of memory are necessary, its runtime is highly dependent on the constrained capacity of DRAM and which requests can be serviced from there. Fig. 5.9 shows that up to 32MB there is limited potential for improvement.

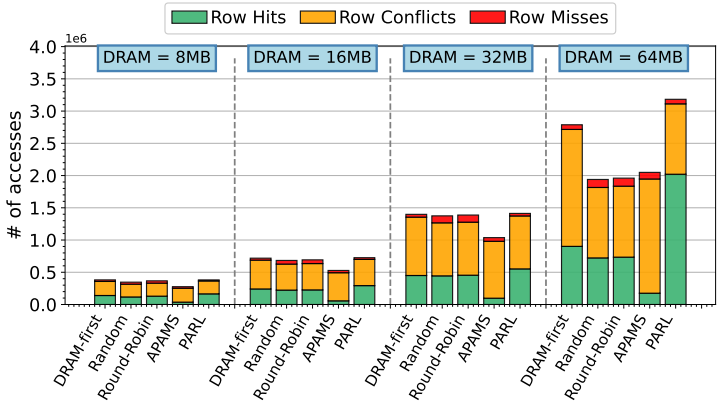


Figure 5.9: Distribution of DRAM accesses - `471.omnetpp`

The page access rate in Fig. 5.10 shows a spike of accesses in the beginning and at the end of the workload lifetime. At larger capacities more requests can be serviced from DRAM, but the new pages that appear later are unaccounted.

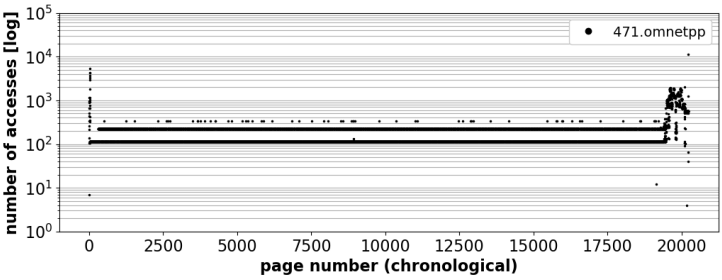


Figure 5.10: Page access rate for `471.omnetpp`.

At 64MB (16,384 pages) nearly 75% of the total number of requests can be serviced from DRAM, but later highly accessed pages still cannot be placed in DRAM. The chronological placement in DRAM-first also does not align well

with the less regular locality of accesses. PARL demonstrates higher hitrate or 400,000 more accesses, and the slowdown is reduced to ∼30%.

`433.milc` is a workload with a substantially larger working set size of 92,056 pages. Only 18% of it can be allocated to 64MB of DRAM and no policy achieves more than 25% hitrate. The achieved access distribution is shown in Fig. 5.11. If most accessed pages were placed in DRAM by a hypothetical Oracle, the hitrate would only improve to 38%, discounting the impact on scheduling.

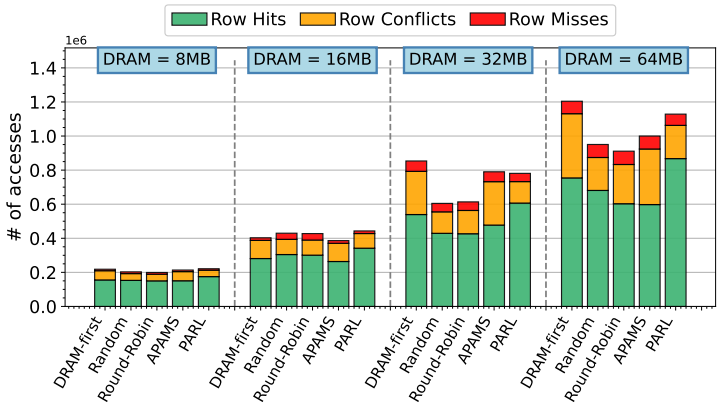


Figure 5.11: Distribution of DRAM accesses - `433.milc`.

Despite this, PARL still demonstrates the same effect of improving device utilization, considering the limited potential for reducing the runtime.

In summary, workloads with periodic page allocation and more dynamic access profiles confirm the previous observation about adaptability of PARL. When the workload working set size exceeds the available memory capacity, additional improvements could be achieved by migration. While it is subject to page selection and migration mechanism, impacts of which have been discussed already, higher hitrates achieved by PARL would reduce the number of migrations.

O2: *PARL also improves runtime of workloads with periodic page allocation. Higher hitrates at smaller runtime improvements indicate that the number of migrations for larger workloads can be reduced.*

5.4 Workloads with continuous page allocation

Lastly, the continuous page allocation category would correspond to workloads with potentially the largest working set sizes and most complex access profiles. Fig. 5.12 shows the slowdown for representative workloads in this category.

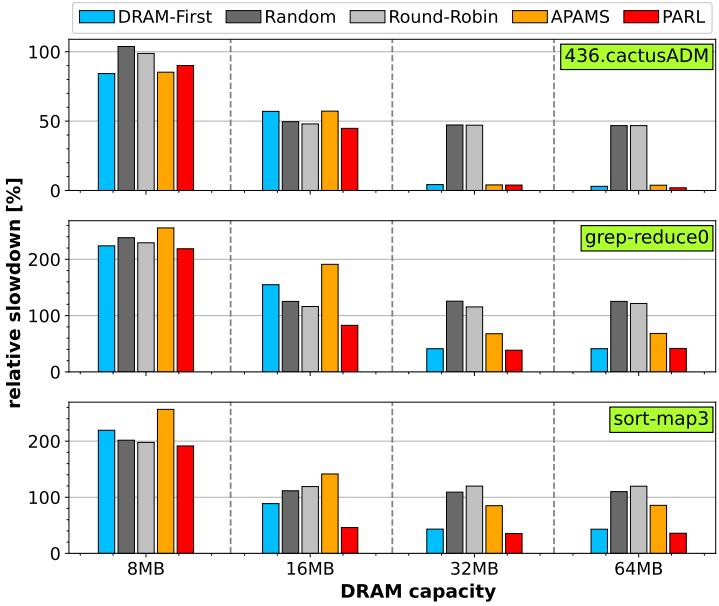


Figure 5.12: Relative slowdown for workloads with continuous page allocation.

PARL successfully reduces the slowdown for these workloads, and continues improving the runtime more than other polices as memory capacity is increased.

`grep-reduce0` and `sort-map3` exhibit higher randomness, given their ratios of address distances to the number of addresses at 49% and 45%, respectively. `436.cactusADM` has a lower number of different address distances, indicating slightly higher regularity, but a high number of accesses per page - 1.2 million accesses to only 8,400 pages. As a result, APAMS assigns enough requests to DRAM to achieve improvements in runtime for these workloads.

Fig. 5.13 shows that both APAMS and DRAM-first achieve similar hitrates, but with worse row buffer locality than PARL. While PARL services a lower number of requests from DRAM, its effective latency is significantly improved and as a result, the total runtime is also lower than other policies.

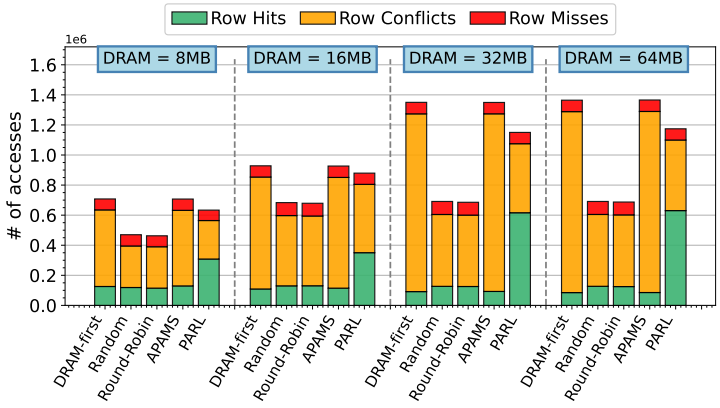


Figure 5.13: Distribution of DRAM accesses - 436.cactusADM.

Here it should be noted that some workloads do not make use of memory parallelism. For `436.cactusADM` similar to the benchmark dual-channel DRAM-only configuration. This behavior was observed in other workloads when comparing single- and dual-channel configurations. This indicates that memory can be unintentionally underutilized due to the workload design and adaptive control agents could exploit this shortcoming.

In summary, PARL has demonstrated its ability to improve workload runtime across different page allocation profiles, more than policies with rigid decision-making logic. It adapts to different circumstances and memory access patterns, rather than search for them or follow predetermined rules. Naive policies must be followed up by migration, but PARL has shown that significant hitrate improvements can be achieved with only the initial page placement.

O3: *PARL adapts to different memory configurations and workloads with dynamic profiles that exhibit periodic or continuous and non-uniform page access profiles.*

The analysis so far has considered all proposed page allocation profiles and workloads representative of those, given that there was reasonable improvement potential within limited memory capacities. However, there are also extremely large workloads, in any of the categories, that exceed the scope of considered memory configurations, such as `433.milc`. Additional examples, such as `473.astar` and `grep-map2`, are considered next to derive further insights on the behavior PARL and other policies.

5.5 Limitations of Initial Page Placement Policies

For extremely large workloads, initial page placement may have limited benefits without subsequent migration. If a workload significantly exceeds the constraints of the memory configuration, static placement of pages will inevitably have a limited impact on runtime. Given a large number of accessed pages in different phases of a workload, some pages that are not currently used or will not be used must be migrated to avoid runtime degradation.

As shown in Fig. 5.14, `473.astar` exhibits a significant slowdown with no noticeable improvement until DRAM capacity is increased to 64MB and beyond. PARL, random, and round-robin policies reduce the slowdown by ∼70% from the smallest considered DRAM capacity by distributing requests between the two devices. Both DRAM-first and APAMS maintain performance degradation of 200% or more across these configurations.

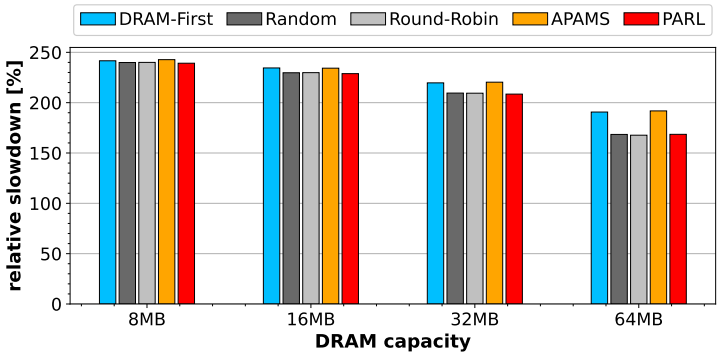


Figure 5.14: Relative slowdown of `473.astar` (continuous profile).

The page access rate of this workload is shown in Fig. 5.15. It can be seen that it accesses 70,000 pages with highly varied access rates. In this case, very limited memory capacity cannot account for a significant number of pages that occurs later in the workload lifetime. Nonetheless, PARL improves the utilization of the memory device as seen in the previous sections.

Fig. 5.16 shows that PARL achieves more row hits and slightly better hitrate, but the high access latency of the SCM-M device limits the runtime improvements. However, limited improvements are observed across all policies due to the scale and profile of `473.astar`. It has 505,551 different address distances (28% ratio), implying that this workload is highly irregular in the locations it accesses and this constrains the achievable physical locality.

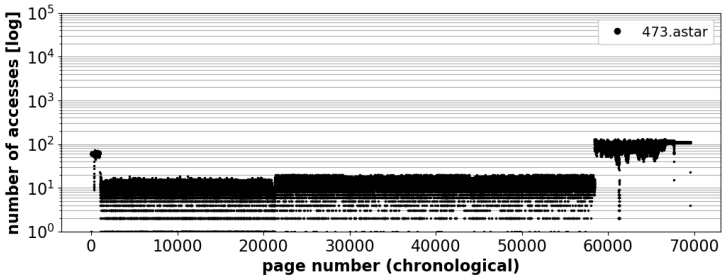


Figure 5.15: Page access rate for `473.astar`.

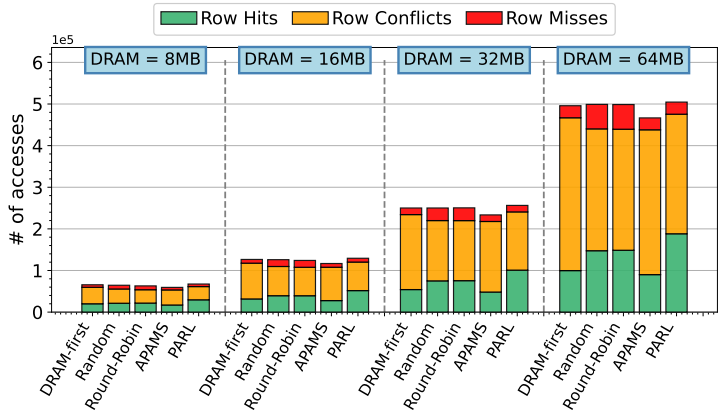


Figure 5.16: Distribution of DRAM accesses - `473.astar`.

Another example of extremely large workloads is `grep-map2`, with a continuous page allocation profile that gradually becomes more periodic. It also has 10x more memory accesses from only half as many pages as `473.astar`, with a page access rate profile matching `grep-map0` shown in Fig. 3.11. The runtime achieved for `grep-map2` is shown in Fig. 5.17.

PARL, random, and round-robin policies affect the runtime similarly, with DRAM-first and APAMS falling behind by 150%. Under such strict memory capacity constraints and page placement without migration, any runtime improvement is due the distribution of the requests among the devices. Fig. 5.18 shows that the available DRAM capacity dictates the achievable performance when the main memory capacity is not reasonably scaled to the working set size of the workload. Doubling the capacity from 32MB to 64MB results in more than a 3x increase in the number of requests.

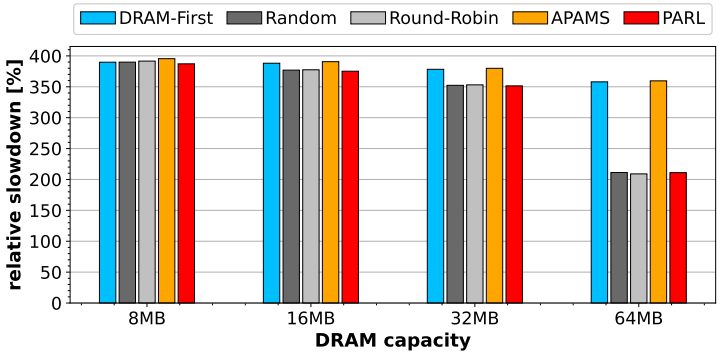


Figure 5.17: Relative slowdown - `grep-map2`.

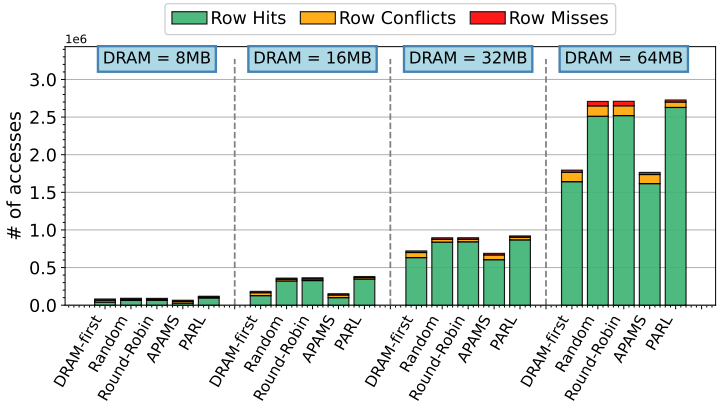


Figure 5.18: Distribution of DRAM accesses - `grep-map2`.

Due to the page access timeline (similar to Fig. 3.11), DRAM-first cannot improve the hitrate of `grep-map2`. This shows that the chronological order of pages does not correspond to the number of requests, and that such workloads depend on migration unless memory capacity is increased.

In summary, initial page allocation has significant improvement potential but not without a limit. For extremely large workloads migration is necessary, but PARL demonstrates hitrate improvements without it.

O4: *PARL can adapt to the limited range of improvements dictated by the scale of the system and the size of the workload.*

5.6 Summary

As it was outlined in the motivation behind this work, HMM systems need a flexible and adaptive management methods. The goal is to develop a control agent that can make decisions without following predefined rules that are based on the expected behavior of a workload, or a given system configurations.

The identified gap in the HMM management literature is that proposed methods are migration-driven and follow heuristics around memory accesses and memory devices, including their attributes and specifications. The proposal in this work was to develop a control agent that improves the initial page placement in HMM by using an adaptive decision-making method. The goal is to improve workload runtime and reduce the burden of migration. RL was presented as a promising alternative among other ML methods. However, the state definition of an RL agent has to be adjusted such that it satisfies the MDP properties and does not rely on the memory accesses.

Since the goal of memory management is to improve utilization of system resources, observing them is better suited to the envisioned control agent. As such, PARL has been designed for device arbitration in HMM using system-level state observations and evaluated against other initial page placement policies from the literature in this chapter.

The results have shown that PARL satisfies the goals set. It has been shown that memory management does not have to rely on analyzing memory access profiles, but understanding them is indispensable when evaluating the various management methods. The runtime improvements were achieved due to improving the effective access latency as defined by the reward function of the RL agent. Observing the state of the memory system allows PARL to improve the latency of servicing a request by implicitly improving the row buffer locality, which is a hidden physical property of the memory devices.

After evaluating PARL in a cycle-accurate environment against other initial page placement policies, it can be concluded that PARL has achieved its goals. It adapted to different workload profiles and achieved improvements that are possible within the constraints set by the system, such as whether the memory capacity is reasonable in comparison to the working set size of a workload.

Next, PARL is contrasted with an ML-based page scheduling method implemented in a model-based environment that does not allow for a fair runtime comparison. Thus, DRAM hitrates will be evaluated across the same set of workloads and memory configurations.

Chapter 6

Evaluation: Page Migration Policies using Machine Learning

The previous evaluation chapter established that system-level attributes can enable an RL agent to learn an initial HMM page placement optimizing workload runtime. In this chapter, PARL is compared to Kleio - the state-of-the-art HMM page management proposal based on machine learning. An overview and design details of Kleio were presented earlier in Section 3.3.

To support the comparison across two different simulators and to account for potential inaccuracies in modeling memory processes, a hypothetical Oracle is developed within Ramulator. This Oracle has pre-existing knowledge of the most accessed pages in a workload, and will place them in the DRAM device to maximize its hitrate. The Oracle is also designed for initial page placement only, and does not migrate pages. This does not take into account the time taken to service those requests, as it was shown previously that higher hitrate may not lead to better runtime if device-level locality is not optimized.

This chapter continues with an evaluation of the DRAM hitrate, followed up by a relative (qualitative) comparison of the runtime slowdown achieved by Kleio and PARL in their respective environment.

6.1 Overview

As it would be expected from a migration-based management method, Kleio improves DRAM hitrates across most workloads and system configurations. Table 6.1 gives an overview of simulated workloads that had a significant number of migrations and exceeded the minimum of 64MB of memory capacity to prevent immediate placements in DRAM by either of the methods.

Category	Workload	Pages Used	Min. Capacity	Description
continuous	403mod.gcc	11,135	64MB	*C compiler (increased trace length)*
continuous	434.zeusmp	26,192	128MB	*magnetohydrodynamics*
continuous	436.cactusADM	8,400	64MB	*physics equations solver*
continuous/periodic	437.leslie3D	19,224	128MB	*computational fluid dynamics (CFD)*
continuous	458.sjeng	26,294	128MB	*game tree search, pattern recognition*
periodic	471.omnetpp	20,329	128MB	*discrete network event simulation*
continuous	473.astar	69,587	512MB	*path finding in video games*
periodic	483.xalancmbk	9,754	64MB	*XLST document processor*

Table 6.1: Recorded trace files from the referenced benchmark suites [200, 201].

However, due to the design of Kleio, some large workloads with dynamic profiles could not be simulated in a reasonable amount of time under any of the memory configurations and be available for a full evaluation here.

`ycsb` workloads with four variations "a", "b", "c", and "d", require 256MB and use 50,000-56,000 pages with a continuous page allocation profile. All four exhibit higher randomness in accessed locations, with 300,000-380,000 different address distances and 34%-38% ratios. Variations "a" and "b" could be simulated for 8-32MB configurations but exhibited an unexpectedly low number of migrations - 104 and 110, respectively. The hitrates were at 55%-62%, compared to 26%-47% of PARL. This indicates that these workloads have more predictable access patterns that do not generate errors in the history-based predictor and benefit from the oracle replacing LSTMs in Kleio. At 64MB PARL achieved ∼60% hitrate, but Kleio did not finish the simulations after over a week of time.

On the other hand, a smaller workload, such as `462.libquantum` with highly regular accessed locations, resulted in 168,718 migrations for 8MB and 16MB configurations. Kleio had 36.09% hitrate in DRAM for both, while PARL had 26.46% and 51.24%, respectively. This workload fits within 32MB of memory and needs no migration since no "cold" pages were detected to demote to SCM-M. All pages are placed in DRAM - hence, this is a 100% hitrate by definition, while PARL achieved 99% while observing the system states.

These observations show that migration-driven methods are more complex to accurately model and execute. The impact of using LSTM models instead of the oracle-based version would be even more detrimental to simulation times.

6.2 Comparison of DRAM Hitrates

Fig. 6.1 shows the DRAM hitrate under Kleio, PARL, and the hypothetical Oracle, and the number of migrations by Kleio.

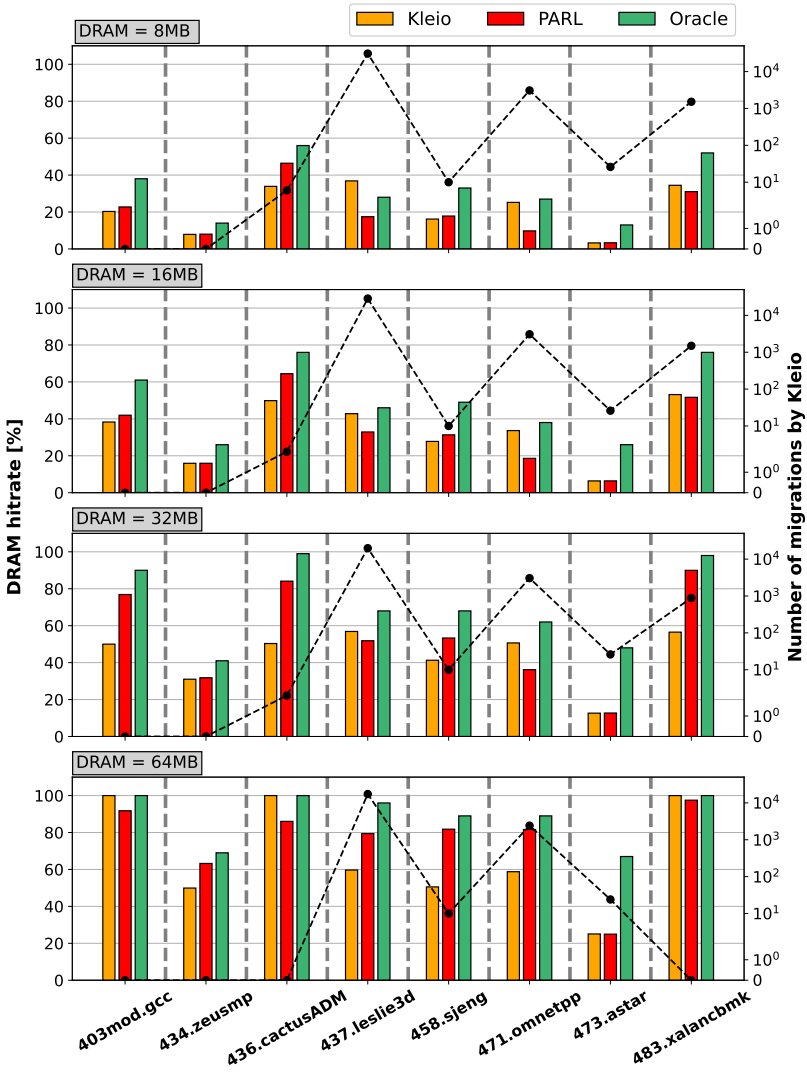


Figure 6.1: DRAM hitrates of Kleio, PARL, and Oracle (left y-axis) with an overlay of the number of migrations performed by Kleio (right y-axis, semi-log)

As it can be seen, Kleio does not outperform PARL or the Oracle right away, unless a workload is extremely large and exceeds the potential of initial page placement. This shows that workload runtime as the objective for improvement can also lead to better hitrates. The first observation from this figure is that the hypothetical Oracle demonstrates the maximum DRAM hitrates achievable under initial page placement exclusively. In most cases, this is higher than both Kleio and PARL. However, since higher hitrates do not guarantee lower effective access latency, as it was shown in a cycle-accurate environment, using it as the main objective might not be the most optimal choice.

Overall, this plot confirms an earlier conclusion that workloads with more dynamic access profiles are likely to migrate pages more often. `403mod.gcc` and `434.zeusmp` have profiles that align with the predictive mechanisms in Kleio and do not need migration.

Remaining workloads have more complex patterns that are not observable from the page access rate plots shown earlier (Fig. 3.11 or 5.15). The total access rates do not capture the momentary intensity which is used by Kleio to decide on migration. As an example, the highest number of migrations (after `462.libquantum`) was detected for `grep-map1` that follows the profile in Fig. 3.11. 89,922 migrations were necessary for 16MB of memory and 65,968 for 32MB. Other variations did not complete the simulations, with only `grep-map3` resulting in 65,834 migrations at 32MB. These migrations allowed Kleio to achieve 78%-80% hitrate at larger capacities. In comparison, PARL improved the hitrate from 1.5% to 35% across all considered memory configurations and the Oracle could achieve hypothetical 35%-80% hitrates.

Another previously mentioned expectation about memory capacities can also be confirmed from Fig. 6.1. Kleio uses migration to improve the hitrate under smaller memory capacities, especially if the workload has hot pages in the later phases. Migration can clearly be advantageous for such workloads, e.g. `471.omnetpp` or `grep-map`$_i$. For 8MB and 16MB of DRAM Kleio achieves hitrate on par with the Oracle (25% and 34%), while PARL could improve it up to 10% and 19%. Since every location is accessed more than once throughout this trace file, dynamically migrating these pages to DRAM is more beneficial to the hitrate than the static distribution by the initial page placement policies. At 32MB, 40% of pages in this workload can be placed within DRAM and this allows PARL to achieve a 36% hitrate, while Kleio and the Oracle achieve 50% and 60%, by migration and prior knowledge, respectively.

Improvements by Kleio do not continue to scale, though, as it can be seen from the hitrates at 64MB where both PARL and the Oracle outperform it by $\sim$25%. This is also observed for workloads such as `437.leslie3d` and `483.xalancmbk`. For the former, Kleio performed more than 30,000 migrations and outperformed

PARL by 19% and the Oracle by 10% for smaller capacities. The number of migrations and the hitrate do not improve significantly despite increased capacity. PARL demonstrates that it adapts to the change in the system and improves runtime together with the hitrate.

`434.zeusmp` and `473.astar` show that for extremely large workloads, neither improved initial page placement nor the subsequent migration can overcome the limitations of memory capacity. This is also confirmed by the Oracle achieving similarly low hitrates until memory capacity is increased. `473.astar` also shows how prior knowledge of the hot pages in the later phases of the workload is beneficial (Fig. 5.15), while no other policy is able to predict such access profiles.

Table 6.2 reports the geometric mean of hitrates across all evaluated workloads, per memory configuration. Migration results in marginally better hitrate when DRAM capacity is constrained, i.e. up to 2.5%. Increasing the memory capacity highlights the impact of initial allocation and adaptive methods.The Oracle is 14%-22% better than PARL, and 11%-29% better than Kleio across different memory configurations.

method	**8 MB**	**16 MB**	**32 MB**	**64 MB**
Kleio	17.60%	28.31%	40.05%	62.12%
PARL	15.08%	26.89%	46.93%	70.98%
Oracle	29.02%	46.10%	68.64%	87.75%

Table 6.2: Geometric mean of DRAM hitrates across all evaluated workloads for every memory configuration.

In summary, it is clear that migration can be beneficial when DRAM capacity is constrained, but its time and other associated costs are not considered in detail. Furthermore, increasing the DRAM hitrate does not always align with the goal of improving the runtime, and the Oracle has shown that migration does not always maximize hitrates either.

O5: *Improved initial page placement of PARL can outperform the hitrates of the migration-based approach of Kleio. It also achieves DRAM hitrates comparable to Kleio and the Oracle under the strictest capacity constraints.*

Having evaluated the PARL against another ML-based method including migration, the next section presents an overview of the runtime slowdown and recorded improvements by Kleio within its own environment. These results are contrasted with PARL for a high-level analysis.

6.3 Comparison of Workload Runtime

Fig. 6.2 shows the runtime improvements by Kleio from its own benchmark DRAM-only memory configuration. Due to the scheduling intervals and continuous migration overhead, Kleio always reports a minimum of 60% slowdown from the DRAM-only configuration even if capacity is large enough to fit the workload.

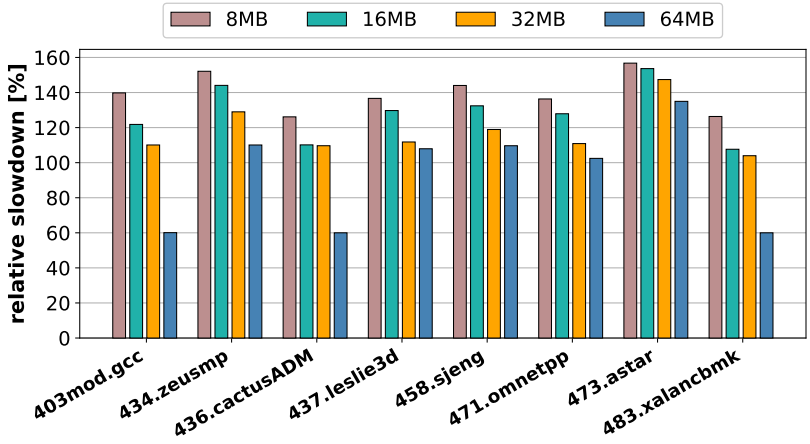


Figure 6.2: Relative runtime slowdown for Kleio from its own benchmark DRAM-only configuration.

Kleio shows that the naive round-robin page allocation can be improved by the means of migration. For `473.astar` and `434.zeusmp`, however, it performs very few page migrations and the slowdown is not reduced as DRAM capacity is increased.

Qualitatively comparing the runtime improvements across different memory configurations for the same workload, it can be seen that PARL adapts better than Kleio. As DRAM capacity was quadrupled, Kleio reduced the runtime by 40% for either of these workloads. In contrast, PARL improved the runtime by 80% for `473.astar` (Fig. 5.14) and by 240% for `434.zeusmp` (Fig. 6.3).

The limited improvements can be explained by whether enough pages have been identified as high-priority and used in the training of LSTM models or as the internal oracle. As it was the case for heuristics-based policies, assumptions on access patterns influence the efficiency of any control agent, including ML-based ones. Kleio proposes a sophisticated model for *predicting* hotness of high-priority pages, but relies on simpler methods to *identify* those pages.

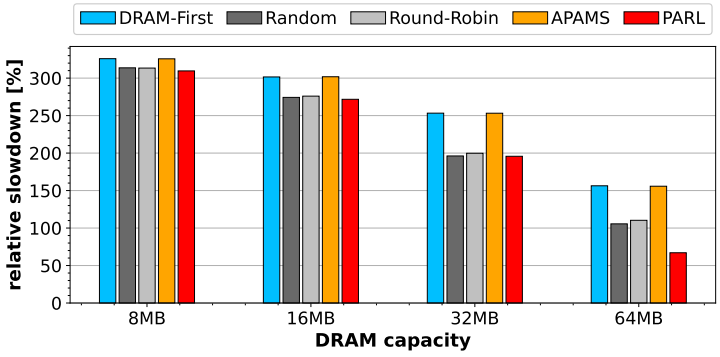


Figure 6.3: Relative slowdown improvements by PARL - 434.zeusmp.

For `473.astar`, 15 access patterns were identified and analyzed by Kleio, corresponding to 515 pages out of nearly 70,000 pages. Only 13 pages were managed by the oracle (or the LSTM model). For `434.zeusmp`, there was only one pattern matching 1,460 pages out of 26,000. Out of 8,400 pages in `436.cactusADM`, 5 patterns were identified (996 pages). All workloads had a small number of pages offloaded to the oracle, and the decision-making is mainly driven by the history-based predictor. In contrast, a simple analysis of address distances results in 505,551 different patterns for `473.astar` (28% of all addresses), 7,776 patterns for `434.zeusmp` (0.21%), and 2,168 patterns (0.17%) for `436.cactusADM`.

The authors of Kleio have investigated improvements to page selection in their subsequent studies [163, 202]. However, pattern analysis remains tailored to the workloads, memory configurations, and assumptions on a match between them.

In summary, Kleio demonstrates that migration can be necessary and effective in improving the hitrate and runtime. However, it does not consider the cost and overhead of migration and shortcomings of pattern analysis. Moreover, the runtime improvements are limited unless the workload is sufficiently small to be placed in DRAM.

O6: *PARL has demonstrated that it can match and outperform hitrates of a migration-based approach, and that better initial page allocation can reduce the need for migration. Moreover, PARL is better at adapting to different memory configurations and maximizes the improvements within the given constraints.*

6.4 Summary

Kleio offers a method to profile workloads and develops an ML or oracle model to represent the access frequency of the high-priority pages. The goal of Kleio is to predict page hotness in the next scheduling interval, and migrate pages between HMM devices accordingly. Given that many workloads use a large number of pages, Kleio applies a selection mechanism to identify high priority pages based on the error rate of a history-based predictor.

The proposed LSTM models for these pages are effective in their predictions, but susceptible to the shortcomings of relying on memory accesses and using request-level information for training. As it has been mentioned in the earlier chapters, while ML can be useful in processing data and deriving insights, it is not always applicable as a control agent. In those cases, a rule-based decision-making has to be designed around the chosen ML method. Then, the resulting control agent faces the challenges of biases and rigidity in its actions, and the resources necessary to support ML.

Moreover, Kleio does not model or account for the impact its placements could have on the memory devices. While migration improves the hitrate, it is unkown whether it will infer any penalties in the device scheduler and increase the effective access latency.

As it was posed in the research goals of this work, analyzing memory accesses is not necessarily the only method of managing memory and source of information for decision-making. PARL achieves similar or better results by the means of observing the system around it, which allows it to adapt and improve workload runtime and memory utilization by only improving the initial page placement. Hitrates similar or even better than Kleio indicate that before migration needs to be invoked, memory management proposals should explore how to maximize the utilization of the resources available. If the design of an RL (or any ML) agent is specified to match the nature of the task at hand, it could be successful in its objectives and remain feasible for deployment.

With this chapter, the design (Chapter 3-4) and evaluation (Chapters 5-6) of the method proposed to address the identified gap in HMM management (Chapters 1-2) have been concluded. Next, Chapter 7 close this thesis.

Chapter 7

Conclusion

This chapter concludes this thesis, and summarizes the motivation and the findings by reflecting on the benefits and potential shortcomings of the proposal. Suggestions on improving the current proposal and future work are discussed too, and two sets of preliminary exploration results are provided in Appendix C.

7.1 Conclusion

In this work, the initial page placement in an HMM system was investigated. PARL is the proposed method to learn a distribution of requests to memory among the available devices without analyzing the requests or following explicit assumptions about the access patterns.

The research goals set in this work aimed to design an adaptive policymaker guided by a predefined target rather than following rules. The objective set for this policymaker is to improve the initial data placement and reduce the burden of data being migrated within or out of the memory system. The outcome of this policymaker achieving its objective should be reflected in the improved runtime of workloads executed within a hybrid memory system.

These goals are derived from the extensive analysis of the existing literature around the design of HMM and applicable management methods. The gap identified within these works is two-fold: (1) reliance on migration, starting from initial placement policies that are simple in their nature and (2) lack of flexible and adaptive management methods, opting for rule-based decision-making.

Overall, it can be concluded that this work, and the proposed RL agent PARL, have met the set goals and addressed the identified gap. The results of conducted experiments discussed in Chapters 5 and 6 have shown that PARL has consistently adapted to the given constraints of its environment (i.e. the system and memory configurations), and exploited the potential improvements achievable for the workloads executed in that environment. Chapter 3 and Section 4.5 have shown that the design of an RL agent can be suitable for main memory systems. By observing the utilization of the available resources, memory management can be a task feasible for an ML approach, reducing or eliminating the biases and assumptions of human-driven algorithms.

In particular, the proposed RL environment state definition using system-level metrics reduces the implementation overhead of PARL and its decision-making space, contrary to the claims in the literature. This allows the agent to not rely on the resource-intensive analysis of patterns in memory accesses or their attributes. A slight departure from the classic Q-value update function also enables PARL to be more optimistic in its learning by following the received rewards. The proposed reward function based on the effective access latency achieves improvements in row buffer locality, which is a property of the memory devices inaccessible to the system.

As it was mentioned, PARL is a proposal on device arbitration in HMM and is concerned with memory management, rather than a proposal on page allocation methods which is concerned with memory accesses. Next, main points of reflection on this work are briefly discussed, including its strengths and limitations.

Explainability of Machine Learning Methods

ML methods are commonly criticized for the lack of or obfuscated interpretability. Due to the automated behavior and the scale of data sets used in these methods, explaining why a decision has been made is often an issue. PARL does not provide explicit information on how memory access patterns are distributed between the memory devices, in contrast to literature that proposes rules for this by human designers. The main advantage of PARL is that it is not programmed to identify the temporal or spatial locality of accesses. As a result, the behavior of this agent and its decision-making logic is explainable in terms of the quantifiable rewards it receives from the system, the utilization state of the resources that it observes, and the clearly defined set of actions available.

Learning Behavior of Reinforcement Learning Agents

For the purposes of this work, the experiments were set up such that the RL arbiter learns a policy through iterated simulations of each evaluated workload. This allows for a more controlled observation on how the agent performs, and it was shown that it adapts to every workload and every memory configuration it was deployed on. This work does not explicitly explore methods and improvements to the learning and training processes of ML methods. While there is undoubtedly potential for improvements, this process is also inherent to all similar control agents. The absence of pattern-specific information in the observations and the results across various workloads indicates that the state definitions of the arbiter are sufficiently generalizable. This point is also reflected in the future work.

Effectiveness of PARL

PARL was able to minimize the runtime to the desired benchmark levels by improving the access distribution and reducing the row buffer conflicts. As a result, together with runtime, the energy efficiency of the memory system is expected to be improved too. Effective latency is reduced by eliminating unnecessary device-level commands and maximizing the impact of the scheduling algorithm in the memory controller - i.e. fewer operational commands (e.g. ACT or PRE). This shows that PARL influences the system more than only in the performance aspect and its effect is extended beyond the defined objective.

Implications of PARL

Design and evaluation of PARL is mostly aligned with the second half of the title of this thesis on use of ML in HMM systems. The first half of the title, or the design and optimization of hybrid memory systems, is reflected in the presented motivation and background material, and in the outcome of this work. Design and optimization of complex and nuanced systems such as HMM implies comparisons in multiple dimensions that are often orthogonal to each other. System hierarchy levels such as workloads, compilers, processors, memory, etc., follow agreed upon design guidelines, interfaces, and protocols. However, the internal designs are often abstracted from each other, and follow the limitations of their technologies or implementation mechanisms. To enable an informed design path and feedback loop across these abstracted layers, a unified evaluation framework is necessary to ensure a fair assessment of different technologies and management policies.

The literature on HMM proposes methods tailored to the given constraints, and may have limited applicability when they are changed. As a result, a method designed to maximize the lifetime of a memory device with endurance concerns would not be applicable to an HMM system that employs devices without such limitations. By showing that PARL performed predictably and adapted to various workloads and memory configurations, it can be used or extended as an evaluation framework for different memory technologies. This ensures that the performance or energy consumption optimization potential of the changes in memory or other configurable parameters are maximized by an agnostic control agent, such as PARL.

In conclusion, PARL has achieved the set research goals and has potential for further improvement. The proposed future extensions are briefly discussed next.

7.2 Future Work

Considering the work accomplished and insights gained, a few potential future developments are proposed here. This includes improvements to the current design of PARL, as well as extensions to accommodate new research directions.

Potential incremental improvements to the agent include designing a different exploration rate decay mechanism. While exponential decay with non-zero minimum has been sufficient to observe exploratory behavior later in the lifetime of the agent, it is also possible to dynamically spike the exploration rate to force the agent to try different actions and refresh its Q-values.

Deployment Potential

PARL has been developed in a simulated environment, and thus, relied on scaled down configuration of the system and workloads. Consequently, the following improvements and extensions are possible. First, the scope of considered system configurations can be extended to either larger trace files (with limited accuracy in Ramulator) or use full system simulators such as gem5 [141]. Next, an OS-level (e.g. kernel) adaptation of PARL can be developed to evaluate its impact on a real system, with workloads executed online instead of trace file playback. This would also avoid issues of longer simulation times with real workloads in gem5. Lastly, an RTL-level implementation of PARL would provide fine-grained estimates on its implementation cost and hardware resources necessary. However, this could result in an arbiter placement that is device-level if implemented together with a corresponding memory controller and does not necessarily align with the proposed scope of the arbitration agent.

Learning in Dynamic Environments

Following the previous points, an RL agent deployed in a real system would be exposed to concurrently executed workloads, and adapt its behavior online. To minimize the potential impact on performance while the arbiter is adapting, it can be deployed with Q-values pre-trained (initialized) for different types of workloads and desired optimization targets. In order to avoid a biased starting point due to potentially misidentifying a workload type, the exploration rate in this phase can be set higher or momentarily increased. This would force the agent to learn more, rather than follow the path of the existing Q-values.

Reward Functions

In the current implementation of PARL, the reward function is performance-oriented as it is based on latency. This definition has expected benefits for the energy consumption efficiency, as it was mentioned in the conclusion insights. Nonetheless, PARL can be configured to learn a dedicated strategy to minimize the energy consumption. This will likely negatively impact performance by, for example, attempting to avoid the energy consumption concerns of DRAM.

A new reward function based on a power model integrated into the simulator could be used as feedback to the agent. Energy consumption specifications, expressed similarly to the timing parameters, is a potential implementation. Alternatively, simulators for power consumption of the evaluated memory technologies could be interfaced with Ramulator [203]. However, in this case, a single episode of learning would have to be completed before consumed energy can be calculated because the device command history has to be exported to the power simulator for processing. This would not allow the agent to make adjustments to its decisions online, as it does with real-time latency information using the performance-oriented rewards in its current implementation.

Extensions to PARL

A real system implementation with existing page migration mechanism would be an important extension to PARL. Alternatively, a validated migration mechanism with an extended storage model in Ramulator could be used a starting point. However, interfacing main memory and storage simulators could provide greater modeling accuracy, despite the higher design efforts necessary. Another potential extension is to incorporate the method proposed in [108] and add controller-level insights for improvements in both the address space management and the device command scheduling.

Overall, the configurable, adaptive, and non-biased approach to initial data placement presented in this work lays out a suitable foundation for further research in hybrid memory management. The technology-agnostic nature of PARL allows for a fair comparison and evaluation of various memory and system configurations, by following the feedback from the system and making decisions to maximize rewards, rather than following predefined rules.

As a whole, this work has explored an alternative approach in memory management using machine learning, addressing the shortcomings and limitations of existing literature. The proposed method achieves its goals and provides foundation for future work in various directions.

Publications

E. Karimov, T. Evenblij, S.A. Chamazcoti, F. Catthoor "PARL: Page Allocation in Hybrid Main Memory using Reinforcement Learning", Journal of Systems Architecture, Volume 159, 2025. [110]

K. Akhunov, D. Biswas, **E. Karimov**, A. Sharma, H. Oh, M. Rosmeulen, J. Ryckaert, J. Myers "Framework for Augmenting Main Memory with CXL-connected Emerging Memory Alternatives", ISCAS 2025. (accepted for publication)

Appendix A

Page Fault Mechanism

Fig. 3.4 presented earlier in the thesis represents the workloads with the most page faults, such as "grep-map0" with 4,993 faults in the 64MB configuration and baseline PTT. Reducing the capacity to 16MB results in a 490% increase, or 29,569 faults. In contrast, the same capacity change results in a 26% increase (from 5,969,131 to 7,580,364 faults) using the modified PTT. Another workload ("470.lbm") had 37,998 page faults in the baseline for both the 32MB and 16MB configurations, or a 180% increase from 64MB memory with 13,422 faults. The modified PTT results in a 29% increase in page faults when reducing capacity from 64MB to 32MB, and a further 11% increase from 32MB to 16MB for a total of 10,629,249 faults.

The "grep-map0" trace uses 38,000 different pages with a total of 9.4 million accesses. Given the memory hierarchy configuration (i.e. LLC), those are filtered down to 7.7 million accesses to the main memory. Such a small difference between cache-level and memory-level accesses indicates that this workload accesses new locations each time and does not benefit from the cache locality. This conclusion can be confirmed by the visualization of the page accesses throughout the lifetime of the workload, plotted against the chronological order of appearance of each accessed page. One example has been shown already in Fig. 3.11. Further confirmation comes from the access rates of the cache access rates as seen in Table 3.1.

On the other hand, "470.lbm" uses 46,000 pages accessed 10 million times. Despite the similarity in the "volume" of accesses in both trace files, LLC benefits "470.lbm". Most pages are accessed only tens of times, indicating frequent switching between pages (Fig. A.1). This can be seen from a significantly lower number of cache accesses shown in Table 3.1.

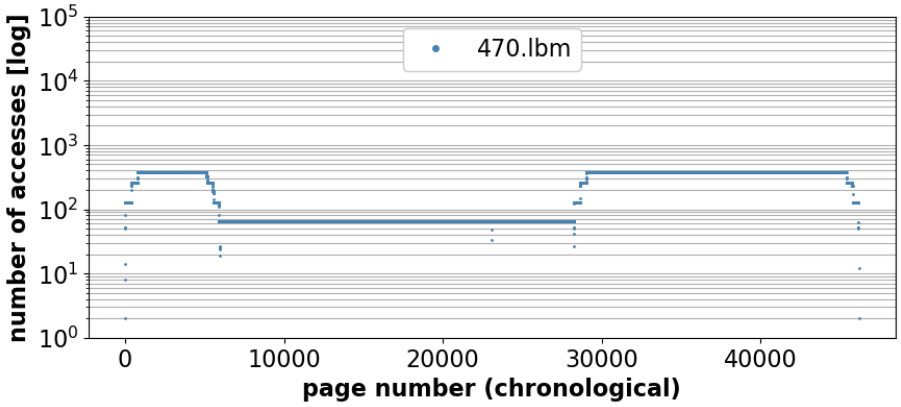


Figure A.1: Access rate of pages in the order their appearance.

Overall, the increased number of page faults is a significant contributor to associated page fault delays that results in simulation time increasing significantly, such that the largest workloads cannot be feasibly simulated repeatedly to evaluate the learning behavior of the RL agent or to observe the impact of system configuration changes.

Appendix B

Workload Classification using Page Allocation Timeline

Fig. B.4-B.7 show the timelines of workloads based on the interaction with the requested pages (at the end of this appendix section). The scale of each workload is proportional to itself, and green dots are used to represent when a new page is accessed and thus, requires a new allocation in main memory. Gray dots represent already allocated pages being accessed again. This figure does not identify the pages being allocated and reused, and only sorts them in the order of their appearance. Fig. B.4-B.6 correspond to the classification proposed in Section 4.3, and Fig. B.7 lists the workload with a combined continuous & periodic profile, that are considered as continuous in general.

As an example, in Fig. B.1 it can be seen how pages are allocated with respect to the number of requests made to those pages. This workload makes its last request to a new page at request #524,222 - while the total number of requests made in this workload is 6,447,884. In contrast, Fig. B.2 shows a workload with a continuous profile that allocates new pages throughout its lifetime and frequently, expressed in low average number between requests and low standard deviation. The average is calculated including the last request number to determine the distance between the last request to a new page and the lifetime of the workload. Higher standard deviation implies that the workload has a periodic or one-time profile.

All workloads with the average number of requests between new page allocations and the standard deviation are reported in Fig. B.3.

Overall, this is a qualitative method of classification, serving as a reference point for distinguishing the workloads. It is relevant in this work as it aligns with the focus on the initial page placement. Quantitatively, for more fine-grained profiling efforts, workloads can be analyzed for a number of various metrics. These can be cache- or main memory-level statistics, as mentioned previously in the discussion on page faults, or attributes derived from other parameters in the system, such as performance counters, MPKI, etc.

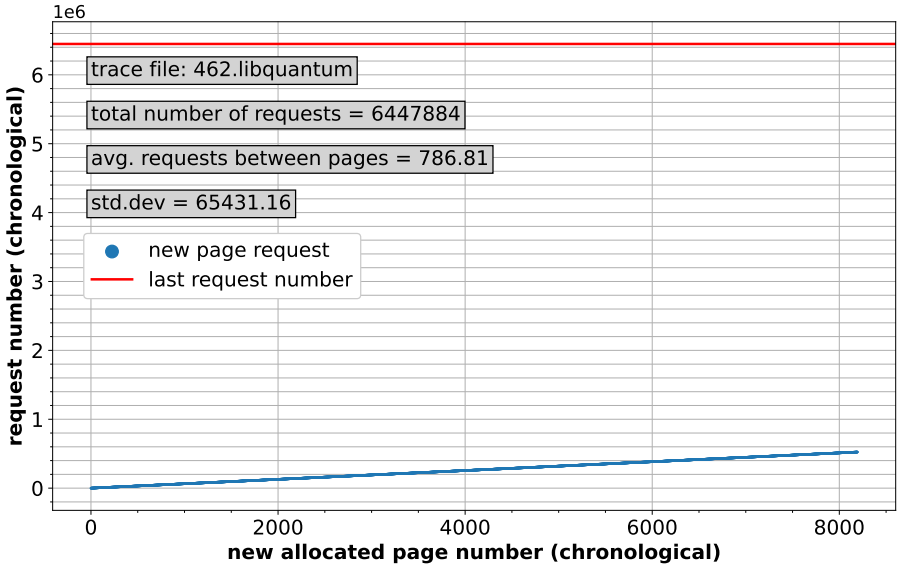


Figure B.1: Occurrence of new page allocations and related statistics

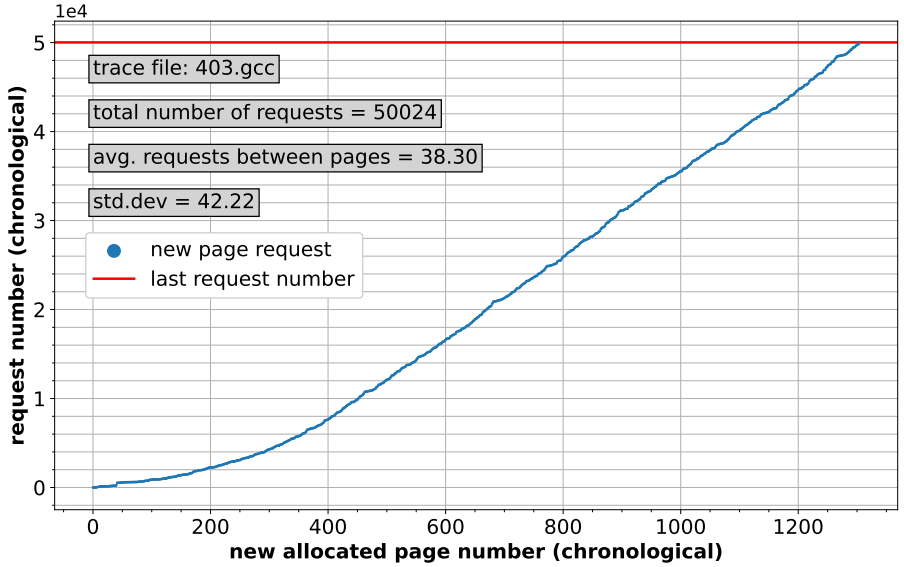


Figure B.2: Occurrence of new page allocations and related statistics

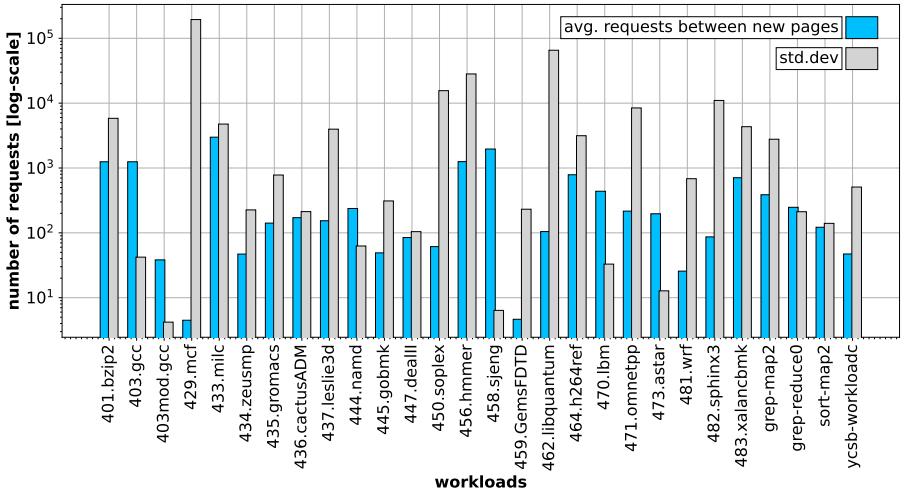


Figure B.3: Average number of requests between new page allocations for all workloads.



Figure B.4: Page occurrence timeline for workloads with a one-time profile.

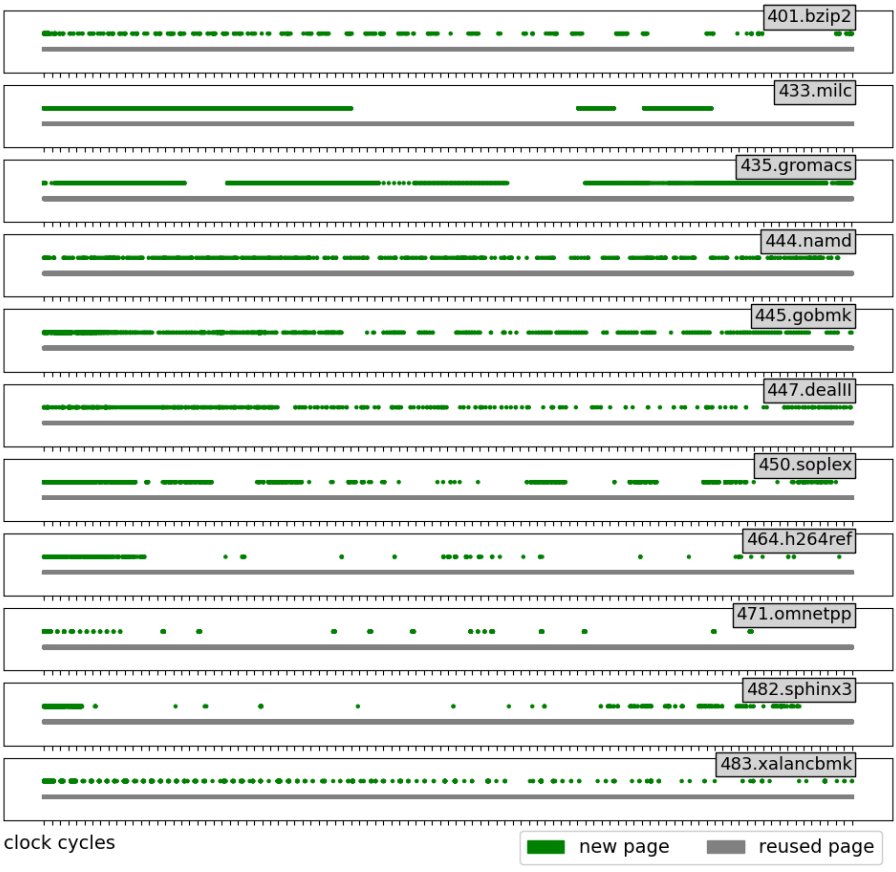


Figure B.5: Page occurrence timeline for workloads with a periodic profile.

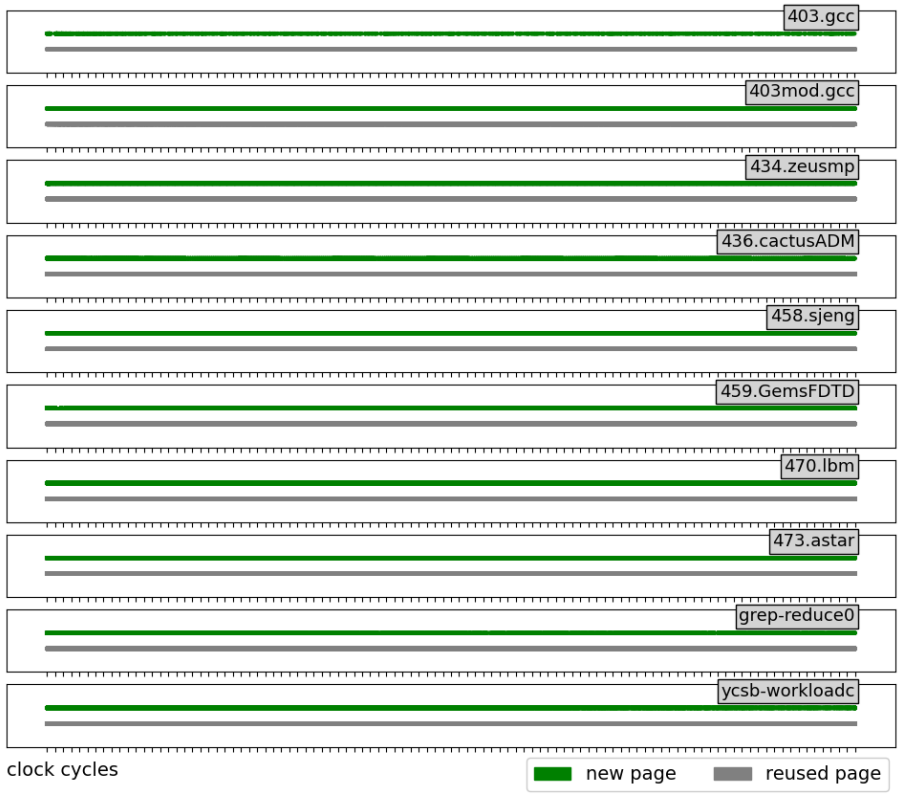


Figure B.6: Page occurrence timeline for workloads with a continuous profile.

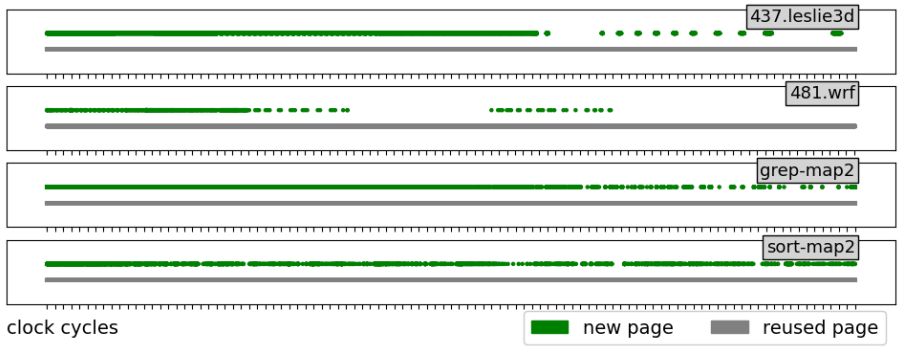


Figure B.7: Page occurrence timeline for workloads with a continuous & periodic profile.

Appendix C

Exploration of Future Work

Two preliminary experiments are introduced here, serving as early exploration of potential extensions to the presented work. First, PARL is evaluated in an environment with different workloads and shared experiences, to observe its impact on the system and the workloads. In the second half, the response of the workloads to using alternative memory technologies will be evaluated, combined with the impact of reward functions.

C.1 Shared Learning Across Workloads

For these experiments, PARL has been deployed in a configuration with a shared Q-table across several workloads instead of iteratively learning from the same workload.

The results show that it is as successful in improving runtime as in the isolated learning. Fig. C.1 shows a set of randomly chosen 12 workloads out of the entire set used in evaluations. One of these workloads is randomly chosen for every simulation episode. In this figure, the instructions per cycle (IPC) executed by the processor are reported for each workload, instead of evaluating the time in memory explicitly. IPC is a common evaluation metric, however, it also includes the computational part of the workload. It includes the change in the memory cycles, and could take values in the range of [0,4] due to the 4-wide instruction issue windows. Each workload is compared to its own benchmark IPC achieved under the same dual-channel DRAM-only main memory.

At a short lifetime of 120 episodes, it can be observed that the agent is able to

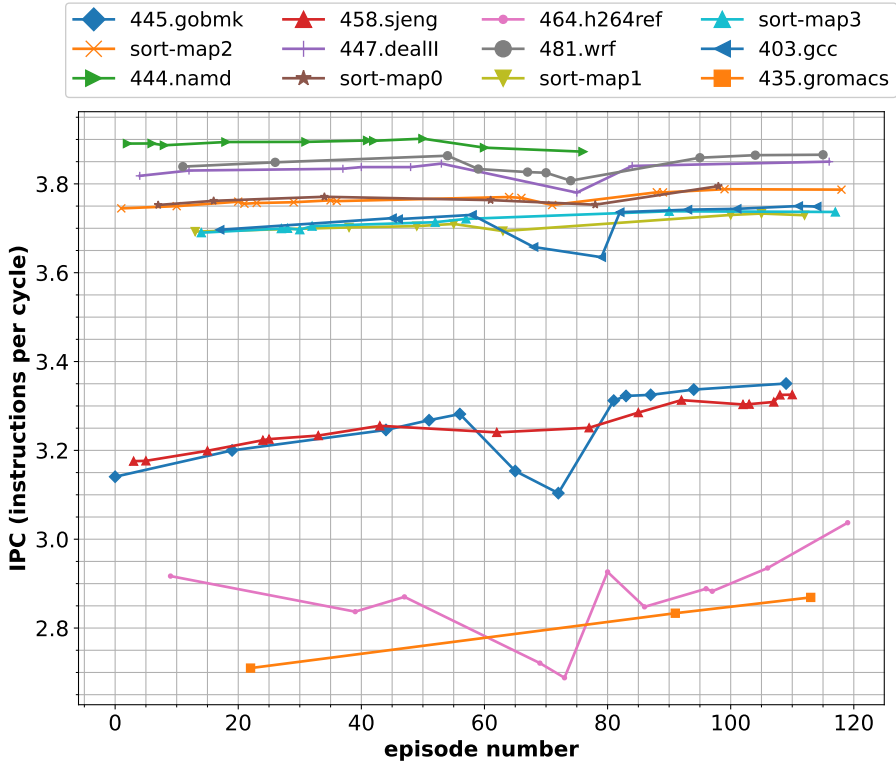


Figure C.1: PARL with a shared Q-table across 12 workloads, showing improvements in IPC achieved for every workload simulated sequentially.

learn an arbitration strategy even when the workloads are changing because of the agent observes the state of the system. These observations are sufficiently generalized and only reflect the system itself - therefore, the learned Q-values do not reflect the workloads, but how the system was used and what decision led to greater rewards and lower effective latency. Each workload shows improvements in performance, and within 10-15% of the benchmark IPC. Maximizing the performance and eliminating the fluctuations is a subject for future work on optimizing and improving this extension.

C.2 Memory Technology Exploration

In this work, the evaluation was based on using DRAM and a variant of SCM-M based on PCM. Memory configuration, including the technologies, has a significant impact on how the workload will respond and how the agent can learn within the constraints. An early timing model for IGZO-based DRAM was used here, including variations in the timing specifications of a PCM-based configuration. Technology specifications are derived from outside this work, representing early experimental results that are not completely validated and serve only as examples.

C.2.1 Improvement potential under different memory technologies and rewards

Fig. C.2 depicts three variations of the reward function that were previously discussed in Section 3.2.3. Two HMM configurations are deployed to observe the impact on the IPC. "DRAM&IGZO" is the alternative configuration, while "DRAM&PCM" is similar to the setup used in evaluation of this work.

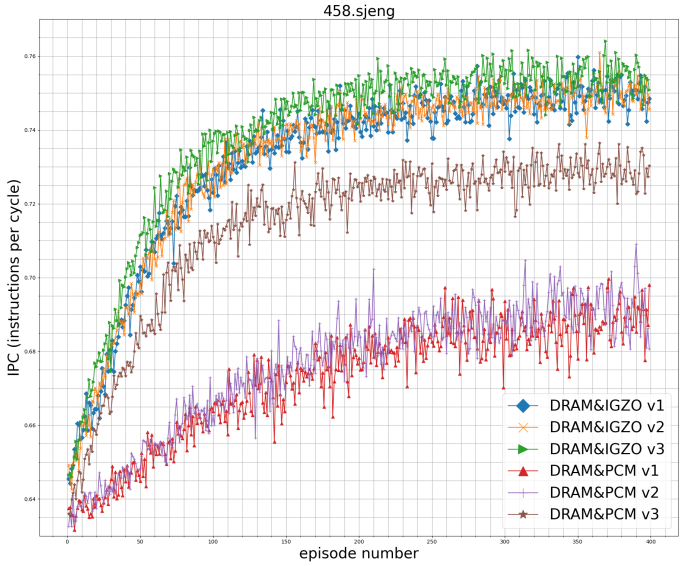
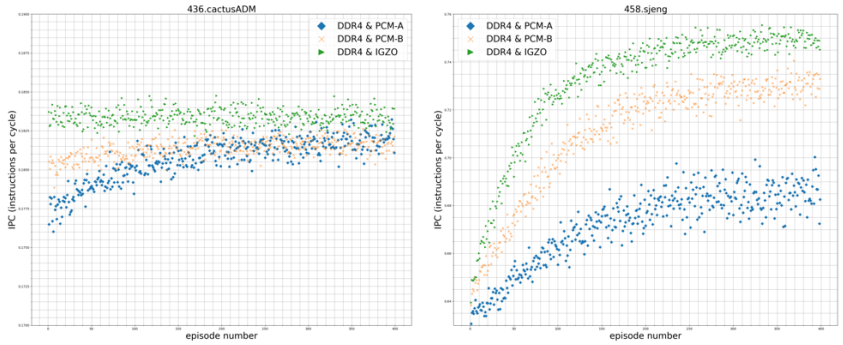


Figure C.2: IPC improvements by PARL in `458.sjeng` using different technology combinations and reward functions.

The reward function #1 scales down the reward by a factor of 10^6 in case of a page fault, including the page fault delay. Reward function #2 uses a negative value instead. Reward function #3 is an extension of the latter that also does not allow a page fault to occur unless capacity of all memory devices is exhausted. In some workloads, the third version of the reward function has a significant impact even with slower technology pairing such as "DRAM&PCM".

C.2.2 Hybrid Main Memory with Different Technologies

Changes in the characteristics of technologies will be reflected in the feedback to the agent and impact initial data placement. In addition to Fig. C.2, Fig. C.3 shows three different technology pairings. Here, "DRAM&PCM-A" and "DRAM&PCM-B" are variants of timing specifications of a PCM device. "DRAM&IGZO" are devices with similar timing characteristics.



(a) 436.cactusADM (b) 458.sjeng

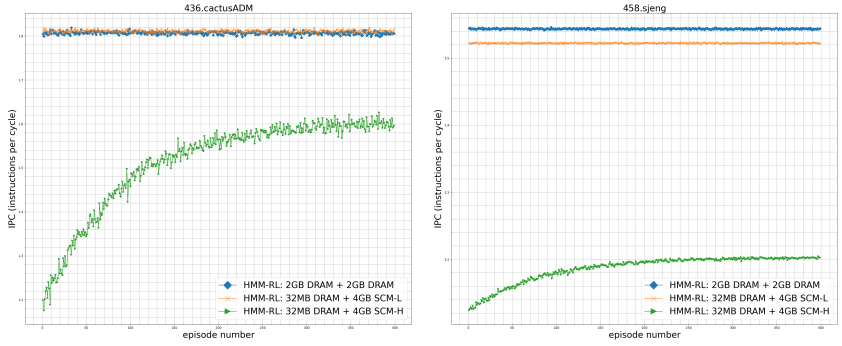
Figure C.3: IPC improvements by PARL in sample workloads using different technology combinations in HMM.

Replacing PCM with IGZO-DRAM in `436.cactusADM` shows that when the discrepancy in device specifications is eliminated, in this workload the potential of the RL agent to learn is lower, as the latency in either device is similar. Therefore, there is little difference as to where to initially place the data. For `458.sjeng`, however, the effect of IGZO-DRAM is more pronounced. This workload can benefit from lower latency devices, and PARL adapts to maximize the performance under "DRAM&PCM-B". Yet, "DRAM&IGZO" has both devices with low latency and more performance to be extracted. As it was concluded already, PARL adapts and learns to optimize its decisions based on the constraints of underlying technologies.

C.2.3 Hybrid Main Memory with Homogeneous Technology

Lastly, in addition to a hybrid system, PARL also has an effect for a homogeneous DRAM-only main memory. In multi-channel configuration, the default address management assigns parts of the PA to the device-level locations (channels, rows, columns, etc.). This is known as address mapping, and it is exclusively a device-level mechanism. For example, the least significant bit (LSB) of the address could be used as the device channel number. The LSB changes more often than the most significant bit (MSB) and ensures that requests are frequently forwarded to alternating channels for memory parallelism.

When PARL learns to choose between HMM devices, the decision it makes is similar to choosing one of the channels. Replacing an SCM-M device in HMM with a second DRAM results in effectively a dual-channel DRAM memory controlled by PARL. The agent demonstrated its ability to initial page placement here too and results it achieved were on par with the benchmark dual-channel DRAM configuration across all workloads evaluated. An example is shown in Fig. C.4. DRAM-only HMM is represented by the "DRAM+DRAM" configuration, while "SCM-L" and "SCM-H" correspond to SCM-M with timing specifications of "lower" or "higher" access latency.



(a) 436.cactusADM (b) 458.sjeng

Figure C.4: IPC improvements by PARL under DRAM-only HMM.

PARL maintains the performance of the DRAM-only HMM, since it has two devices with matching characteristics and thus, little tradeoff choosing one of them. In some cases, such as `ycsb` workloads and `436.cactusADM` (Fig. C.4a, the IPC was improved by 10% than the reference without any changes to the design of PARL. This indicates that the existing memory management mechanisms have potential to exploit improvements in address and resource management with flexible and adaptive agents.

Appendix D

Declaration of the Generative AI Use

Use of Generative AI

Parts of the code used for data analysis and visualization was developed in assistance from the automated responses of ChatGPT, to parse the generated results from the experiments, and to shape them into the desired form to generate plots used in this thesis (such as Chapters 3, 5, and 6).

The text/code/images in this thesis are my own (unless otherwise specified) and generative AI has only been used in accordance with the KU Leuven guidelines and appropriate references have been added. I have reviewed and edited the content as needed and I take full responsibility for the content of the thesis.

Bibliography

[1] J. von Neumann, "First draft of a report on the edvac", *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993. DOI: 10.1109/85.238389.

[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious", *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995, ISSN: 0163-5964. DOI: 10.1145/216585.216588. [Online]. Available: https://doi.org/10.1145/216585.216588.

[3] R. Sites, "It's the memory, stupid", *Microprocessor Report*, vol. 10, no. 10, pp. 2–3, 1996.

[4] S. A. McKee, "Reflections on the memory wall", *Proceedings of the 1st Conference on Computing Frontiers*, CF '04, p. 162, 2004. DOI: 10.1145/977091.977115. [Online]. Available: https://doi-org.kuleuven.e-bronnen.be/10.1145/977091.977115.

[5] D. Patterson and J. Hennessy, *Computer Organization and Design, Enhanced: The Hardware/Software Interface* (The Morgan Kaufmann Series in Computer Architecture and Design). Elsevier Science, 2014, ISBN: 9780128012857. [Online]. Available: https://books.google.be/books?id=G7IMAwAAQBAJ.

[6] S. A. Przybylski, *Cache and memory hierarchy design: a performance directed approach*. Morgan Kaufmann, 1990.

[7] G. Sun, *Exploring memory hierarchy design with emerging memory technologies*. Springer, 2014, vol. 267.

[8] L. Wu, L. Yuan, and J. You, "Survey of large-scale data management systems for big data applications", *Journal of computer science and technology*, vol. 30, no. 1, pp. 163–183, 2015.

[9] G. A. Gibson, J. S. Vitter, and J. Wilkes, "Strategic directions in storage i/o issues in large-scale computing", *ACM Comput. Surv.*, vol. 28, no. 4, pp. 779–793, Dec. 1996, ISSN: 0360-0300. DOI: 10.1145/242223.242300. [Online]. Available: https://doi.org/10.1145/242223.242300.

[10] O. Mutlu, "Memory scaling: A systems architecture perspective", in *2013 5th IEEE International Memory Workshop*, 2013, pp. 21–25. DOI: 10.1109/IMW.2013.6582088.

[11] R. Nair, "Evolution of memory architecture", *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1331–1345, 2015. DOI: 10.1109/JPROC.2015.2435018.

[12] A. Makarov, V. Sverdlov, and S. Selberherr, "Emerging memory technologies: Trends, challenges, and modeling methods", *Microelectronics Reliability*, vol. 52, no. 4, pp. 628–634, 2012, Advances in non-volatile memory technology, ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2011.10.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271411004859.

[13] K. Fang, L. Chen, Z. Zhang, and Z. Zhu, "Memory architecture for integrating emerging memory technologies", in *2011 International Conference on Parallel Architectures and Compilation Techniques*, 2011, pp. 403–412. DOI: 10.1109/PACT.2011.71.

[14] Skymart, *References – understanding memory*. [Online]. Available: https://dieattachfilm.com/reference-article-memory/.

[15] R. Dennard, *U.s. patent us3387286a: Field-effect transistor memory*, 1968. [Online]. Available: https://patents.google.com/patent/US3387286A/.

[16] R. Dennard, "Evolution of the mosfet dynamic ram—a personal view", *IEEE Transactions on Electron Devices*, vol. 31, no. 11, pp. 1549–1555, 1984. DOI: 10.1109/T-ED.1984.21751.

[17] P. Dylan, K. Jeff, B. Tanj, C. Wega, and A. Afzal, *The memory wall: Past, present, and future of dram*, 2024. [Online]. Available: https://semianalysis.com/2024/09/03/the-memory-wall/.

[18] K. H. Lee, *A strategic analysis of the dram industry after the year 2000*, 2013. [Online]. Available: https://dspace.mit.edu/handle/1721.1/81087.

[19] F. Ferro, *Deep learning neural networks drive demands on memory bandwidth*, 2018. [Online]. Available: https://semiengineering.com/deep-learning-neural-networks-drive-demands-on-memory-bandwidth/.

[20] E. Sperling, *Dram choices are suddenly much more complicated*, 2023. [Online]. Available: https://semiengineering.com/dram-choices-are-suddenly-much-more-complicated/.

[21] J. M. Lima, *Ai: The new frontier in the european data center market*, 2023. [Online]. Available: https://www.jsa.net/blog/ai-the-new-frontier-in-the-european-data-center-market/60434/.

[22] S. Shiratake, "Scaling and performance challenges of future dram", *2020 IEEE International Memory Workshop (IMW)*, pp. 1–3, 2020. DOI: 10.1109/IMW48823.2020.9108122.

[23] J. Yu and K. Aflatooni, "Leakage current in dram memory cell", *2006 16th Biennial University/Government/Industry Microelectronics Symposium*, pp. 191–194, 2006. DOI: 10.1109/UGIM.2006.4286380.

[24] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories", *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 363–374, 2010. DOI: 10.1109/MICRO.2010.42.

[25] S. Ghose, A. G. Yaglikçi, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal, M. O'Connor, and O. Mutlu, "What your dram power models are not telling you: Lessons from a detailed experimental study", *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 3, 2018. DOI: 10.1145/3224419. [Online]. Available: https://doi.org/10.1145/3224419.

[26] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices: Mitigating data movement bottlenecks", *SIGPLAN Not.*, vol. 53, no. 2, pp. 316–331, 2018, ISSN: 0362-1340. DOI: 10.1145/3296957.3173177. [Online]. Available: https://doi.org/10.1145/3296957.3173177.

[27] A. Eisenman, D. Gardner, I. AbdelRahman, J. Axboe, S. Dong, K. Hazelwood, C. Petersen, A. Cidon, and S. Katti, "Reducing dram footprint with nvm in facebook", *Proceedings of the 13th EuroSys Conference, EuroSys 2018*, vol. 2018-January, Apr. 2018. DOI: 10.1145/3190508.3190524.

[28] I. Bhati, M.-T. Chang, Z. Chishti, S.-L. Lu, and B. Jacob, "Dram refresh mechanisms, penalties, and trade-offs", *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 108–121, 2016. DOI: 10.1109/TC.2015.2417540.

[29] O. Mutlu, "The rowhammer problem and other issues we may face as memory becomes denser", *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 1116–1121, 2017. DOI: 10.23919/DATE.2017.7927156.

[30] R. H. Norman, *U.s. patent us3562721a: Solid state switching and memory apparatus*, 1963. [Online]. Available: https://patents.google.com/patent/US3562721A/.

[31] P. Machanick, "The case for sram main memory", *SIGARCH Comput. Archit. News*, vol. 24, no. 5, pp. 23–30, Dec. 1996, ISSN: 0163-5964. DOI: 10.1145/242694.242709. [Online]. Available: https://doi.org/10.1145/242694.242709.

[32] F. Masuoka, "Bit by bit erasable e2prom with single transistor per bit", *1981 International Electron Devices Meeting*, pp. 20–23, 1981. DOI: 10.1109/IEDM.1981.189988.

[33] S. W. Fong, C. M. Neumann, and H.-S. P. Wong, "Phase-change memory—towards a storage-class memory", *IEEE Transactions on Electron Devices*, vol. 64, no. 11, pp. 4374–4385, 2017. DOI: 10.1109/TED.2017.2746342.

[34] W.-W. Z. Sheng Teng Hsu, *U.s. patent us6531371b2: Electrically programmable resistance cross point memory*, 2001. [Online]. Available: https://patents.google.com/patent/US6531371/.

[35] J. Heidecker, *Mram technology status*, version V2, 2013. DOI: 2014/43135. [Online]. Available: https://hdl.handle.net/2014/43135.

[36] H. Bahn and K. Cho, "Implications of nvm based storage on memory subsystem management", *Applied Sciences (Switzerland)*, vol. 10, 3 Feb. 2020, ISSN: 20763417. DOI: 10.3390/app10030999.

[37] B. Pourshirazi, M. V. Beigi, Z. Zhu, and G. Memik, "Wall: A writeback-aware llc management for pcm-based main memory systems", *2018 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 449–454, 2018. DOI: 10.23919/DATE.2018.8342051.

[38] S. Kargar and F. Nawab, "Challenges and future directions for energy, latency, and lifetime improvements in nvms", *Distrib. Parallel Databases*, vol. 41, no. 3, pp. 163–189, Sep. 2022, ISSN: 0926-8782. DOI: 10.1007/s10619-022-07421-x. [Online]. Available: https://doi.org/10.1007/s10619-022-07421-x.

[39] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of candidate device technologies for storage-class memory", *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 449–464, 2008. DOI: 10.1147/rd.524.0449.

[40] M. Gupta, M. Perumkunnil, A. Fantini, S. A. Chamazcoti, W. Kim, M. Bardon, G. Kar, and A. Furnémont, "Design exploration of igzo diode based vcma array design for storage class memory applications", *ESSDERC 2022 - IEEE 52nd European Solid-State Device Research Conference (ESSDERC)*, pp. 241–244, 2022. DOI: 10.1109/ESSDERC55479.2022.9947187.

[41] A. Belmonte, H. Oh, N. Rassoul, G. Donadio, J. Mitard, H. Dekkers, R. Delhougne, S. Subhechha, A. Chasin, M. J. van Setten, L. Kljucar, M. Mao, H. Puliyalil, M. Pak, L. Teugels, D. Tsvetanova, K. Banerjee, L. Souriau, Z. Tokei, L. Goux, and G. S. Kar, "Capacitor-less, long-retention (>400s) dram cell paving the way towards low-power and high-density monolithic 3d dram", *2020 IEEE International Electron Devices Meeting (IEDM)*, pp. 28.2.1–28.2.4, 2020. DOI: 10.1109/IEDM13553.2020.9371900.

[42] M. K. Gupta, M. Perumkunnil, D. Biswas, S. A. Chamazcoti, G. S. Kar, A. Furnémont, and J. Ryckaert, "Design technology co-optimization of 1d-1vcma to improve read performance for scm applications", *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2023. DOI: 10.1109/ISCAS46773.2023.10181335.

[43] W. Kang, L. Chang, Y. Zhang, and W. Zhao, "Voltage-controlled mram for working memory: Perspectives and challenges", *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 542–547, 2017. DOI: 10.23919/DATE.2017.7927047.

[44] H. Yang, X. Hao, Z. Wang, R. Malmhall, H. Gan, K. Satoh, J. Zhang, D. H. Jung, X. Wang, Y. Zhou, B. K. Yen, and Y. Huai, "Threshold switching selector and 1s1r integration development for 3d cross-point stt-mram", *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 38.1.1–38.1.4, 2017. DOI: 10.1109/IEDM.2017.8268513.

[45] M. Gupta, M. Perumkunnil, K. Garello, S. Rao, F. Yasin, G. Kar, and A. Furnémont, "High-density sot-mram technology and design specifications for the embedded domain at 5nm node", *2020 IEEE International Electron Devices Meeting (IEDM)*, pp. 24.5.1–24.5.4, 2020. DOI: 10.1109/IEDM13553.2020.9372068.

[46] R. Carpenter, W. Kim, K. Sankaran, N. Ao, M. Ben Chroud, A. Kumar, A. Trovato, G. Pourtois, S. Couet, and G. S. Kar, "Demonstration of a free-layer developed with atomistic simulations enabling beol compatible vcma-mram with a coefficient $\geq$ 100fj/vm", *2021 IEEE International Electron Devices Meeting (IEDM)*, pp. 17.6.1–17.6.4, 2021. DOI: 10.1109/IEDM19574.2021.9720579.

[47] Y. Huai, "Spin-transfer torque mram (stt-mram): Challenges and prospects", *AAPPS bulletin*, vol. 18, no. 6, pp. 33–40, 2008.

[48] H. Siegmann, "Spin polarized electrons", *Europhysics News*, vol. 14, no. 2, pp. 9–12, 1983.

[49] K. Garello, F. Yasin, S. Couet, L. Souriau, J. Swerts, S. Rao, S. Van Beek, W. Kim, E. Liu, S. Kundu, D. Tsvetanova, K. Croes, N. Jossart, E. Grimaldi, M. Baumgartner, D. Crotti, A. Fumémont, P. Gambardella, and G. Kar, "Sot-mram 300mm integration for low power and ultrafast embedded memories", in *2018 IEEE Symposium on VLSI Circuits*, 2018, pp. 81–82. DOI: 10.1109/VLSIC.2018.8502269.

[50] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth, "Traffic management: A holistic approach to memory placement on numa systems", *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pp. 381–394, 2013. DOI: 10.1145/2451116.2451157. [Online]. Available: https://doi.org/10.1145/2451116.2451157.

[51] C. Lameter, "Numa (non-uniform memory access): An overview: Numa becomes more common because memory controllers get close to execution units on microprocessors.", *Queue*, vol. 11, no. 7, pp. 40–51, Jul. 2013, ISSN: 1542-7730. DOI: 10.1145/2508834.2513149. [Online]. Available: https://doi.org/10.1145/2508834.2513149.

[52] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies", *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pp. 34–45, 2009. DOI: 10.1145/1555754.1555761. [Online]. Available: https://doi.org/10.1145/1555754.1555761.

[53] S. M. A. Iqbal and Asaduzzaman, "Cache-mab: A reinforcement learning-based hybrid caching scheme in named data networks", *Future Gener. Comput. Syst.*, vol. 147, no. C, pp. 163–178, Oct. 2023, ISSN: 0167-739X. DOI: 10.1016/j.future.2023.04.032. [Online]. Available: https://doi.org/10.1016/j.future.2023.04.032.

[54] D. Huang, A. Pahlevan, L. Costero, M. Zapater, and D. Atienza, "Reinforcement learning-based joint reliability and performance optimization for hybrid-cache computing servers", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5596–5609, 2022. DOI: 10.1109/TCAD.2022.3158832.

[55] G. Singh, R. Nadig, J. Park, R. Bera, N. Hajinazar, D. Novo, J. Gómez-Luna, S. Stuijk, H. Corporaal, and O. Mutlu, "Sibyl: Adaptive and extensible data placement in hybrid storage systems using online reinforcement learning", *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, pp. 320–336, 2022. DOI:

10.1145/3470496.3527442. [Online]. Available: https://doi.org/10.1145/3470496.3527442.

[56] K. Kavi, S. Pianelli, G. Pisano, G. Regina, and M. Ignatowski, "Memory organizations for 3d-drams and pcms in processor memory hierarchy", *Journal of Systems Architecture*, vol. 61, pp. 539–552, 10 Nov. 2015, ISSN: 13837621. DOI: 10.1016/j.sysarc.2015.07.009.

[57] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-based hybrid memory management", *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 152–165, 2017. DOI: 10.1109/CLUSTER.2017.130.

[58] T. D. Doudali, S. Blagodurov, A. Vishnu, S. Gurumurthi, and A. Gavrilovska, "Kleio: A hybrid memory page scheduler with machine intelligence", *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, pp. 37–48, 2019. DOI: 10.1145/3307681.3325398. [Online]. Available: https://doi.org/10.1145/3307681.3325398.

[59] M. Katsaragakis, L. Papadopoulos, C. Baloukas, and D. Soudris, "Memory management methodology for application data structure refinement and placement on heterogeneous dram/nvm systems", *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 748–753, Mar. 2022. DOI: 10.23919/DATE54114.2022.9774666. [Online]. Available: https://ieeexplore.ieee.org/document/9774666/.

[60] M. Lee, D. H. Kang, J. Kim, and Y. I. Eom, "M-clock: Migration-optimized page replacement algorithm for hybrid dram and pcm memory architecture", *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pp. 2001–2006, 2015. DOI: 10.1145/2695664.2695675. [Online]. Available: https://doi.org/10.1145/2695664.2695675.

[61] R. Salkhordeh and H. Asadi, "An operating system level data migration scheme in hybrid dram-nvm memory architecture", *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 936–941, 2016. DOI: 10.3850/9783981537079_0605. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7459442.

[62] Z. Yan, D. Lustig, D. Nellans, and A. Bhattacharjee, "Nimble page management for tiered memory systems", *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, pp. 331–345, Apr. 2019. DOI: 10.1145/3297858.3304024.

[63] S. A. Chamazcoti, M. Gupta, H. Oh, T. Evenblij, F. Catthoor, M. P. Komalan, G. S. Kar, and A. Furnemont, "Exploring pareto-optimal hybrid main memory configurations using different emerging memories", *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–, 2022. DOI: 10.1109/TCSI.2022.3222573.

[64] H. Yoon, J. Meza, R. Ausavarungnirun, R. A. Harding, and O. Mutlu, "Row buffer locality aware caching policies for hybrid memories", in *2012 IEEE 30th International Conference on Computer Design (ICCD)*, 2012, pp. 337–344. DOI: 10.1109/ICCD.2012.6378661.

[65] Y. Chi, H. Liu, G. Peng, X. Liao, and H. Jin, "Transformer: An os-supported reconfigurable hybrid memory architecture", *Applied Sciences*, vol. 12, no. 24, 2022, ISSN: 2076-3417. DOI: 10.3390/app122412995. [Online]. Available: https://www.mdpi.com/2076-3417/12/24/12995.

[66] J.-G. Kim, S.-D. Kim, and S.-K. Yoon, "Q-selector-based prefetching method for dram/nvm hybrid main memory system", *Electronics*, vol. 9, no. 12, 2020, ISSN: 2079-9292. DOI: 10.3390/electronics9122158. [Online]. Available: https://www.mdpi.com/2079-9292/9/12/2158.

[67] M. Hildebrand, J. T. Angeles, J. Lowe-Power, and V. Akella, "A case against hardware managed dram caches for nvram based systems", *Proceedings - 2021 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2021*, pp. 194–204, Mar. 2021. DOI: 10.1109/ISPASS51385.2021.00036.

[68] Intel, *Intel optane persistent memory start up guide*, 2020. [Online]. Available: https://cdrdv2-public.intel.com/841964/Intel_Optane_Persistent_Memory_Start_Up_Guide.pdf.

[69] T. Kilburn, R. B. Payne, and D. J. Howarth, "The atlas supervisor", *Proceedings of the December 12-14, 1961, Eastern Joint Computer Conference: Computers - Key to Total Systems Control*, AFIPS '61 (Eastern), pp. 279–294, 1961. DOI: 10.1145/1460764.1460786. [Online]. Available: https://doi.org/10.1145/1460764.1460786.

[70] *Intel 64 and ia-32 architectures software developer's manual combined volumes: 1, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d, and 4*, Intel Corporation, Jun. 2024. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/825743?fileName=325462-sdm-vol-1-2abcd-3abcd-4.pdf.

[71] J. Bartlett, *Inside memory management*, IBM, Nov. 2004. [Online]. Available: https://developer.ibm.com/tutorials/l-memory/.

[72] F. Catthoor, S. Wuytack, G. de Greef, F. Banica, L. Nachtergaele, and A. Vandecappelle, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Springer US, 2013, ISBN: 9781475728491. [Online]. Available: https://books.google.be/books?id=y7HaBwAAQBAJ.

[73] E. Abrossimov, M. Rozier, and M. Shapiro, "Generic virtual memory management for operating system kernels", *SIGOPS Oper. Syst. Rev.*, vol. 23, no. 5, pp. 123–136, Nov. 1989, ISSN: 0163-5980. DOI: 10.1145/74851.74863. [Online]. Available: https://doi.org/10.1145/74851.74863.

[74] L. Liu, Y. Li, C. Ding, H. Yang, and C. Wu, "Rethinking memory management in modern operating system: Horizontal, vertical or random?", *IEEE Transactions on Computers*, vol. 65, no. 6, pp. 1921–1935, 2016. DOI: 10.1109/TC.2015.2462813.

[75] M. Tofte and J.-P. Talpin, "Region-based memory management", *Information and Computation*, vol. 132, no. 2, pp. 109–176, 1997, ISSN: 0890-5401. DOI: https://doi.org/10.1006/inco.1996.2613. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540196926139.

[76] G. Attardi, T. Flagella, and P. Iglio, "A customisable memory management framework for c++", *Software: Practice and Experience*, vol. 28, no. 11, pp. 1143–1184, 1998.

[77] S. Kannan, A. Gavrilovska, V. Gupta, and K. Schwan, "Heteroos: Os design for heterogeneous memory management in datacenter", *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pp. 521–534, 2017. DOI: 10.1145/3079856.3080245. [Online]. Available: https://doi.org/10.1145/3079856.3080245.

[78] L. Liu, H. Yang, Y. Li, M. Xie, L. Li, and C. Wu, "Memos: A full hierarchy hybrid memory management framework", *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pp. 368–371, 2016. DOI: 10.1109/ICCD.2016.7753305.

[79] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for nvm+dram hybrid main memory", *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, p. 14, 2009.

[80] W. Tian, J. Li, Y. Zhao, C. J. Xue, M. Li, and E. Chen, "Optimal task allocation on non-volatile memory based hybrid main memory", *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*, pp. 1–6, Nov. 2011. DOI: 10.1145/2103380.2103382.

[Online]. Available: https://dl.acm.org/doi/10.1145/2103380.2103382.

[81] G. Park, S. Rho, J. S. Kim, and D. Nam, "Towards optimal scheduling policy for heterogeneous memory architecture in many-core system", *Cluster Computing*, vol. 22, pp. 121–133, 1 Mar. 2019, ISSN: 15737543. DOI: 10.1007/s10586-018-2825-4.

[82] M. Prabhu, H. N. Upadhyay, and R. Vijay, "Hyper switching memory utilization on hybrid main memory for improved task execution and reduced power consumption", *Microprocessors and Microsystems*, vol. 72, Feb. 2020, ISSN: 01419331. DOI: 10.1016/j.micpro.2019.102891.

[83] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Software-managed energy-efficient hybrid dram/nvm main memory", *Proceedings of the 12th ACM International Conference on Computing Frontiers, CF 2015*, May 2015. DOI: 10.1145/2742854.2742886.

[84] G. Nakagawa and S. Oikawa, "Nvm/dram hybrid memory management with language runtime support via mrw queue", *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 1–6, Jun. 2015. DOI: 10.1109/SNPD.2015.7176226.

[85] N. Agarwal, D. Nellans, M. Stephenson, M. O'Connor, and S. W. Keckler, "Page placement strategies for gpus within heterogeneous memory systems", *ACM SIGPLAN Notices*, vol. 50, pp. 607–618, 4 Apr. 2015, ISSN: 15232867. DOI: 10.1145/2694344.2694381.

[86] S. R. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, and K. Schwan, "Data tiering in heterogeneous memory systems", *Proceedings of the 11th European Conference on Computer Systems, EuroSys 2016*, Apr. 2016. DOI: 10.1145/2901318.2901344.

[87] K. Wu, Y. Huang, and D. Li, "Unimem: Runtime data managementon non-volatile memory-based heterogeneous main memory", *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, 2017. DOI: 10.1145/3126908.3126923. [Online]. Available: https://doi.org/10.1145/3126908.3126923.

[88] C. Wang, J. Zigman, F. Lv, H. Cui, H. Volos, X. Feng, T. Cao, O. Mutlu, and G. H. Xu, "Panthera: Holistic memory management for big data processing over hybrid memories", *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 347–362, Jun. 2019. DOI: 10.1145/3314221.3314650.

[89] F. Wen, M. Qin, P. Gratz, and N. Reddy, "Software hint-driven data management for hybrid memory in mobile systems", *ACM Transactions on Embedded Computing Systems*, vol. 21, 1 Jan. 2022, ISSN: 15583465. DOI: 10.1145/3494536.

[90] S. Jiang, F. Chen, and X. Zhang, "Clock-pro: An effective improvement of the clock replacement", *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, p. 35, 2005.

[91] G. Dhiman, R. Ayoub, and T. Rosing, "Pdram: A hybrid pram and dram main memory system", *Proceedings of the 46th Annual Design Automation Conference*, pp. 664–469, Jul. 2009. DOI: 10.1145/1629911.1630086. [Online]. Available: https://dl.acm.org/doi/10.1145/1629911.1630086.

[92] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Simple but effective heterogeneous main memory with on-chip memory controller support", *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, Nov. 2010. DOI: 10.1109/SC.2010.50.

[93] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems", *Proceedings of the international conference on Supercomputing*, pp. 85–95, May 2011. DOI: 10.1145/1995896.1995911.

[94] S. Lee, H. Bahn, and S. H. Noh, "Clock-dwf: A write-history-aware page replacement algorithm for hybrid pcm and dram memory architectures", *IEEE Transactions on Computers*, vol. 63, no. 9, pp. 2187–2200, 2014. DOI: 10.1109/TC.2013.98.

[95] B. Gerofit, A. Shimada, A. Hori, T. Masamichi, and Y. Ishikawa, "Cmcp: A novel page replacement policy for system level hierarchical memory management on many-cores", *HPDC 2014 - Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, pp. 73–84, 2014. DOI: 10.1145/2600212.2600231.

[96] N. Agarwal and T. F. Wenisch, "Thermostat: Application-transparent page management for two-tiered main memory", *ACM SIGPLAN Notices*, vol. 52, pp. 631–644, 4 Apr. 2017, ISSN: 15232867. DOI: 10.1145/3037697.3037706.

[97] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-based hybrid memory management", *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 152–165, Sep. 2017. DOI: 10.1109/CLUSTER.2017.130.

[98] S. Kim, S. H. Hwang, and J. W. Kwak, "Adaptive-classification clock: Page replacement policy based on read/write access pattern for hybrid dram and pcm main memory", *Microprocessors and Microsystems*, vol. 57, pp. 65–75, Mar. 2018, ISSN: 01419331. DOI: 10.1016/j.micpro.2018.01.003.

[99] J. Zhan, Y. Zhang, W. Jiang, J. Yang, L. Li, and Y. Li, "Energy-aware page replacement and consistency guarantee for hybrid nvm–dram memory systems", *Journal of Systems Architecture*, vol. 89, pp. 60–72, Sep. 2018, ISSN: 13837621. DOI: 10.1016/j.sysarc.2018.07.004.

[100] L. D. S. Oliveira, A. C. Yamin, R. H. S. Reiser, and H. S. Santos, "F-hybridmem: A consensual analysis via fuzzy consensus measures and penalty functions", *IEEE International Conference on Fuzzy Systems*, vol. 2022-July, 2022, ISSN: 10987584. DOI: 10.1109/FUZZ-IEEE55066.2022.9882879.

[101] L. D. S. Oliveira, R. C. D. Moura, G. B. Schneider, M. L. Pilla, A. C. Yamin, R. H. S. Reiser, and B. R. C. Bedregal, "Toward a fuzzy logic-based consensus analysis in hybrid memory management", *IEEE International Conference on Fuzzy Systems*, vol. 2021-July, Jul. 2021, ISSN: 10987584. DOI: 10.1109/FUZZ45933.2021.9494558.

[102] L. C. P. M and S. R. R, "Optimal page allocation of hybrid main memory using page caching algorithm", *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 8, pp. 522–526, 3 Sep. 2019, ISSN: 22773878. DOI: 10.35940/ijrte.A3192.098319. [Online]. Available: https://www.ijrte.org/portfolio-item/A3192058119/.

[103] L. Zhang, R. Karimi, I. Ahmad, and Y. Vigfusson, "Optimal data placement for heterogeneous cache, memory, and storage systems", *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, pp. 1–27, 1 May 2020. DOI: 10.1145/3379472.

[104] X. Wang, H. Liu, X. Liao, and H. Jin, "Superpage-friendly page table design for hybrid memory systems", *Communications in Computer and Information Science*, vol. 1257 CCIS, pp. 623–641, 2020, ISSN: 18650937. DOI: 10.1007/978-981-15-7981-3_46.

[105] J. Kim, W. Choe, and J. Ahn, "Exploring the design space of page management for multi-tiered memory systems", *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 715–728, Jul. 2021. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/kim-jonghyeon.

[106] 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, ISSN: 19374151. DOI: 10.1109/TCAD.2022.3231836.

[107] M. Katsaragakis, K. Stavrakakis, D. Masouros, L. Papadopoulos, and D. Soudris, "Adjacent LSTM-Based Page Scheduling for Hybrid DRAM/NVM Memory Systems", *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)*, Open Access Series in Informatics (OASIcs), vol. 107, J. Bispo, H.-P. Charles, S. Cherubin, and G. Massari, Eds., 7:1–7:12, 2023, ISSN: 2190-6807. DOI: 10.4230/OASIcs.PARMA-DITAM.2023.7. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.PARMA-DITAM.2023.7.

[108] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach", *2008 International Symposium on Computer Architecture*, pp. 39–50, 2008. DOI: 10.1109/ISCA.2008.21.

[109] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, "Fair queuing memory systems", in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, 2006, pp. 208–222. DOI: 10.1109/MICRO.2006.24.

[110] E. Karimov, T. Evenblij, S. A. Chamazcoti, and F. Catthoor, "Parl: Page allocation in hybrid main memory using reinforcement learning", *Journal of Systems Architecture*, vol. 159, p. 103 310, 2025, ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2024.103310. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762124002479.

[111] B. F. Skinner, *Contingencies of reinforcement : a theoretical analysis.* (The century psychology series). New York (N.Y.): Appleton-Century-Crofts, 1969.

[112] B. F. Skinner, *The technology of teaching* (The century psychology series). New York (N.Y.): Appleton-Century-Crofts, 1968.

[113] F. M. Albar and A. J. Jetter, "Heuristics in decision making", in *PICMET '09 - 2009 Portland International Conference on Management of Engineering & Technology*, 2009, pp. 578–584. DOI: 10.1109/PICMET.2009.5262123.

[114] M. Mohri, *Foundations of machine learning*, 2018.

[115] G. Gigerenzer and W. Gaissmaier, "Heuristic decision making", *Annual review of psychology*, vol. 62, no. 2011, pp. 451–482, 2011.

[116] D. Pilat and S. Krastev, *Why do we take mental shortcuts? heuristics, explained.* 2021. [Online]. Available: `https://thedecisionlab.com/biases/heuristics`.

[117] H. Müller-Merbach, "Heuristics and their design: A survey", *European Journal of Operational Research*, vol. 8, no. 1, pp. 1–23, 1981.

[118] M. W. Berry, A. Mohamed, and B. W. Yap, *Supervised and unsupervised learning for data science*. Springer, 2019.

[119] R. S. Sutton, "Reinforcement learning: An introduction", *A Bradford Book*, 2018.

[120] J. Hestness, N. Ardalani, and G. Diamos, "Beyond human-level accuracy: Computational challenges in deep learning", in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '19, Washington, District of Columbia: Association for Computing Machinery, 2019, pp. 1–14, ISBN: 9781450362252. DOI: `10.1145/3293883.3295710`. [Online]. Available: `https://doi-org.kuleuven.e-bronnen.be/10.1145/3293883.3295710`.

[121] G. A. Lewis, S. Bellomo, and I. Ozkaya, "Characterizing and detecting mismatch in machine-learning-enabled systems", in *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*, IEEE, 2021, pp. 133–140.

[122] A. D'Amour, K. Heller, D. Moldovan, B. Adlam, B. Alipanahi, A. Beutel, C. Chen, J. Deaton, J. Eisenstein, M. D. Hoffman, *et al.*, "Underspecification presents challenges for credibility in modern machine learning", *Journal of Machine Learning Research*, vol. 23, no. 226, pp. 1–61, 2022.

[123] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, R. Tibshirani, and J. Friedman, "Overview of supervised learning", *The elements of statistical learning: Data mining, inference, and prediction*, pp. 9–41, 2009.

[124] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning", *The elements of statistical learning: Data mining, inference, and prediction*, pp. 485–585, 2009.

[125] C. M. Eckhardt, S. J. Madjarova, R. J. Williams, M. Ollivier, J. Karlsson, A. Pareek, and B. U. Nwachukwu, "Unsupervised machine learning methods and emerging applications in healthcare", *Knee Surgery, Sports Traumatology, Arthroscopy*, vol. 31, no. 2, pp. 376–381, 2023.

[126] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised machine learning for networking: Techniques, applications and research challenges", *IEEE access*, vol. 7, pp. 65 579–65 615, 2019.

[127] J. Wang and F. Biljecki, "Unsupervised machine learning in urban studies: A systematic review of applications", *Cities*, vol. 129, p. 103 925, 2022.

[128] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey", *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[129] M. van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes", in *Reinforcement Learning: State-of-the-Art*, M. Wiering and M. van Otterlo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 3–42, ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_1. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3%5C_1.

[130] V. Gullapalli, *Reinforcement learning and its application to control*. University of Massachusetts Amherst, 1992.

[131] C. Ribeiro, "Reinforcement learning agents", *Artificial intelligence review*, vol. 17, pp. 223–250, 2002.

[132] C. J. C. H. Watkins and P. Dayan, "Q-learning", *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992698. [Online]. Available: https://doi.org/10.1007/BF00992698.

[133] G. Tesauro, "Temporal difference learning and td-gammon", *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995, ISSN: 0001-0782. DOI: 10.1145/203330.203343. [Online]. Available: https://doi.org/10.1145/203330.203343.

[134] E. Even-Dar and Y. Mansour, "Learning rates for q-learning", in *Computational Learning Theory*, D. Helmbold and B. Williamson, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 589–604, ISBN: 978-3-540-44581-4.

[135] F. M. Learning, *The critical role of discount factor in reinforcement learning*. [Online]. Available: https://futuremachinelearning.org/the-critical-role-of-discount-factor-in-reinforcement-learning/.

[136] R. Amit, R. Meir, and K. Ciosek, "Discount factor as a regularizer in reinforcement learning", in *International conference on machine learning*, PMLR, 2020, pp. 269–278.

[137] S. Mahadevan, "Average reward reinforcement learning: Foundations, algorithms, and empirical results", *Mach. Learn.*, vol. 22, no. 1–3, pp. 159–195, Jan. 1996, ISSN: 0885-6125. DOI: 10.1007/BF00114727. [Online]. Available: https://doi-org.kuleuven.e-bronnen.be/10.1007/BF00114727.

[138] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough", *Artificial Intelligence*, vol. 299, p. 103 535, 2021, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2021.103535.

[139] O. Berger-Tal, J. Nathan, E. Meron, and D. Saltz, "The exploration-exploitation dilemma: A multidisciplinary framework", *PloS one*, vol. 9, no. 4, e95693, 2014.

[140] J. G. March, "Exploration and exploitation in organizational learning", *Organization science*, vol. 2, no. 1, pp. 71–87, 1991.

[141] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator", *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011, ISSN: 0163-5964. DOI: 10.1145/2024716.2024718. [Online]. Available: https://doi.org/10.1145/2024716.2024718.

[142] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator", *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016. DOI: 10.1109/LCA.2015.2414456.

[143] L. Steiner, M. Jung, F. S. Prado, K. Bykov, and N. Wehn, "Dramsys4. 0: An open-source simulation framework for in-depth dram analyses", *International Journal of Parallel Programming*, vol. 50, no. 2, pp. 217–242, 2022.

[144] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "Dramsim: A memory system simulator", *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 100–107, Nov. 2005, ISSN: 0163-5964. DOI: 10.1145/1105734.1105748. [Online]. Available: https://doi.org/10.1145/1105734.1105748.

[145] 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*, 2012, pp. 392–397. DOI: 10.1109/ISVLSI.2012.82.

[146] J. E. Graver, "On the foundations of linear and integer linear programming i", *Mathematical Programming*, vol. 9, no. 1, pp. 207–226, 1975.

[147] S. Khuri and H.-C. Hsu, "Visualizing the cpu scheduler and page replacement algorithms", *SIGCSE Bull.*, vol. 31, no. 1, pp. 227–231, Mar. 1999, ISSN: 0097-8418. DOI: 10.1145/384266.299764. [Online]. Available: https://doi.org/10.1145/384266.299764.

[148] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures", *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, 2019. DOI: 10.1162/neco_a_01199.

[149] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017. DOI: 10.1109/TNNLS.2016.2582924.

[150] L. Medsker and L. C. Jain, *Recurrent neural networks: design and applications*. CRC press, 1999.

[151] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching", *SIGOPS Oper. Syst. Rev.*, vol. 29, no. 5, pp. 79–95, Dec. 1995, ISSN: 0163-5980. DOI: 10.1145/224057.224064. [Online]. Available: https://doi.org/10.1145/224057.224064.

[152] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "Dramdig: A knowledge-assisted tool to uncover dram address mapping", *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020. DOI: 10.1109/DAC18072.2020.9218599.

[153] H. Nam, S. Baek, M. Wi, M. J. Kim, J. Park, C. Song, N. S. Kim, and J. H. Ahn, "Dramscope: Uncovering dram microarchitecture and characteristics by issuing memory commands", in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1097–1111. DOI: 10.1109/ISCA59077.2024.00083.

[154] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking dram design and organization for energy-constrained multi-cores", in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10, Saint-Malo, France: Association for Computing Machinery, 2010, pp. 175–186, ISBN: 9781450300537. DOI: 10.1145/1815961.1815983. [Online]. Available: https://doi.org/10.1145/1815961.1815983.

[155] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (salp) in dram", *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 368–379, 2012. DOI: 10.1109/ISCA.2012.6237032.

[156] M. Jung, D. M. Mathew, C. Weis, N. Wehn, I. Heinrich, M. V. Natale, and S. O. Krumke, "Congen: An application specific dram memory controller generator", *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, pp. 257–267, 2016. DOI: 10.1145/2989081.2989131. [Online]. Available: https://doi.org/10.1145/2989081.2989131.

[157] B. Akin, J. C. Hoe, and F. Franchetti, "Hamlet: Hardware accelerated memory layout transform within 3d-stacked dram", in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, 2014, pp. 1–6.

[158] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems", *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 416–427, 2012.

[159] S. Bayliss and G. A. Constantinides, "Application specific memory access, reuse and reordering for sdram", in *International Symposium on Applied Reconfigurable Computing*, Springer, 2011, pp. 41–52.

[160] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns", in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Jul. 2018, pp. 1919–1928. [Online]. Available: https://proceedings.mlr.press/v80/hashemi18a.html.

[161] S. Schelter, F. Biessmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas, "On challenges in machine learning model management", *IEEE Data Engineering Bulletin*, 2015. [Online]. Available: https://www.amazon.science/publications/on-challenges-in-machine-learning-model-management.

[162] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies", *ACM Comput. Surv.*, vol. 55, no. 6, Dec. 2022, ISSN: 0360-0300. DOI: 10.1145/3533378. [Online]. Available: https://doi.org/10.1145/3533378.

[163] T. D. Doudali, D. Zahka, and A. Gavrilovska, "Cori: Dancing to the right beat of periodic data movements over hybrid memory systems", *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 350–359, 2021. DOI: 10.1109/IPDPS49936.2021.00043.

[164] S. Ghose, T. Li, N. Hajinazar, D. S. Cali, and O. Mutlu, "Demystifying complex workload-dram interactions: An experimental study", *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, 2019. DOI: 10.1145/3366708. [Online]. Available: https://doi.org/10.1145/3366708.

[165] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization", *IEEE Micro*, vol. 27, no. 3, pp. 63–72, 2007. DOI: 10.1109/MM.2007.56.

[166] P. J. Denning, "Working set analytics", *ACM Comput. Surv.*, vol. 53, no. 6, Feb. 2021, ISSN: 0360-0300. DOI: 10.1145/3399709. [Online]. Available: https://doi-org.kuleuven.e-bronnen.be/10.1145/3399709.

[167] K. Siu, D. M. Stuart, M. Mahmoud, and A. Moshovos, "Memory requirements for convolutional neural network hardware accelerators", in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 111–121. DOI: 10.1109/IISWC.2018.8573527.

[168] S. Sen and A. Raghunathan, "Approximate computing for long short term memory (lstm) neural networks", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2266–2276, 2018. DOI: 10.1109/TCAD.2018.2858362.

[169] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal, "In-memory computing in emerging memory technologies for machine learning: An overview", in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218505.

[170] C. Boden, T. Rabl, and V. Markl, "Distributed machine learning - but at what cost?", in *Machine learning systems workshop at the 2017 conference on neural information processing systems*, 2017.

[171] C. Huyen, *Designing machine learning systems.* " O'Reilly Media, Inc.", 2022.

[172] Y. Gao, Y. Liu, H. Zhang, Z. Li, Y. Zhu, H. Lin, and M. Yang, "Estimating gpu memory consumption of deep learning models", in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1342–1352, ISBN: 9781450370431. DOI: 10.1145/3368089.3417050. [Online]. Available: https://doi.org/10.1145/3368089.3417050.

[173] M. L. Puterman, "Markov decision processes", *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

[174] Y. Li, *Deep reinforcement learning*, 2018. arXiv: 1810.06339 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1810.06339.

[175] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning", *Now Foundations and Trends in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018, ISSN: 1935-8237. DOI: 10.1561/2200000071.

[176] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey", *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. DOI: 10.1109/MSP.2017.2743240.

[177] M. Marciniszyn Mehringer, F. Duguet, and M. Baust, *Benchmarking deep neural networks for low-latency trading and rapid backtesting on nvidia gpus*, Feb. 2023. [Online]. Available: https://developer.nvidia.com/blog/benchmarking-deep-neural-networks-for-low-latency-trading-and-rapid-backtesting-on-nvidia-gpus/.

[178] T. D. Doudali, *Kleio source code*, 2021. [Online]. Available: https://github.com/GTkernel/coeus-sim.

[179] JEDEC, *Jc-42.3c ddr4 sdram standard*, 2021. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-4a.

[180] D. Gouk, M. Kwon, J. Zhang, S. Koh, W. Choi, N. S. Kim, M. Kandemir, and M. Jung, "Amber: Enabling precise full-system simulation with detailed modeling of all ssd resources", in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 469–481. DOI: 10.1109/MICRO.2018.00045.

[181] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "Mqsim: A framework for enabling realistic studies of modern multi-queue ssd devices", in *Proceedings of the 16th USENIX Conference on File and Storage Technologies*, ser. FAST'18, Oakland, CA, USA: USENIX Association, 2018, pp. 49–65, ISBN: 9781931971423.

[182] A. Silberschatz, P. Galvin, and G. Gagne, *Operating System Concepts*. John Wiley & Sons, Incorporated, 2010, ISBN: 9780470233993. [Online]. Available: https://books.google.be/books?id=Go-GPwAACAAJ.

[183] C. Tirumalasetty, C. C. Chou, N. Reddy, P. Gratz, and A. Abouelwafa, "Reducing minor page fault overheads through enhanced page walker", *ACM Trans. Archit. Code Optim.*, vol. 19, no. 4, Sep. 2022, ISSN: 1544-3566. DOI: 10.1145/3547142. [Online]. Available: https://doi.org/10.1145/3547142.

[184] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping", *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pp. 278–287, 1999.

[185] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan, "Learning to utilize shaping rewards: A new approach of reward shaping", *ArXiv*, vol. abs/2011.02669, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:226254296.

[186] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling", *SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 128–138, May 2000, ISSN: 0163-5964. DOI: 10.1145/342001.339668. [Online]. Available: https://doi.org/10.1145/342001.339668.

[187] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems", in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 416–427. DOI: 10.1109/ISCA.2012.6237036.

[188] M. Hassan, H. Patel, and R. Pellizzoni, "A framework for scheduling dram memory accesses for multi-core mixed-time critical systems", in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015, pp. 307–316. DOI: 10.1109/RTAS.2015.7108454.

[189] S. Spanò, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Matta, A. Nannarelli, and M. Re, "An efficient hardware implementation of reinforcement learning: The q-learning algorithm", *IEEE Access*, vol. 7, pp. 186 340–186 351, 2019. DOI: 10.1109/ACCESS.2019.2961174.

[190] P. R. Gankidi and J. Thangavelautham, "Fpga architecture for deep learning and its application to planetary robotics", in *2017 IEEE Aerospace Conference*, 2017, pp. 1–9. DOI: 10.1109/AERO.2017.7943929.

[191] D. S. Malik, *Data structures using C++*. Cengage Learning, 2010.

[192] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[193] M. Li, Z. Huang, L. Chen, J. Ren, M. Jiang, F. Li, J. Fu, and C. Gao, "Contemporary advances in neural network quantization: A survey", in *2024 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2024, pp. 1–10.

[194] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference", in *Low-power computer vision*, Chapman and Hall/CRC, 2022, pp. 291–326.

[195] N. Laubeuf, J. Doevenspeck, I. A. Papistas, M. Caselli, S. Cosemans, P. Vrancx, D. Bhattacharjee, A. Mallik, P. Debacker, D. Verkest, F. Catthoor, and R. Lauwereins, "Dynamic quantization range control for analog-in-memory neural networks acceleration", *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, no. 5, Jun. 2022, ISSN: 1084-4309. DOI: 10.1145/3498328. [Online]. Available: https://doi.org/10.1145/3498328.

[196] N. Laubeuf, "Analog compute in memory and breaking digital number representations", in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022, pp. 1–2. DOI: 10.1109/VLSI-SoC54400.2022.9939611.

[197] S. L. Harris and D. Harris, "8 - memory systems", in *Digital Design and Computer Architecture*, S. L. Harris and D. Harris, Eds., Morgan Kaufmann, 2022, pp. 498–541, ISBN: 978-0-12-820064-3. DOI: https://doi.org/10.1016/B978-0-12-820064-3.00008-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128200643000088.

[198] imec, *Interuniversity microelectronics centre*. [Online]. Available: https://www.imec-int.com/en/about-us.

[199] S. Ganesh, *Ddr4 sdram - understanding timing parameters*, Feb. 2023. [Online]. Available: https://www.systemverilog.io/design/understanding-ddr4-timing-parameters/.

[200] J. L. Henning, "Spec cpu2006 benchmark descriptions", *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006, ISSN: 0163-5964. DOI: 10.1145/1186736.1186737. [Online]. Available: https://doi.org/10.1145/1186736.1186737.

[201] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb", *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pp. 143–154, 2010. DOI: 10.1145/1807128.1807152. [Online]. Available: https://doi.org/10.1145/1807128.1807152.

[202] T. D. Doudali and A. Gavrilovska, "Coeus: Clustering (a)like patterns for practical machine intelligent hybrid memory management", *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 615–624, 2022. DOI: 10.1109/CCGrid54584.2022.00071.

[203] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool", *URL: http://www. drampower. info*, vol. 22, 2012.



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
ESAT-INSYS
Kapeldreef 75 - box 4055
B-3001 Leuven
https://www.kuleuven.be/wieiswie/en/unit/regional/55685388