THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

A Taxonomy and Design Methodology for Hybrid Memory Systems

DMITRY KNYAGININ

Division of Computer Engineering
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014

A Taxonomy and Design Methodology for Hybrid Memory Systems
Dmitry Knyaginin

Copyright © 2014 Dmitry Knyaginin

Technical report 117L
ISSN 1652-876X
Department of Computer Science and Engineering
Computer Architecture Research Group

Division of Computer Engineering
Chalmers University of Technology
SE-412 96 GOTHENBURG, Sweden
Phone: +46 (0)31-772 10 00

Author e-mail: `dmitry.knyaginin@chalmers.se`

Printed by Chalmers Reproservice
Gothenburg, Sweden 2014

A Taxonomy and Design Methodology for Hybrid Memory Systems

Dmitry Knyaginin

Division of Computer Engineering, Chalmers University of Technology

ABSTRACT

The number of concurrently executing processes and their memory demand in multi-core systems continue to grow. Larger and still fast main memory is needed for meeting the demand and avoiding an increase in backing store accesses that are much slower and less energy efficient than main memory accesses. Luckily, Non-Volatile Memory (NVM) technologies can bridge the cost, density, performance, and energy efficiency gaps between backing store and DRAM, the conventional main memory technology. Thus, NVM can be combined with DRAM into *hybrid* main memory striving to enjoy both the larger capacity enabled by NVM and the speed and energy efficiency of DRAM.

NVM adds a new dimension to the system design space inspiring researchers to investigate sophisticated hybrid memories. This has resulted in a large body of work that, unfortunately, lacks systematization. The thesis at hand addresses this problem by proposing a taxonomy and a notation for classifying hybrid main memory organizations.

The design space of hybrid systems is large, and the best partitioning of resources between DRAM and NVM is nontrivial. The high implementation and computation efforts of detailed modeling impede extensive design space exploration required for finding the most promising design points. This thesis aids such extensive exploration by proposing a workload methodology and first-order models for system-level execution time and energy. Next, the thesis contributes with *Rock*, an insightful performance model showing how memory system throughput can be boosted by installing more DRAM and NVM thus motivating Design-time Resource Partitioning (DRP). The lack of an approach suitable for extensive partitioning is addressed by proposing *Crystal*, a DRP method powered by the system-level models and framing partitioning as an optimization problem, such that the first-order nature of the models does not restrict its applicability, as shown by validation. Crystal is practical and facilitates early and rapid DRP finding promising design points for further detailed evaluation. For instance, Crystal shows how for specific workloads higher performance and energy efficiency can be achieved by employing NVM with the speed and energy consumption of NAND Flash instead of a much faster and more energy efficient NVM technology like phase-change memory.

Keywords: DRAM, Non-Volatile Memory, Taxonomy, Methodology, Design Space Exploration, System-Level Models, Performance, Energy Efficiency.

Preface

The thesis at hand is for the degree of Licentiate of Engineering, a Swedish degree between MSc and PhD. Parts of the contributions presented in this thesis have previously been accepted to workshops:

- **Dmitry Knyaginin**, Sally A. McKee, and Georgi N. Gaydadjiev, “A hybrid main memory systems taxonomy,” in *Memory Architecture and Organization Workshop, co-located with Embedded Systems Week*, Tampere, Finland, Oct. 2012, pp. 1–6.
- **Dmitry Knyaginin**, Georgi N. Gaydadjiev, and Per Stenström, “Crystal: A design-time resource partitioning method for hybrid main memory,” in *Workshop on Reproducible Research Methodologies, co-located with Int. Symp. on High Performance Computer Architecture*, Orlando, FL, USA, Feb. 2014, pp. 1–6.

The following manuscript contains parts of the contributions presented in this thesis and has been submitted to an international conference:

- **Dmitry Knyaginin**, Georgi N. Gaydadjiev, and Per Stenström, “Crystal: A design-time resource partitioning method for hybrid main memory,” Under review since Mar. 2014.

Acknowledgments

I cannot thank enough my advisor, Per Stenström, for his inspiration, mentorship, and guidance. Without Per I would not be able to navigate in the deep waters of research.

Many thanks to my co-advisor and examiner, Georgi N. Gaydadjiev, for his input and feedback, and for the opportunities to practice research discussions in Russian.

I am grateful to Sally A. McKee for her input at the early stages of my PhD program, for the successful collaboration on the taxonomy paper, and especially for her help with editing the text of the first draft of Crystal.

A big thank you to Lars Svensson for his support, advice, and weekly meetings during 2011-12. The work done during that period laid the ground for this thesis.

I would like to extend my gratitude to everybody with whom I have been fortunate to discuss my work. In particular, I would like to thank Anurag Negi and Magnus Själander for countless discussions inside and outside the premises of Chalmers, and Vinay Jethava for his help with various mathematical questions.

Many thanks to Eva Axelsson, Gerardo Schneider, Jan Jonsson, Johan Karlsson, Koen Lindström Claessen, Malin Nilsson, Marianne Pleen-Schreiber, Peter Helander, Rune Ljungbjörn, and Tiina Rankanen for providing administrative and technical support. I would also like to thank Bhavishya Goel and Jacob Lidman for maintaining the *ttitania* cluster. A special thank you to Anna-Lena Karlsson for her help during my first days at Chalmers and for finding a beautiful apartment for me.

I express my deep appreciation to everybody at the Department of Computer Science and Engineering and especially at the Division of Computer Engineering for contributing to the warm and professional environment. In particular, I thank Alen Bardizbanyan and Angelos Arelakis for the productive atmosphere inside our office.

Finally, I would like to thank all my friends. Special thanks to Alex Geppert, Dima Mishenin, Jeff Jung, John Moyes, Katarina Steffenburg, Martin Lever, Morris Stuttard, Oleg Bogdanov, and Vilhelm Verendel for their support, and to my family—Victoria, Vladimir, and Oleg—for their love.

The thesis at hand is based upon work supported by the Swedish Research Council (Vetenskapsrådet) under the Chalmers Adaptable Multicore Processing Project (CHAMPP).

Dmitry Knyaginin
Gothenburg, May 2014

Contents

List of Figures

List of Tables

Acronyms

1T	One Transistor
1T1C	One Transistor and One Capacitor
1T1MTJ	One Transistor and One Magnetic Tunnel Junction
1T1R	One Transistor and One Resistor
CPU	Central Processing Unit
DDR	Double Data Rate
DIMM	Dual In-line Memory Module
DLL	Delay-Locked Loop
DRA	Design-time Resource Allocation
DRAM	Dynamic Random Access Memory
DRP	Design-time Resource Partitioning
FeRAM	Ferroelectic Random Access Memory
HDD	Hard Disk Drive
I/O	Input/Output
L1C	Level-One Cache
LLC	Last Level Cache
LRU	Least Recently Used
MLC	Multi-Level Cell
MRAM	Magnetoresistive Random Access Memory
MTJ	Magnetic Tunnel Junction

NVM	Non-Volatile Memory
ONFI	Open NAND Flash Interface
OS	Operating System
PCM	Phase-Change Memory
RDP	Run-time Data Placement
RRAM	Resistive Random Access Memory
SATA	Serial Advanced Technology Attachment
SCM	Storage-Class Memory
SDRAM	Synchronous Dynamic Random Access Memory
SLC	Single-Level Cell
SRAM	Static Random Access Memory
SSD	Solid-State Disk
STT-MRAM	Spin Transfer Torque Magnetoresistive Random Access Memory
VLSI	Very Large Scale Integration

1 Introduction

The contemporary trend of increasing core counts in computing systems [1] implies growing numbers of concurrently executing programs in the system workload. The problem sizes of contemporary programs continue to grow, too, especially in the domains of scientific computing, databases, and consolidated cloud environments. These two trends result in a continuously increasing demand for larger and still fast memory of computing systems.

A typical high-performance memory system is organized as a hierarchy. Central Processing Unit (CPU) memory, such as the register file and a cache hierarchy from Level-One Cache (L1C) down to Last Level Cache (LLC), is followed by larger but slower main memory that is further followed by even more large and slow backing store. Caches are conventionally implemented as Static Random Access Memory (SRAM), main memory is built from Dynamic Random Access Memory (DRAM), and backing store is composed of magnetic Hard Disk Drives (HDDs). Each of the technologies has pros and cons, offering different bit densities (the number of bits per mm^2), bit costs,

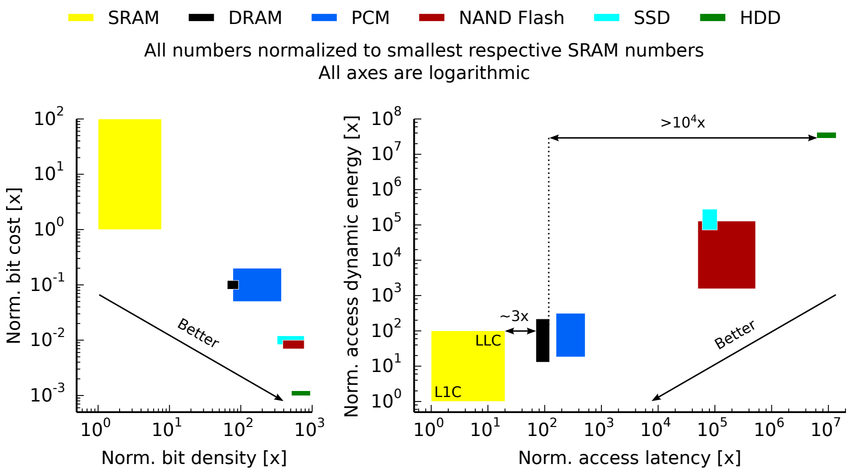


Figure 1.1: *Normalized bit cost vs. bit density and access dynamic energy vs. access latency of memory technologies*

access latencies, and dynamic energies. The purpose of the hierarchy is to enjoy the pros while hiding the cons that negatively impact the performance and energy efficiency of the entire system. However, the gaps between the adjoining levels of the conventional memory hierarchy are rather large and uneven. For instance, the access latency gap between SRAM LLC and DRAM main memory is about three times, while between DRAM and HDD this gap is at least four orders of magnitude. The larger the gaps, the more challenging it is to hide the cons of the hierarchy levels. Thus, bridging or reducing the gaps between DRAM and HDD remains a major design challenge.

Fortunately, technological progress in the field of solid-state memory [1, 2] has contributed a number of promising Non-Volatile Memory (NVM) technologies that fit in the gaps between DRAM and HDD. One such technology, Phase-Change Memory (PCM) [3, 4], is gaining maturity and has the potential to offer a higher bit density than DRAM but at a penalty of slower and less energy efficient accesses. Another technology, NAND Flash, is mature and offers a higher bit density and a lower bit cost than DRAM, but accessing it is even slower and less energy efficient than accessing PCM. NVM has enabled high-performance disks, such as Solid-State Disk (SSD) built from NAND Flash.

Figure 1.1 aggregates the bit density, cost, access latency, and dynamic energy numbers of representative memory technologies [1, 5–12], where access denotes both random reads and writes of typical sizes. All of the numbers are normalized to the smallest respective numbers of SRAM and are expressed as ratios ($\times$). They are of a sufficient accuracy to illustrate the gaps between the technologies. The numbers are represented by ranges, because they depend on many factors, including the implementation process and the type and size of access. Such ranges of values for each technology and characteristic are shown by the rectangles. For instance, consider the right part of the figure (normalized access dynamic energy vs. access latency). The bottom left corner of the SRAM rectangle shows the access latency and dynamic energy of L1C, and the top right corner shows those of LLC. The left edge of the DRAM rectangle depicts the latency of a read access, the right edge the latency of a write access, the bottom edge the dynamic energy of a read access served by a single DRAM device, and the top edge the dynamic energy of a write access served by a rank of eight less energy-efficient DRAM devices. For the purposes of this chapter no further details need to be discussed, as the figure clearly shows that the gaps exist even if broad ranges of technology characteristics are considered.

NVM technologies can help bridge the gaps between the levels of the conventional memory hierarchy from the main memory side by enabling *hybrid main memory*, i.e., main memory divided into two or more partitions where each partition is optimized for specific purposes (e.g., performance, energy efficiency, capacity, or cost). In addition, NVM technologies can help reduce the gaps from the backing store side by enabling faster and more energy efficient disks, such as SSD. This thesis investigates hybrid main memory. The rest of this chapter details the problems addressed by the thesis, lists my contributions, and explains the organization of the thesis.

1.1 Problem Statements

Hybrid main memory systems have attracted vivid interest among researchers that have proposed and investigated a plurality of hybrid system organizations [13–30]. Unfortunately, this large body of work lacks systematization. This complicates positioning new hybrid memory proposals within the existing body of work. Systematization of hybrid main memory system organizations is one of the problems addressed by this thesis.

An obvious design challenge for hybrid main memory is that memory resources are restricted from the physical, technological, and cost perspectives. This boils the design process down to optimization of resources under design goals that can differ. For instance, if the goal is to reduce the cost of main memory, DRAM can be combined with a less expensive NVM technology to create hybrid memory of a lower cost but the same capacity as DRAM-only main memory. If the goal is to increase capacity, DRAM can be combined with a bit-denser NVM technology to store more bits in the same physical area as DRAM-only main memory. Finding the best amounts of DRAM and NVM is a nontrivial design-time resource partitioning problem. The best solution depends on many factors, including the workload, properties of memory technologies, and characteristics of subsystems other than main memory and disk (e.g., the CPU). Characteristics of hybrid systems implementing different partitioning options can vary widely. This makes design-time partitioning a fundamental problem for hybrid main memory systems with a multi-dimensional design space.

Partitioning of main memory resources between different technologies has been studied using simulators [14–16] and prototyping [13]. Simulation typically involves a significant implementation overhead and consumes large computational resources, impeding extensive design space exploration. Prototyping with, e.g., a virtual machine monitor [13], requires a substantial implementation effort and restricts exploration to a given host configuration (e.g., the total capacity of main memory). The lack of a design-time partitioning method suitable for rapid and extensive hybrid memory design space exploration is another problem tackled by this thesis.

1.2 Contributions

The thesis addresses the problems stated above by making the following contributions:

- First, a taxonomy of hybrid main memory organizations is proposed [31]. The taxonomy is applied to classify and illustrate the existing diversity of hybrid systems, highlighting organizations that have received most attention among researchers.
- Next, the thesis contributes models for first-order estimation of system-level execution time and energy of conventional and hybrid main memory systems. The models embody a light-weight tool for obtaining insights about memory system design trade-offs. They power a model named *Rock*, that illustrates the potential of design-time resource partitioning for improving memory system performance.
- Further, the models enable the final contribution of the thesis: *Crystal*, a design-time resource partitioning method for hybrid main memory [32]. Crystal facilitates quick and early identification of the most promising resource partitioning options for detailed evaluation. Thus, Crystal greatly simplifies the design process and represents a valuable addition to the system designer's toolbox.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 first explains how programs and workloads—the stimuli of the memory system—are represented and classifies them by memory behavior. The chapter identifies the type of workloads that benefit from hybrid main memory and introduces programs used for composing workloads in this thesis. Next, memory technologies—the building blocks of hybrid memory systems—are presented in Chapter 3 from basic memory cell properties to device organization and operation to device modeling. The device models provide estimates of device characteristics used throughout this thesis (including Figure 1.1). Chapter 4 presents the first contribution, a taxonomy of hybrid main memory systems, systematizing the existing plurality of their organizations. Chapter 5 leverages the previous chapters and presents the second and third contributions of this thesis: Section 5.1 describes the models for system-level execution time and energy, Section 5.2 introduces Rock, and Section 5.3 presents Crystal. The thesis is concluded by Chapter 6 offering a review of the contributions in Section 6.1 and an overview of future work in Section 6.2.

2 Workload Methodology

Workloads are the stimuli driving design decisions in computing systems. It takes a certain amount of time and energy to produce results for a particular workload on a particular system, and reducing system-level execution time and/or energy are major design goals. The goals can be achieved by optimizing system hardware, software, or the code of the workload. This thesis is about optimizing the main memory of computing systems by increasing its capacity for executing workloads faster and with less energy.

A workload may comprise one or more programs that are single- or multi-threaded. This thesis considers multi-program workloads where each program is single-threaded. Benefits from memory system optimization realizable by workloads depend on their memory behavior. This thesis uses workloads for which execution time and energy can be reduced by increasing main memory capacity.

Main memory optimizations can be evaluated using simulators of different degrees of detail (e.g., full-system, system-call emulation, execution-driven, or trace-driven) or analytic memory models. The latter approach represents systems at the highest level of abstraction compared to the other approaches and thus is the fastest. As has been stated in Section 1.1, the design space of interest in this thesis is large and detailed modeling impedes its extensive exploration. Analytic modeling facilitates such extensive exploration and thus suits the purposes of this thesis best. It requires representing each program by a profile (described in Section 2.1), and the procedure of recording profiles is labeled *profiling* here. The aggregate behavior of programs in a workload is modeled by distributing available main memory capacity among the programs in a way that represents their memory behavior (as defined by their profiles) as well as memory management.

The next section explains profiling of single-thread programs, Section 2.2 describes the representation of entire workloads, and Section 2.3 introduces a workload classification by the kind of benefits that workloads enjoy from memory system optimizations. Section 2.4 presents the profiling setup and the profiles of programs used for composing workloads in this thesis, and Section 2.5 summarizes the chapter.

2.1 Program Profiling

The goal of profiling in this thesis is to create compact representations of programs sufficient for quick evaluation of main memory optimization ideas in Chapter 5. Such profiling requires the detailed execution and analysis of a program just once, when recording its profile, as described further in this section. Each evaluation iteration reuses the profile instead of running the program in detail, and this dramatically reduces the computation effort of evaluation.

Memory system optimization is typically performed for specific program execution intervals of interest. Information in main memory is managed at the granularity of an Operating System (OS) page, and pages referenced by a program during an execution interval constitute the program's *working set* of that interval [33]. In this thesis, each program is profiled for its dominant execution interval in the *steady state*, i.e., after system warmup, when no cold (compulsory) misses in the LLC occur. A profile contains:

1. The miss curve below LLC, denoted by $Miss(C)$, i.e., the number of main memory capacity misses as a function of its capacity C, where main memory is fully-associative and employs the Least Recently Used (LRU) replacement policy. In general, the miss curve can represent all memory references of the program, but here LLC filters them, such that the miss curve represents only accesses below LLC. It shows how the number of disk accesses reduces as the capacity of main

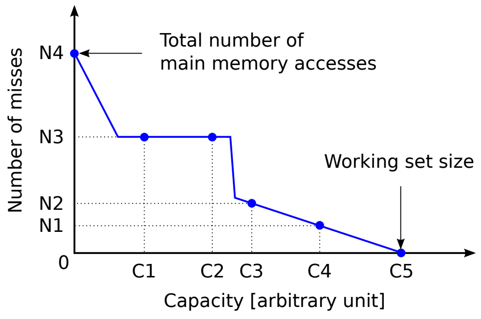


Figure 2.1: *Example miss curve*

memory is increased, and is thus an essential program characteristic for evaluation of memory system optimizations that increase main memory capacity.

2. The fraction of writes below LLC, denoted by fWr, required for distinguishing read and write misses at each point of the miss curve: Among $y = Miss(x)$ misses for each capacity x, $(1 - fWr)\cdot y$ are read misses, and $fWr\cdot y$ are write misses. Distinguishing reads and writes is important, because memory technologies can have asymmetric read and write access characteristics, as described in Chapter 3.

3. The time spent on computation plus all cache accesses, denoted by T_{CPU}, required for representing the constant part of execution time that is not affected by main memory optimization.

The miss curve represents the size of the working set and the reuse distance (LRU stack distance [34]) of pages within the working set, as explained below. Figure 2.1 shows an example miss curve, where capacity is expressed in arbitrary units. The total number of main memory accesses is defined by the $Miss(0)$ point (e.g., $N4$ in Figure 2.1). Each point of the miss curve corresponds to the number of capacity misses made by the program if it is given the amount of memory denoted by the horizontal coordinate of the point, where that memory is fully associative and implements the LRU replacement policy. The number of disk accesses is given by the point with the horizontal coordinate equal to the main memory capacity allocated to the program. For instance, if the program is allocated capacity $C4$, it makes $N1$ disk accesses. The working set size is given by the horizontal coordinate of the point where the number of misses is zero, because cold misses are not profiled (e.g., $C5$ in Figure 2.1).

The reuse distance of pages within the working set is represented by the steepness of the miss curve: The number of misses y at each capacity x, expressed in the number of pages, is equal to $N_{total} - N_{hit} - N_{cold}$, where N_{total} is the number of accesses to all pages in the working set (the total number of main memory accesses), N_{hit} is the number of accesses to pages with reuse distances less than x, and N_{cold} is the number of cold misses ($N_{cold} = 0$ because cold misses are not profiled). If the miss curve decreases from point (x_0, y_0) to point (x_1, y_1), i.e., $x_0 < x_1$ and $y_0 > y_1$, it means that there are pages with reuse distances greater than or equal to x_0 and less than x_1, where both x_0 and x_1 are expressed in the number of pages. Thus, increasing main memory capacity from x_0 to x_1 reduces the number of disk accesses. For instance, such are capacities $C3$ and $C4$ in Figure 2.1, respectively ($N2 > N1$). If the miss curve is flat between the two points (forms a flat plateau), i.e., $x_0 < x_1$ but $y_0 = y_1$, it means that pages with reuse distances greater than or equal to x_0 actually have reuse distances greater than or equal to x_1. Thus, increasing main memory capacity from x_0 to x_1 does not reduce the number of disk accesses. For instance, such are capacities $C1$ and $C2$ in Figure 2.1, respectively (the number of misses is $N3$ at both $C1$ and $C2$). In general, a program can have pages with different reuse distances, making the shape of the program's miss curve complex: It may contain a number of decreasing and flat fragments.

The shape of the miss curve is linked to the concept of *utility* of memory capacity as follows. A program's utility of a given capacity can be defined as the reduction in the number of misses that the program enjoys if it obtains the capacity. The steeper the miss curve between points (x_0, y_0) and (x_1, y_1), i.e., the greater $y_0 - y_1$, the higher the program's utility of main memory capacity $x_1 - x_0$. The flatter the miss curve between the two points, i.e., the smaller $y_0 - y_1$, the lower the program's utility of that capacity. For instance, the utility of capacity $C2 - C1$ in Figure 2.1 is zero, and the utility of capacity $C3 - C2$ is $N3 - N2$. The concept of utility is used in the next section, when defining the aggregate behavior of programs in a multi-program workload.

2.2 Workload Representation

Multi-program workloads where each program is defined by its profile are represented as follows. All programs in a workload are single-threaded, assigned one per core, and run concurrently. Available main memory capacity is distributed among them at design-time by a procedure labeled *Design-time Resource Allocation (DRA)*. Each program is allocated memory capacity according to its utility of capacity relative to that of the other programs in the workload. DRA represents characteristics of both the programs and memory management. A memory management policy is labeled *low-utility* DRA policy if it allocates more memory capacity to programs with low utility of it, and it is labeled *high-utility* if it allocates more capacity to programs with high utility of it.

A program's utility of a particular memory capacity slice is estimated by the program's miss curve. For instance, programs A and B have been allocated 1MB each, and there is another 1MB slice to allocate. Program A has 10M misses at capacity 1MB and 1M misses at capacity 2MB, and program B has 10M and 9M misses at these capacities, respectively. The difference in the number of misses between the two capacities is greater for program A, thus it has a higher utility of the 1MB slice than program B and wins it according to the high-utility DRA policy. Program B has a lower utility of the slice and thus wins it according to the low-utility DRA policy.

There are multiple ways of implementing DRA. For a given multi-program workload and DRA policy, the final distribution depends on an initial allocation and the granularity of DRA. The final distribution is represented by main memory slices allocated to each program in the workload. The initial allocation defines main memory capacity allocated to each program by default (the minimum that each program gets when it starts execution). The granularity of DRA defines how much memory is awarded to a program at each iteration of DRA. Figure 2.2 shows my implementation of the high- and low-utility DRA. The algorithm starts with an initial allocation where each program in the workload gets slice cs_{min} of main memory capacity. Flag $done$ for each program indicates if allocation has been completed, i.e., if the program has been allocated enough main memory capacity to fit its entire working set. The working set size of each program, denoted by wss_i, is the horizontal coordinate of the last point of the program's miss curve. C_{left} keeps track of available main memory (left for allocation). After initial allocation, the algorithm distributes available memory capacity iteratively at the granularity of a capacity slice delta (Δcs). For each program, the difference in the number of misses ΔN is estimated between two points on the miss curve: one at the current slice and one at the new capacity that is the current slice plus the slice delta. If the current capacity is such that it entirely fits the working set of a program, the program is labeled $done$. If the new capacity exceeds the working set size of a program, the last point of the miss curve is extended (then the number of misses at the new capacity is zero, since cold misses are not profiled). On the next step, a list is created of ΔN of programs that still participate in DRA, i.e., those not yet labeled as $done$. If the high-utility policy is chosen, the list is sorted in descending order, such that the program with the largest ΔN (the highest utility) is on the top of the

```
Data:
    $n$ programs defined by their miss curves $Miss_i(C)$ and working set sizes $wss_i$, $0 \leq i < n$
    Total main memory capacity $C_{total}$
    Minimum capacity slice $cs_{min}$ allocated per program
    Capacity slice delta $\Delta cs$ that is iterative allocation granularity
    DRA policy type $dra\_type$ (high- or low-utility)
Result:
    Capacity slices for each program $cs_i$, $0 \leq i < n$
foreach program $i$, $0 \leq i < n$ do
    $cs_i = cs_{min}$                                   /* Initial allocation */
    $done_i = False$
$C_{left} = C_{total} - n \cdot cs_{min}$
while $C_{left}$ and allocation not done for all programs do
    foreach program $i$, $0 \leq i < n$ do
        if $cs_i \geq wss_i$ then
            $\Delta N_i = 0$                   /* Allocation for program is done */
            $done_i = True$
        else
            if $cs_i + \Delta cs \geq wss_i$ then
                $\Delta N_i = Miss(cs_i) - Miss(wss_i)$
            else
                $\Delta N_i = Miss(cs_i) - Miss(cs_i + \Delta cs)$
    Create list $[\Delta N]$ of $\Delta N_i$ for $0 \leq i < n$ such that $done_i = False$
    if $dra\_type$ is high-utility then
        Sort $[\Delta N]$ in descending order               /* High-utility DRA */
    else
        Sort $[\Delta N]$ in ascending order                /* Low-utility DRA */
    Programs with same $\Delta N$ as program on top of list are $winners$
    foreach program in $winners$ do
        if $C_{left}$ then
            $cs_i \mathrel{+}= \Delta cs$                        /* Award $\Delta cs$ to program */
            $C_{left} \mathrel{-}= \Delta cs$
return $cs_i$, $0 \leq i < n$
```

Figure 2.2: *Algorithm of high- and low-utility DRA*

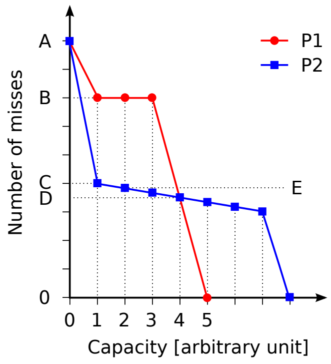


Figure 2.3: *Miss curves for illustrating high-utility, low-utility, and utility-agnostic DRA policies*

list. If the low-utility policy is chosen, the list is sorted in ascending order, such that the program with the smallest ΔN (the lowest utility) is on the top. Starting with the program on the top of the list, programs with the same ΔN are labeled $winners$, awarded the capacity slice delta, and available memory capacity C_{left} is updated accordingly. When there is not enough memory available, some programs might not be awarded the slice delta even if they are $winners$. This iterative DRA repeats until either all programs in the workload have been allocated enough capacity to fit their entire working sets, or until there is no more main memory available.

The total number of disk accesses for a given workload is not always smaller if the high-utility DRA policy is employed instead of the low-utility one. Consider the example miss curves of programs $P1$ and $P2$ in Figure 2.3. $P1$ and $P2$ form a workload. Point 0 on the horizontal axis corresponds to an initial capacity allocation, and there are five capacity slices (expressed in arbitrary units) to distribute between the programs. Let us first consider the low-utility DRA policy. $P1$ wins the first slice because $A - B < A - C$, the second and third slices because $B - B < A - C$, and the fourth and fifth slices because $B - D < A - C$ and $D - 0 < A - C$, respectively. The number of disk accesses is zero for $P1$, A for $P2$, and hence A for the entire workload. Let us now consider the high-utility DRA policy. $P2$ wins the first slice because $A - C > A - B$. $P1$ wins the second slice because $A - B > C - E$. Now $P1$ has hit its flat plateau, and $P2$ wins the remaining three slices because its miss curve is decreasing. The number of disk accesses is B for $P1$, D for $P2$, and hence $B + D$ for the workload. The high-utility

DRA policy has resulted in a greater number of disk accesses made by the workload than the low-utility DRA policy ($B + D > A$).

Another distinct DRA policy is agnostic to utility and distributes the available capacity evenly among programs in the workload, as long as their working sets do not fit entirely into their respective capacity slices. I label this policy *utility-agnostic* DRA policy. For the example miss curves in Figure 2.3 and the five capacity slices in question, the policy results in three slices allocated to $P1$ and two slices allocated to $P2$, given that allocation starts with $P1$. The number of disk accesses made by the workload is thus $B + E$, which is by $E - D$ greater than that if the high-utility DRA policy is used.

2.3 Workload Classification

A workload is labeled *in-memory* if it does not access disk in the steady state (after system warmup), i.e., its working set fits entirely into main memory. Likewise, a workload is labeled *not-in-memory* if it accesses disk in the steady state, i.e, its working set size exceeds the capacity of main memory.

As a result of memory system optimization, main memory capacity can be increased. For instance, the baseline memory technology can be partly replaced with a technology that enables storing more bits. In-memory workloads do not benefit in terms of execution time and energy from a larger memory capacity, because their working sets fit into the baseline capacity, and increasing capacity cannot decrease the number of capacity misses. Not-in-memory programs benefit from a memory capacity increase as long as there are pages with reuse distances greater than or equal to the baseline capacity and less than the increased capacity (where the capacities are expressed in the number of pages).

2.4 Profiling Setup and Program Profiles

A program profile contains the miss curve ($Miss(C)$), the fraction of writes (fWr), and the computation and cache time (T_{CPU}), as described in Section 2.1. A profile of a program can be extracted from its memory access trace and execution statistics recorded during the interval of interest. For instance, they can be obtained with: 1) a simulator (e.g., gem5 [35]) including models of a CPU and its cache hierarchy; or 2) an execution-driven (e.g., Pin [36] or Valgrind [37]) model of a cache hierarchy. In this thesis, I compile programs from SPEC CPU2006 [38] and NPB 3.3 [39] for the 64-bit extension of x86 and run them individually on gem5 to record main memory access traces and execution statistics in the system-call emulation mode (system calls are not profiled). For each program, the miss curve is reconstructed from its main memory access trace using Mattson's stack algorithm [34] at the granularity of a 4KB page. gem5 is configured to simulate an in-order CPU running at 1GHz with 64KB, 4-way, LRU, split L1 caches and a 1MB, 8-way, LRU, unified L2 cache. The cache line size is 64B. The SPEC CPU2006 programs are profiled with reference inputs and the NPB 3.3 programs are profiled with large problem sizes (e.g., size `C` for `CG`). For each program, I identify one major 2B-instruction execution interval using the SimPoint methodology [40] and simulate it after fast-forwarding and warming the system for 500M instructions. A single execution interval does not fully represent an entire program, but I observe that the working set sizes of the major execution intervals closely match those of the entire programs.

For brevity, I choose a subset of programs with larger working sets (above 150MB) and diverse behavior, as described below. The programs are:

- `CG`, a conjugate gradient kernel with irregular memory accesses as a result of sparse matrix-vector multiplications;
- `470.lbm`, a computational fluid dynamics program implementing a lattice Boltzmann method;
- `429.mcf`, a combinatorial optimization program implementing single-depot vehicle scheduling using a network simplex algorithm accelerated with a column generation;
- `458.sjeng`, an artificial intelligence program playing chess by implementing game tree search and pattern recognition; and
- `450.soplex`, a simplex linear program solver employing sparse linear algebra.

Program names are used as profile names for simplicity. For instance, `lbm` denotes `470.lbm` with the reference inputs during its major 2B-instruction execution interval. Likewise, `mcf`, `sjeng`, and `soplex` denote `429.mcf`, `458.sjeng`, and `450.soplex`, respectively.

Table 2.1 shows that the working set sizes of `CG` and `lbm` are similar, but `CG` has about 100$\times$ smaller fraction of writes (fWr) and about 2$\times$ greater computation and cache time (T_{CPU}). Evaluating a memory system optimization first using workloads containing `CG` and then workloads containing `lbm` can help reveal how the results of the optimization depend on fWr and T_{CPU}. The working set of `mcf` is the largest among the programs, and that of `sjeng` is the smallest (about 10$\times$ smaller than the working set of `mcf`, about 2.5$\times$ smaller than those of `CG` and `lbm`, and about 1.5$\times$

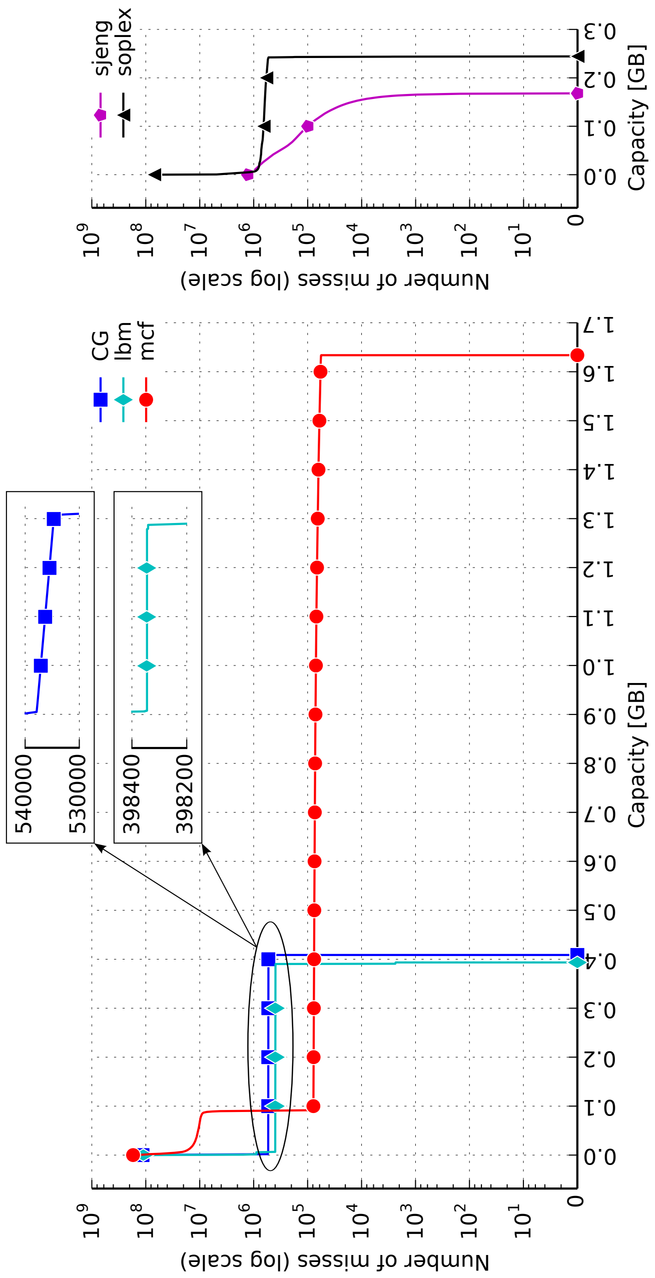


Figure 2.4: *Miss curves of* `CG`*,* `lbm`*,* `mcf`*,* `sjeng`*, and* `soplex`

Table 2.1: *Selected characteristics of* `CG`, `lbm`, `mcf`, `sjeng`, *and* `soplex`

	CG	**lbm**	**mcf**	**sjeng**	**soplex**
Working set size [MB]	419	403	1674	172	251
fWr **[%]**	0.4	42.9	20.2	44.1	16.5
T_{CPU} **[s]**	8.4	4.4	6.6	3.4	4.6

smaller than that of `soplex`). The programs can be used for composing multi-program workloads with aggregate working set sizes that differ by increments from hundreds of megabytes to gigabytes. Such workloads are essential for the evaluation of memory system optimizations that increase main memory capacity.

Figure 2.4 shows that the miss curve of each program, except `sjeng`, has a distinct plateau representing the bulk of the working set, and a relatively small part referenced significantly more than the remainder of the working set. For `mcf`, such part is about 100MB and forms a plateau itself. Decreasing fragments of the miss curves illustrate that pages have gradually increasing reuse distances. Flat fragments of the miss curves illustrate that pages have equal reuse distances. For instance, `lbm`'s plateau is totally flat, and thus the pages of the bulk of its working set share the same reuse distance that is the size of the entire working set. On the contrary, `sjeng` has no plateaus, its miss curve decreases monotonically, and so the pages of its working set have different, gradually increasing reuse distances.

2.5 Summary

Programs in a multi-program workload are represented by compact profiles. The profiles are recorded once and reused throughout the evaluation of main memory optimizations. Memory management and the way programs affect each other's behavior are represented by DRA. The high- and low-utility DRA policies are two distinct cases of DRA driven by the program utility of memory capacity. The utility-agnostic DRA is a distinct example of a policy that neglects the program utility of memory capacity. Not-in-memory workloads are of primary interest in this thesis, since they benefit in terms of execution time and energy from a main memory capacity increase that can be achieved by memory system optimizations. The next chapter presents memory technologies that make such optimizations possible.

3 Memory Technologies

Memory technologies are the building blocks of memory systems. A number of technologies with different characteristics exist today, and they have to be well understood in order to choose the most appropriate technologies for a particular memory system. The characteristics vary among technologies and even across different device implementations of the same technology. In order to correctly interpret numbers reported by various sources and to be able to model the operation of memory devices, it is necessary to understand their basic organization and operational principles. This chapter describes these principles without going too deep into the nuances of Very Large Scale Integration (VLSI). Section 3.1 presents memory technologies as follows. First, the fundamental cell characteristics are presented: basic memory cell physics (the principles of storing data in the memory element), cell structure (the mechanism of accessing the memory element), and cell bit density (the number of bits per cell, i.e., the number of bits encoded by a single physical state of the memory element). They determine such memory characteristics as data retention (the period of time during which the cell remains stable without power) and write endurance (the maximum number of cell state modifications that the cell can perform reliably). Next, Section 3.2 presents ways of organizing cells into arrays and accessing them on the device level that eventually affect the device access latencies, access energies, and maintenance overhead (e.g., for maintaining data integrity within the arrays and during transfers across the device interface). The specifics of device organization, implementation, and fabrication determine the area efficiency and thus the bit density of the device. Section 3.3 introduces my analytic, device-level models for estimating the timing, energy, and power characteristics of DRAM, PCM, NAND Flash, SSD, and HDD.

3.1 Basic Memory Cell Properties

Memory technologies are classified as *charge-based* if they represent data as electric charge. Otherwise, e.g., if data are represented as resistance, memory technologies are classified as *non-charge-based*. Memory technologies are classified as *Single-Level Cell (SLC)* if one memory cell state encodes one bit of information, and as *Multi-Level Cell (MLC)* if it encodes two or more bits. Memory technologies are typically classified as *volatile* if their cells lose state within seconds after disconnecting from power, otherwise they are classified as *non-volatile*.

3.1.1 DRAM

DRAM is a charge-based, One Transistor and One Capacitor (1T1C), SLC memory technology. One bit of data is stored as charge on the capacitor of the memory cell. The capacitor is accessed via the transistor. Charge leaks from the capacitor, thus DRAM is a volatile technology. It is typically required for data integrity that charge is *refreshed* each 64ms at temperatures from $0C^\circ$ to $85C^\circ$ and each 32ms at temperatures from $85C^\circ$ to $95C^\circ$ [41]. Primarily because the charge is unstable, one DRAM cell encodes a single bit. Reading (sensing) the cell depletes the capacitor, thus DRAM reads are destructive. DRAM cells have endurance of about 10^{16} accesses, primarily defined by the endurance of the access transistor. DRAM is a mature technology and is conventionally employed in the main memory of computing systems. It is used as the baseline memory technology in this thesis.

3.1.2 Flash

Flash is a charge-based, One Transistor (1T), SLC/MLC memory technology. The cell transistor has both the control gate and a *floating* gate that is not directly connected to any control signals. Data are stored as charge on the floating gate of the transistor. The cell is programmed by injecting electrons into the floating gate and erased by ejecting them out from the floating gate by means of quantum tunneling. The cell can be programmed only if it is first erased. The floating gate is electrically isolated from the control gate by the *interpoly oxide* and from substrate by the *tunnel oxide*, which prevents charge leakage. Thus, Flash is a non-volatile technology, and its retention time is typically above ten years. Because the charge is stable and can be controlled, the cell can encode a number of bits. However, the higher the number of bits per cell, the lower the cell endurance: Each program/erase cycle damages the tunnel oxide of the floating gate, creating defects that trap electrons thus degrading its electrical characteristics and narrowing margins between different cell states. As a result, the programmed and erased states of a worn-out cell cannot be reliably distinguished. The endurance of SLC Flash is about 10^5 program/erase cycles [9], and that of three-bit MLC Flash is about 10^3 cycles [42]. Flash is a mature technology and is of interest in this thesis because it enables high-density storage.

3.1.3 PCM

PCM is a non-charge-based, One Transistor and One Resistor (1T1R), SLC/MLC memory technology. Bits of information are stored as the resistance of the chalcogenide alloy of the cell, as described below. The cell is programmed (set) when the alloy is in the polycrystalline phase (low resistance). This phase is achieved by first melting the alloy by injecting current and then cooling it at a rate slower than the crystal growth rate. If the melted alloy is cooled at a rate faster than the crystal growth rate, it gets locked in the amorphous phase (high resistance), and the cell becomes erased (reset). PCM is a non-volatile technology, but is prone to the resistivity drift effect: The resistance of the alloy in the amorphous phase continuously increases and is highly temperature-dependent. Resistivity drift does not matter for SLC PCM, but complicates the implementation of MLC PCM. However, there are mechanisms for coping with the negative effect of the drift [43, 44], and PCM is commonly used as an MLC technology. Because of thermal expansion and contraction, each set/reset cycle degrades the current injection contacts and the resistance uniformity of the chalcogenide alloy. The endurance of PCM cells is estimated from 10^5 to 10^8 set/reset cycles [10, 45, 46]. PCM is a relatively new technology and is popular among researchers. It is of interest in the scope of this thesis because: 1) it is MLC-capable and 2) it employs different device physics compared to Flash and thus offers significantly different timing and electrical characteristics, as is shown later in this chapter.

3.1.4 Magnetic Memory

One example of magnetic memory is Magnetoresistive Random Access Memory (MRAM): a non-charge-based, One Transistor and One Magnetic Tunnel Junction (1T1MTJ), SLC technology. The Magnetic Tunnel Junction (MTJ) comprises two ferromagnetic layers: one *fixed* (the direction of magnetization is fixed) and one *free* (the direction of magnetization can be changed). One bit of information is stored as the magnetization of the free layer relative to the fixed layer: The MTJ is in the reset state (low resistance) when the two layers are parallel (spin-aligned) and in the set state (high resistance) when the layers are anti-parallel. MRAM is a non-volatile technology and has the endurance of 10^{16} write cycles, primarily defined by the endurance of the access transistor [47]. Spin Transfer Torque Magnetoresistive Random Access Memory (STT-MRAM) is a contemporary implementation of MRAM and uses spin-polarized current to change the magnetic orientation of the free layer [48]. It is a promising technology and attracts vivid interest of memory researchers. However, neither STT-MRAM nor MRAM are explicitly considered further in this thesis, because they do not currently offer higher bit densities than DRAM [47, 49].

Another representative of magnetic memory is the ferromagnetic material covering the platters of HDDs. One bit is stored as the direction of magnetization of a magnetic region on a track of a platter [5]. The direction of magnetization is read and written by an external head. This is a mature technology conventionally employed in the backing store of computing systems.

3.1.5 Other Technologies

The plurality of memory technologies is not limited to those described above. For instance, one NVM technology of active research is Resistive Random Access Memory (RRAM): a non-charge-based, 1T1R, and MLC-capable. Data are stored as the resistance of the metal oxide of the cell. The cell is set (low resistance) when conductive *filaments* (paths) are formed in the metal oxide and reset (high resistance) when the filaments are ruptured.

The endurance of SLC RRAM ranges from 10^6 to 10^{10} set/reset cycles [50, 51], and that of two-bit MLC RRAM is about 10^7 cycles [52].

Another prominent example of an NVM technology is Ferroelectic Random Access Memory (FeRAM): a non-charge-based, 1T1C, SLC memory technology. One bit is represented by the polarization of the thin film of a ferroelectric material placed between two electrodes in a way similar to the structure of a capacitor [53]. FeRAM endurance is estimated from 10^{10} to 10^{13} read/write cycles [54, 55].

Despite the specifics of their basic device physics and operational characteristics, these technologies are not explicitly considered further in this thesis. The term Storage-Class Memory (SCM) is used for collectively denoting such NVM technologies. The characteristics of interest for main memory optimization by increasing its capacity are represented by DRAM, Flash, and PCM.

3.2 Device Organization and Operation

Memory devices are classified as *mechanical* if they contain moving parts, otherwise they are classified as *solid-state*. This section first describes the general organization and operation of a basic solid-state device. Then it illustrates by example implementation details specific to DRAM, PCM, and Flash, and concludes by describing SSDs (compound solid-state devices) and HDDs (mechanical storage devices).

3.2.1 Basic Solid-State Device

Memory cells of a single solid-state device are organized into an array with *word lines* along rows and *bit lines* along columns. Memory arrays may be logically subdivided into smaller units such as *blocks*, *pages*, or *words* that define the granularity of memory array operations. A *die* may contain one or more memory arrays, each having dedicated sensing and buffering circuitry. The sensing circuits are often referred to as *sense amplifiers* and are required for reading data out from the memory array into the buffering circuitry (which may be integrated with the sense amplifiers). The buffering circuitry is further referred to simply as a *buffer*. Memory *array read* latency is the time required for reading data from the array into the buffer, and the memory *array write* latency is the time required for writing data back from the buffer to the array.

The buffer stores data for fast reads and writes (*column accesses*) via an Input/Output (I/O) interface. The number of I/O data pins, denoted by N_{DQ}, matches the *word width* (or *device width*) that is denoted by $\times N_{DQ}$. For instance, an $\times 8$ device has eight data pins. An input *data mask* pin can be used for masking nonvalid write data. The buffer is typically larger than one word and may be accessed at different granularities or *burst sizes*, i.e., one or more words may be transferred between the buffer and the I/O data pins per column access. *Peak data rate* is the product of the device width and the I/O interface speed. It is the maximum theoretical bandwidth of the I/O data interface. An I/O interface may be synchronous, and it is called a Double Data Rate (DDR) interface if it transfers data on both the rising and falling edges of the I/O clock. High-speed data interfaces require for signal integrity bidirectional *data strobe* pins (e.g., one differential pair) that are used for indicating when read and write data must be captured.

On-die control circuitry implements the command protocol and orchestrates internal operations. Data interfaces running at high frequencies require additional circuits, such as Delay-Locked Loop (DLL), and additional operations for signal integrity maintenance [41]. Static power (sometimes referred to as *background* power) is dissipated when the memory device is powered-on and clocked but not in use, i.e., there are no ongoing internal operations. Power-down (low power) modes may be entered by disabling parts of the die, e.g., the clock circuitry, DLL, and memory array periphery such as row and column decoders.

A single memory device may be composed of one or more dies within a package. Memory devices are often organized into *ranks* such that all devices within a rank are identical and operate in lock-step, servicing the same command simultaneously. The peak data rate of a rank is thus the sum of the peak data rates of all the devices composing it.

3.2.2 DRAM Device

Organizational and performance characteristics of an example $\times 8$ DRAM device [8] are presented in the first column of Table 3.1. The DRAM cells are organized into eight arrays (*banks*), each of which is subdivided into pages (*rows*) of 1KB, such that one array contains 16K pages. The array buffer is referred to as a *row buffer*. A DRAM array is ready for a read operation when its bit lines are precharged. DRAM array reads are destructive, i.e., the page data in the array get lost after they have been read out into the row buffer. The operations of writing a page back and precharging the array bit lines for the next read are collectively referred to as a *precharge* operation. The DRAM device in Table 3.1 implements the widely used third generation of the double data rate (DDR3) Synchronous Dynamic Random Access Memory (SDRAM)

Table 3.1: *Characteristics of example DRAM, PCM, NAND and NOR Flash devices*

Characteristic	Units	DRAM	PCM	NAND Flash	NOR Flash
Volatile		yes	no	no	no
Cell endurance	write cycle	10^{16}	10^{8}	10^{5}	10^{5}
Cell capacity	b/cell	1	1–4	1	2
Die capacity	Gb	1	1–4	16	1
Block size	KB	—	Same as for DRAM	512	128
Page size	B	1024		4096	32
Word width	b	8		8	16
Clock frequency	MHz	800		83	52
DDR		yes		yes	no
Peak data rate	Mb/s	$12.8{\cdot}10^{3}$		1328	832
Burst size	B	8		1–4096	2–32
Latencies:					
Array read	ns/page	13.75	48–250	$25{\cdot}10^{3}$	100
Array write	ns/page	13.75	$150–2{\cdot}10^{3}$	$230{\cdot}10^{3}$	$(270–900){\cdot}10^{3}$
Array erase	ms/block	—	—	0.7	800

interface [41], that yields a peak data rate of 12.8Gb/s at a device clock frequency of 800MHz. Such a high-speed interface requires signal integrity maintenance performed by output impedance (ZQ) calibration [41]. DDR3 supports the burst size of eight words, hence the granularity of a column access is eight bytes for the $\times 8$ device considered here. In addition, DDR3 supports the burst size of four words, but it is emulated, i.e., implemented by masking the last four words of the default eight-word burst. DRAM is volatile and requires periodic *refresh* operations that read data from the array into the row buffer and then write the data back.

3.2.3 PCM Device

PCM device organization is commonly assumed to be similar to that of DRAM devices. Thus, the differences between PCM and DRAM devices lie primarily in the characteristics related to the memory cell: endurance, bit capacity, and array read and array write latencies and dynamic energies. Example $\times 8$ PCM device organization and performance characteristics are presented in the second column of Table 3.1. The PCM device can be assumed to have from one to four times the capacity of the DRAM device discussed above because of its MLC capability and a potentially smaller cell area [16]. The device employs the same I/O interface as the DRAM device. PCM reads are non-destructive, and depending on the device implementation the array read latency can be from several times to at least 10× greater than that of DRAM [10, 56]. The PCM array write latency can be from about 10× to 100× greater than that of DRAM, as determined by the time required for setting the cell into the low-resistance state. In this context, the main advantage of PCM over DRAM is its higher bit density.

3.2.4 NAND Flash Device

NAND Flash is one of the flavors of Flash memory. Unlike DRAM and PCM arrays where each cell is directly connected to its bit line, NAND Flash physically organizes cells into *strings* (in series) such that all cells in one string share a single bit line contact [57]. Reducing the number of bit line contacts reduces the area overhead per cell and thus increases the bit density. The complication of NAND Flash is that reading and programming a single cell of a string requires setting all other cells of that string into specific temporary modes that enable modifying the state of the target cell only [58].

Example characteristics of an ×8 SLC NAND Flash device [9] are presented in the third column of Table 3.1. In this example device, cells are organized into two arrays (*planes*) that are subdivided into blocks of 512KB, and the blocks are subdivided into pages of 4KB. There are 128 pages per block and 2K blocks per array, giving each array 256K pages. Each page also contains additional data (224B) for book-keeping purposes, that are excluded from consideration here for ease of page size comparison among technologies. Array read and program operations are performed on the granularity of a page, and array erase operations are performed on the granularity of a block. The NAND Flash page of the example device is four times larger than that of the DRAM device discussed above. The capacity of the NAND Flash device is 16 times greater than that of the DRAM device and is at least four times greater than that of the PCM device. There are MLC NAND Flash devices of even greater capacities in production, but their cell endurance is lower (e.g., $3 \cdot 10^3$ program/erase cycles [42]) than the 10^5 cycles of the example SLC NAND Flash device. The endurance of the example NAND Flash device is three orders of magnitude lower than that of the PCM device discussed above. Memory technologies like RRAM have the promise to improve the endurance of high-density devices with array read and program latencies similar to those of NAND Flash [59]. Thus, the endurance problem of NAND Flash is not of interest in this thesis, and NAND Flash is used as an example NVM technology in terms of bit density and access characteristics.

The NAND Flash I/O data interface (Open NAND Flash Interface (ONFI) [60]) supports double data rate with an approximately $10\times$ lower peak data rate than that of the example DRAM device. The NAND Flash protocol allows reading out an entire page in one burst. In addition, it allows changing the start word (column) and reading out as many words as needed up to the end of the page. The NAND Flash array read latency is at least three orders of magnitude greater than that of DRAM and at least $100\times$ greater than that of PCM. The array write latency is at least four orders of magnitude greater than that of DRAM and at least two orders of magnitude greater than that of PCM.

3.2.5 NOR Flash Device

Another flavor of Flash memory is NOR Flash, that organizes cells into arrays such that each cell is directly connected to its bit line and can be accessed independently [61]. Characteristics of an example $\times 16$ MLC NOR Flash device [62] are presented in the last column of Table 3.1. Despite its MLC capability, the device happens to have the same endurance as the example SLC NAND Flash device, and the same capacity as the example SLC DRAM device. NOR Flash cells are organized into blocks of 128KB, and one block is the granularity of the erase operation. The NOR Flash page is not an organizational unit of the memory array but just a small buffer that supports random accesses at the word granularity. Unlike the devices discussed above, the NOR Flash device is 16-bit wide, because it is a typical width of the parallel NOR Flash interface. The interface is word-addressable and can operate in the synchronous mode with a peak data rate of 832Mb/s, which is of the same order of magnitude but lower than that of the NAND Flash device. The NOR Flash protocol allows reading out an entire page in a burst at the word granularity. The NOR Flash array read latency is about $10\times$ greater than that of DRAM, falls within the range of the PCM array read latencies, and is at least $100\times$ smaller than that of NAND Flash. The NOR Flash array write latency depends on the granularity of operation, from one word (which is comparable to the NAND Flash array write latency) to 512 words (which is about $3.5\times$ greater). The NOR Flash array erase latency is at least three orders of magnitude greater than that of NAND Flash.

NOR and NAND Flash storage arrays do not have to be taken as one with their respective legacy interfaces. Both NOR and NAND Flash can be assumed to employ the DDR3 interface, just like it is assumed for PCM. The greater device capacity of NAND Flash makes it a better representative of slow but bit-dense NVM technologies. Thus, NAND is the Flash flavor of primary interest in this thesis.

3.2.6 SSD

NAND Flash has enabled Solid-State Disks (SSDs). NAND Flash devices inside an SSD are typically organized such that the I/O bandwidth is maximized by exploiting different forms of operation parallelism. SSDs typically implement mechanisms for hiding the long NAND Flash array write latency. In addition, SSDs can cache data, which further improves their access characteristics. A typical I/O interface employed by SSDs is the 6Gb/s Serial Advanced Technology Attachment (SATA) interface.

3.2.7 HDD

Hard Disk Drives (HDDs) are the conventional backing store devices. An HDD comprises a number of magnetic platters (the storage medium) read and written by mechanical heads. Because of the moving parts, HDD access latencies are counted in milliseconds. The long read access latency can be significantly shortened by prefetching data from the storage medium into a cache [5]. Like SSDs, HDDs typically employ the 6Gb/s SATA interface.

3.3 Device-Level Models

This section presents device-level timing, power, and energy models for DRAM, PCM, NAND Flash, SSD, and HDD. The models for DRAM are derived from a device datasheet [8] and Micron's Power Calculator [63–66], that estimates power of a given memory utilization scenario. My models estimate dynamic energy per access and thus provide insights into device energy characteristics. Micron's models do not represent power mode transition overhead and can produce misleading results when power-down modes are modeled [67]. However, it is safe to derive from Micron's models here, because this thesis considers high-performance systems that do not use power-down modes. The models for PCM and NAND Flash adapt the DRAM models by sharing the DDR3 interface and by adding technology-specific modifications. For the purposes of this thesis, I model the following accesses:

1. A read access with one or more column reads that:
    a) misses in the buffer (requires an array read), denoted by $read\,miss$;
    b) hits in the buffer (the column read(s) are served immediately), denoted by $read\,hit$;

Table 3.2: *Subset of DDR3 SDRAM commands*

Symbol	Description	Symbol	Description
ACT	Activate command (array read)	PRE	Precharge command (array write)
RD	Read command (column read)	REF	Refresh command
WR	Write command (column write)	ZQCS	ZQ Calibration Short command

2. A write access with one or more column writes that:

   a) misses in the buffer, denoted by $write\,miss$;

   b) hits in the buffer, denoted by $write\,hit$.

Accesses 1.a and 2.a are modeled for DRAM, PCM, and NAND Flash, but accesses 1.b and 2.b are modeled only for DRAM that satisfies the needs of Chapter 5.

3.3.1 DRAM Models

An SDRAM device dissipates static (background) power when it is powered-on, clocked (ready to receive commands), but is idle (not executing any operations). The device is in the *precharge standby* mode, if all of its banks are closed, i.e., the row buffers contain no data. If any of the banks is open, i.e., at least one row buffer contains data, the device is in the *active standby* mode. The precharge standby mode is denoted by PRE_STBY and the active standby mode by ACT_STBY. The device transitions from PRE_STBY to ACT_STBY when it receives the *activate* command, denoted by ACT, and back from ACT_STBY to PRE_STBY when it receives the *precharge* command, denoted by PRE. ACT performs an array read, and PRE an array write. When the device is in ACT_STBY, it is ready to receive column access commands, denoted by RD for reads and by WR for writes. The device requires array data integrity maintenance, implemented by the refresh command (REF), and data interface maintenance, implemented by the *ZQ Calibration Short* command (ZQCS). The above subset of the device modes and DDR3 commands is sufficient for the purposes of this thesis, since it represents the specifics of device operation, and power-down modes are of no interest. The commands are summarized in Table 3.2. The device draws a certain amount of current in each of the modes and when executing the commands. The amounts of energy expended statically when the device is idle and dynamically when the device is executing a command are calculated using timing and electrical characteristics tabulated in the device datasheet [8].

Table 3.3: *Selected DRAM characteristics*

Symbol	Value	Units	Description
V	1.5	V	Supply voltage
N_{DQ}	8	–	Number of data pins (device (word) width in bits)
N_{DQS}	2	–	Number of strobe pins
N_{DM}	1	–	Number of data mask pins
BL	8	word	Burst length
P_{size}	1	KB	Page size
$P_{DQ(R)}$	1.1	mW	Power per pin when driving output [65]
$P_{DQ(W)}$	8.2	mW	Power per pin when terminating a write [65]
t_{CLK}	1.25	ns	Clock period
t_{RC}	48.75	ns	ACT to ACT delay used for I_{DD0} measurement
t_{RAS}	35	ns	ACT to PRE delay used for I_{DD0} measurement
t_{RCD}	13.75	ns	ACT to RD or WR (array read to column access) delay
RL	13.75	ns	Column read latency (RD to first word)
WL	13.75	ns	Column write latency (WR to first word)
t_{CCD}	5	ns	Column to column delay (RD to RD or WR to WR)
t_{RTP}	7.5	ns	Column read to precharge delay (RD to PRE)
t_{WR}	15	ns	Write recovery latency (last word to PRE)
t_{RP}	13.75	ns	PRE (row precharge) latency
$t_{RFC(MIN)}$	110	ns	REF period used for I_{DD5B} measurement
t_{REFI}	7.8125	μs	Interval between REF in normal conditions (0-85C°)
t_{ZQCS}	80	ns	ZQCS latency
t_{ZQI}	152	ms	Interval between ZQCS [68]

Table 3.4: *Currents of DRAM revisions F and G*

Symbol	Rev. F	Rev. G	Units	Description
I_{DD0}	120	70	mA	ACT current
I_{DD2N}	70	45	mA	PRE_STBY current
I_{DD3N}	67	45	mA	ACT_STBY current
I_{DD4R}	250	140	mA	RD current
I_{DD4W}	250	145	mA	WR current
I_{DD5B}	260	170	mA	REF current

The datasheet provides numbers for two different versions of the device labeled *revision F* and *revision G*. The revisions share same timing characteristics presented in Table 3.3, but revision F has roughly two times worse electrical characteristics than revision G, as is shown by the current values for the two revisions in Table 3.4. I_{DD0}

is the average current that the device draws during a measurement loop when the memory controller issues ACT each t_{RC} followed by PRE after t_{RAS}. The device draws the I_{DD2N} current when it is in the PRE_STBY mode and I_{DD3N} when it is in the ACT_STBY mode. These two currents are equal for revision G, but slightly differ for revision F (it is unknown why I_{DD3N} is less than I_{DD2N} for revision F). The device draws the I_{DD4R} and I_{DD4W} currents when performing column reads and writes, respectively. I_{DD5B} is the average current that the device draws during a measurement loop when the memory controller issues REF each $t_{RFC(MIN)}$. Each of the currents is the total current drawn by the device (e.g., I_{DD4R} includes I_{DD3N}). Since I_{DD0} and I_{DD5B} are measured for specific scenarios (defined by the measurement loops), their values have to be scaled to the actual scenarios. For instance, the I_{DD5B} value have to be scaled according to the actual interval between the REF commands, that is t_{REFI}.

A DRAM $read\,miss$ first incurs the latency of an array read (t_{RCD}). Next, a column read command (RD) incurs the delay to the first valid word on the data pins (RL) and the data transfer latency of $T_{burst} = BL\cdot t_{CLK}/2$. The delay between two consecutive RD commands is t_{CCD} (that is also equal to T_{burst}) and the RL delays of two consecutive column reads overlap. The delay between the last RD command and PRE (t_{RTP}) is followed by the latency of PRE (t_{RP}). The RL delay of the first column read and T_{burst} of all the data transfers overlap with t_{CCD} of the remaining column reads and the concluding t_{RTP} and t_{RP}. Thus, the latency of a DRAM $read\,miss$ with N_{CA} column reads is given by:

$$\underset{read\,miss}{T_{DRAM}} = t_{RCD} + (N_{CA} - 1)\cdot t_{CCD} + t_{RTP} + t_{RP} \tag{3.1}$$

A DRAM $write\,miss$ incurs t_{RCD}, followed by WL from the first column write (WR) to the first input word on the data pins, and then the data transfer latency (T_{burst}). The delay between two consecutive WR commands is t_{CCD}, and the WL delays of two consecutive WR commands overlap. The delay between the last word on the data pins and PRE (t_{WR}) is followed by t_{RP}. Thus, the latency of a DRAM $write\,miss$ is given by:

$$\underset{write\,miss}{T_{DRAM}} = t_{RCD} + WL + T_{data} + t_{WR} + t_{RP} \tag{3.2}$$

where T_{data} is the data transfer time of N_{CA} column accesses given by:

$$T_{data} = N_{CA}\cdot T_{burst} = N_{CA}\cdot BL\cdot t_{CLK}/2 \tag{3.3}$$

The latencies of the $read\, hit$ and $write\, hit$ accesses are modeled simply as the respective first-word delays followed by the data transfer latencies:

$$T_{\substack{DRAM\\read\, hit}} = RL + T_{data} \tag{3.4}$$

$$T_{\substack{DRAM\\write\, hit}} = WL + T_{data} \tag{3.5}$$

where T_{data} is given by (3.3). The $read\, hit$ and $write\, hit$ accesses coalesce with the $read\, miss$ and $write\, miss$ accesses.

When devices are organized into a rank, the access latencies of each device overlap. For instance, if a rank comprises eight $\times 8$ devices and $BL = 8$, the latency of reading 64B out from the rank is defined by (3.1), and each device performs just one column read ($N_{CA} = 1$) providing 8B.

Static (background) energy expended in the PRE_STBY mode is a constant component of the total device energy and is calculated by multiplying execution time and static power P_{PRE_STBY} given by:

$$P_{PRE\_STBY} = I_{DD2N} \cdot V \tag{3.6}$$

Dynamic energy is calculated per device operation as additional energy on top of static energy. The energy expended in the ACT_STBY mode is counted as dynamic energy, too, according to (3.7) for reads and (3.8) for writes:

$$E_{\substack{DRAM\\ACT\_STBY(read\, miss)}} = P_{DD3N} \cdot \left(T_{\substack{DRAM\\read\, miss}} - t_{RP} \right) \tag{3.7}$$

$$E_{\substack{DRAM\\ACT\_STBY(write\, miss)}} = P_{DD3N} \cdot \left(T_{\substack{DRAM\\write\, miss}} - t_{RP} \right) \tag{3.8}$$

where $P_{DD3N} = (I_{DD3N} - I_{DD2N}) \cdot V$, $T_{\substack{DRAM\\read\, miss}}$ is given by (3.1), and $T_{\substack{DRAM\\write\, miss}}$ is given by (3.2). I_{DD2N} is subtracted from I_{DD3N}, since the latter includes the former, and I want to separate dynamic energy from static energy. However, for the current values in Table 3.4 this dynamic energy is either small compared to static energy (for revision F) or zero (for revision G). I assume that I_{DD3N} equals I_{DD2N} for both DRAM revisions, thus eliminating the ACT_STBY dynamic energy and simplifying the models.

The ACT current (I_{DD0}) is measured as the current drawn by the device during a specific loop of ACT and PRE commands: The delay between ACT and PRE is t_{RAS}, and the delay between two consecutive ACT commands is t_{RC}. The device is in the ACT_STBY mode during t_{RAS} and in the PRE_STBY mode otherwise. In order to separate the current drawn by executing the ACT command, the tabulated ACT_STBY current (I_{DD3N}) is subtracted from I_{DD0} during t_{RAS}, and the PRE_STBY current (I_{DD2N}) is subtracted from I_{DD0} during $(t_{RC} - t_{RAS})$:

$$I'_{DD0} = I_{DD0} - \frac{1}{t_{RC}} \cdot (I_{DD3N} \cdot t_{RAS} + I_{DD2N} \cdot (t_{RC} - t_{RAS})) \tag{3.9}$$

I'_{DD0} is the average current drawn by the device when performing one ACT and one PRE within t_{RC}. Thus, the dynamic energy of performing one ACT and one PRE is given by:

$$E_{DD0} = I'_{DD0} \cdot V \cdot t_{RC} \tag{3.10}$$

Lee et al. [10] estimate that 75% of that energy are expended by executing ACT, and the remaining 25% are expended by executing PRE, hence:

$$E_{ACT} = 0.75 \cdot E_{DD0} \tag{3.11}$$

$$E_{PRE} = 0.25 \cdot E_{DD0} \tag{3.12}$$

The dynamic energy of reading data from the device (performing read column accesses) is calculated using the RD current (I_{DD4R}) separated from the ACT_STBY current (I_{DD3N}) during T_{data} given by (3.3):

$$E_{RD} = (I_{DD4R} - I_{DD3N}) \cdot V \cdot T_{data} \tag{3.13}$$

In addition, reading data out from the device expends the dynamic energy of driving the data and strobe pins:

$$E_{DQ} = P_{DQ(R)} \cdot (N_{DQ} + N_{DQS}) \cdot T_{data} \tag{3.14}$$

The dynamic energy of writing data to the device (performing write column accesses) is calculated similarly to (3.13):

$$E_{WR} = (I_{DD4W} - I_{DD3N}) \cdot V \cdot T_{data} \tag{3.15}$$

with additional energy expended for write termination at the data, strobe, and mask pins:

$$E_{term} = P_{DQ(W)} \cdot (N_{DQ} + N_{DQS} + N_{DM}) \cdot T_{data} \tag{3.16}$$

Thus, the dynamic energies of the $read\, miss$ and $write\, miss$ accesses are defined by:

$$E_{\underset{read\, miss}{DRAM}} = E_{DD0} + E_{RD} + E_{DQ} \tag{3.17}$$

$$E_{\underset{write\, miss}{DRAM}} = E_{DD0} + E_{WR} + E_{term} \tag{3.18}$$

where E_{DD0} is given by (3.10), E_{RD} by (3.13), E_{DQ} by (3.14), E_{WR} by (3.15), and E_{term} by (3.16).

The dynamic energies of the $read\, hit$ and $write\, hit$ accesses are modeled simply as the dynamic energies of column accesses:

$$E_{\underset{read\, hit}{DRAM}} = E_{RD} + E_{DQ} \tag{3.19}$$

$$E_{\underset{write\, hit}{DRAM}} = E_{WR} + E_{term} \tag{3.20}$$

where E_{RD} is given by (3.13), E_{DQ} by (3.14), E_{WR} by (3.15), and E_{term} by (3.16).

When devices are organized into a rank, the access dynamic energies of each device are added together. For instance, if a rank comprises eight $\times 8$ devices and $BL = 8$, the energy of reading out 64B is defined by $8 \cdot E_{\underset{read\, miss}{DRAM}}$ given by (3.17), and each device performs one column read ($N_{CA} = 1$).

The tabulated REF current (I_{DD5B}) is measured in a specific loop of REF commands when one REF is issued each $t_{RFC(MIN)}$. Thus, in order to calculate the REF power of a typical scenario (one REF per t_{REFI}), the I_{DD5B} value is scaled:

$$P_{REF} = (I_{DD5B} - I_{DD2N}) \cdot V \cdot t_{RFC(MIN)} / t_{REFI} \tag{3.21}$$

and the REF energy is calculated for entire execution time.

Table 3.5: *PCM array latencies and energies normalized to those of DRAM*

Operation	Norm. Latency	Norm. Energy
Array read	4.4	2.1
Array write	12.0	43.1

3.3.2 PCM Models

Models similar to those for DRAM are employed for calculating access latencies and dynamic energies of PCM devices, assuming that they implement the DDR3 interface. The operation of PCM differs from DRAM in a number of ways, e.g.: 1) PCM does not require refresh, since it is non-volatile; 2) PCM array writes are required only if data in the buffer are modified (dirty), since PCM array reads are non-destructive; 3) PCM array writes are implemented such that only modified words are written into the array (*partial writes* [10]). These differences are implemented as optimizations of DDR3 for PCM.

The latencies and dynamic energies of the PCM array read and array write are derived by scaling those of DRAM using the ratios estimated by Lee et al. [10] and shown in Table 3.5. The latency of a PCM $read\,miss$ access is calculated similarly to that of DRAM. Since the access does not modify data, PRE is not required, and thus t_{RTP} and t_{RP} are not incurred:

$$\underset{read\,miss}{T_{PCM}} = t^{'}_{RCD} + RL + T_{data} \tag{3.22}$$

where $t^{'}_{RCD} = 4.4{\cdot}\, t_{RCD}$ as per Table 3.5, RL is provided in Table 3.3, and T_{data} is given by (3.3). The latency of a PCM $write\,miss$ access is calculated using the same equation as that for DRAM, but with scaled array read and write latencies:

$$\underset{write\,miss}{T_{PCM}} = t^{'}_{RCD} + WL + T_{data} + t_{WR} + t^{'}_{RP} \tag{3.23}$$

where $t^{'}_{RCD} = 4.4{\cdot}\, t_{RCD}$, $t^{'}_{RP} = 12{\cdot}\, t_{RP}$ as per Table 3.5, WL and t_{WR} are provided in Table 3.3, and T_{data} is given by (3.3).

Static energy expended by a PCM device in the PRE_STBY mode is the same as that of DRAM and is given by (3.6). The dynamic energy of the ACT_STBY mode is zero, because I_{DD3N} and I_{DD2N} are assumed to be equal. The dynamic energies of the PCM array read and write are scaled according to Table 3.5 and are given by (3.24) and (3.25):

$$E'_{ACT} = 2.1 \cdot 0.75 \cdot E_{DD0} \tag{3.24}$$
$$E'_{PRE} = 43.1 \cdot 0.25 \cdot E_{DD0} \tag{3.25}$$

where E_{DD0} is given by (3.10). Partial writes are modeled at the granularity of one burst of BL words by scaling E'_{PRE} linearly, according to the number of dirty words. In case of a write access with N_{CA} column writes, $N_{CA} \cdot BL$ words are written to the array. Given that an array page contains P_{size} bits and the word width is N_{DQ} bits, the scaling factor is:

$$\alpha = \frac{N_{CA} \cdot BL}{P_{size}/N_{DQ}} \tag{3.26}$$

Thus, the dynamic energies of the PCM $read\, miss$ and $write\, miss$ accesses are given by (3.27) and (3.28), respectively:

$$\underset{read\, miss}{E_{PCM}} = E'_{ACT} + E_{RD} + E_{DQ} \tag{3.27}$$
$$\underset{write\, miss}{E_{PCM}} = E'_{ACT} + E_{WR} + E_{term} + \alpha \cdot E'_{PRE} \tag{3.28}$$

where E'_{ACT} is given by (3.24), E_{RD} by (3.13), E_{DQ} by (3.14), E_{WR} by (3.15), E_{term} by (3.16), E'_{PRE} by (3.25), and α is calculated using (3.26) for the number of column writes (N_{CA}) that the access performs.

3.3.3 NAND Flash Models

The access latencies and dynamic energies of NAND Flash can be calculated using models similar to those for DRAM. The native interface of NAND Flash is ONFI [60], that is a relatively slow DDR interface (e.g., 83MHz [9]) where signal lines are shared among command, address, and data. For instance, a read access comprises one command cycle followed by five address cycles, then one more command cycle initiating an array read, followed by data cycles when valid data become available in the buffer [69]. For the purposes of this thesis, I adapt the DDR3 interface to work with NAND Flash arrays. The operation of NAND Flash differs from DRAM in a number of ways, for instance: 1) NAND Flash does not require refresh, since it is non-volatile; 2) NAND Flash array

Table 3.6: *Selected NAND Flash characteristics*

Symbol	Value	Units	Description
V_{CC}	3.3	V	Supply voltage
V_{CCQ}	1.8	V	I/O supply voltage
P_{size}	4096	B	Page size
B_{size}	128	page	Block size
t_R	25	μs	R latency
t_{PROG}	230	μs	PROG latency
t_{BERS}	700	μs	BERS latency
I_{CC1_S}	20	mA	R current
I_{CC2_S}	20	mA	PROG current
I_{CC3_S}	20	mA	BERS current
I_{CC5_S}	5	mA	Bus idle current
I_{SB}	10	μA	Standby current V_{CC}
I_{SBQ}	3	μA	Standby current V_{CCQ}

writes are required only if data in the buffer are dirty, since NAND Flash array reads are non-destructive; 3) NAND Flash array writes can only be performed to an erased array page (this is the *erase-before-write* overhead of Flash).

ONFI denotes the array read operation by R, the array write operation by PROG, and the array erase operation by BERS. The currents drawn by a NAND Flash device executing these operations are tabulated in the datasheet [9]. The tabulated currents include the current drawn by the interface circuits of the device. NAND Flash characteristics of interest are summarized in Table 3.6.

Each array write requires an erased page to be available in the array. The erase operation is performed at the granularity of a block of B_{size} pages. Thus, the device must perform one array erase per B_{size} array writes. The erase-before-write latency overhead can be simplistically accounted for by distributing the array erase latency over B_{size} array writes:

$$t'_{PROG} = \frac{t_{BERS}}{B_{size}} + t_{PROG} \tag{3.29}$$

Assuming the specifics of NAND Flash operation listed above, the latencies of the $read\,miss$ and $write\,miss$ accesses are respectively given by:

$$T_{\underset{read\,miss}{NAND}} = t_R + RL + T_{data} \tag{3.30}$$

$$T_{\underset{write\,miss}{NAND}} = t_R + WL + T_{data} + t_{WR} + t'_{PROG} \tag{3.31}$$

where t_R is provided in Table 3.6, t'_{PROG} is given by (3.29), latencies RL, WL, and t_{WR} are provided in Table 3.3, and T_{data} is given by (3.3).

The static energy of the PRE_STBY mode is the same as that of DRAM and PCM, and the dynamic energy of the ACT_STBY mode is assumed equal to zero for the same reason as for DRAM and PCM. The dynamic energies of the array read (R), write (PROG), and erase (BERS) are given by (3.32), (3.33), and (3.34), respectively:

$$E_R = (I_{CC1\_S} - I_{CC5\_S}) \cdot V_{CC} \cdot t_R \tag{3.32}$$

$$E_{PROG} = (I_{CC2\_S} - I_{CC5\_S}) \cdot V_{CC} \cdot t_{PROG} \tag{3.33}$$

$$E_{BERS} = (I_{CC3\_S} - I_{CC5\_S}) \cdot V_{CC} \cdot t_{BERS} \tag{3.34}$$

where the currents, V_{CC}, and latencies are provided in Table 3.6. The standby currents I_{SB} and I_{SBQ} are three orders of magnitude smaller than I_{CC5_S}, so I omit them from calculations. The erase-before-write energy overhead is simplistically accounted for by distributing array erase energy E_{BERS} over B_{size} array writes:

$$E'_{PROG} = \frac{E_{BERS}}{B_{size}} + E_{PROG} \tag{3.35}$$

Thus, the dynamic energies of the NAND Flash $read\,miss$ and $write\,miss$ accesses are respectively given by (3.36) and (3.37):

$$E_{\underset{read\,miss}{NAND}} = E_R + E_{RD} + E_{DQ} \tag{3.36}$$

$$E_{\underset{write\,miss}{NAND}} = E_R + E_{WR} + E_{term} + E'_{PROG} \tag{3.37}$$

where E_R is given by (3.32), E_{RD} by (3.13), E_{DQ} by (3.14), E_{WR} by (3.15), E_{term} by (3.16), and E'_{PROG} by (3.35).

Table 3.7: *Selected characteristics of SSD and HDD*

	t_{RD}	$t_{RD\,hit}$	t_{WR}	P_{RD}	P_{WR}	P_{idle}
SSD	50μs	30μs	50μs	0.51W	1.05W	30mW
HDD	6.78ms	150μs	3.19ms	4.2W	4.8W	3W

3.3.4 SSD and HDD Models

I characterize disk by its interface bandwidth, random read and write access latencies, the latency of a read access hitting in the disk cache, idle (static) power, read power, and write power. SSD and HDD share the same simple model but with different input numbers, that come from product measurements [11, 12] and are summarized in Table 3.7. I aggregate the smallest measured values across all the products, with the exception that I set the random access latency of SSD to 50μs and its cache hit latency to 30μs. The cache hit latency of HDD is set to 150μs. Both SSD and HDD employ the 6Gb/s SATA interface.

The latencies of a random read, a read hitting in the disk cache, and a random write are given by (3.38), (3.39), and (3.40), respectively:

$$T_{\substack{disk\\read}} = t_{RD} + T_{xfer} \tag{3.38}$$

$$T_{\substack{disk\\read\,hit}} = t_{RD\,hit} + T_{xfer} \tag{3.39}$$

$$T_{\substack{disk\\write}} = T_{xfer} + t_{WR} \tag{3.40}$$

where t_{RD} is the latency of an internal read, $T_{xfer} = \frac{\#bytes\ to\ transfer}{interface\ bandwidth}$ is the interface data transfer latency, $t_{RD\,hit}$ is the latency of an internal read hitting in the disk cache, and t_{WR} is the latency of an internal write.

The dynamic energies of a random read, a read hitting in the disk cache, and a random write are given by (3.41), (3.42), and (3.43), respectively:

$$E_{\substack{disk\\read}} = (P_{RD} - P_{idle}) \cdot T_{\substack{disk\\read}} \tag{3.41}$$

$$E_{\substack{disk\\read\,hit}} = (P_{RD} - P_{idle}) \cdot T_{\substack{disk\\read\,hit}} \tag{3.42}$$

$$E_{\substack{disk\\write}} = (P_{WR} - P_{idle}) \cdot T_{\substack{disk\\write}} \tag{3.43}$$

where P_{RD} is the read power, P_{idle} is the idle power, P_{WR} is the write power and $T_{\substack{disk\\read}}$, $T_{\substack{disk\\read\,hit}}$, and $T_{\substack{disk\\write}}$ are given by (3.38), (3.39), and (3.40), respectively.

Table 3.8: *64B access latencies of DRAM, PCM, and NAND Flash*

DRAM, PCM, and NAND Flash latencies are per rank of eight devices
Ratios (×) normalized to respective DRAM latencies

	$read\,miss$	$write\,miss$
DRAM	35ns	61ns
PCM	2.26×	4.23×
NAND Flash	715×	4253×

Table 3.9: *4KB access latencies of DRAM, PCM, NAND Flash, SSD, and HDD*

DRAM, PCM, and NAND Flash latencies are per rank of eight devices
Ratios (×) normalized to respective DRAM latencies

	$read\,miss$	$write\,miss$
DRAM	350ns	376ns
PCM	1.13×	1.53×
NAND Flash	72×	693×
SSD	157×	146×
HDD	19386×	8492×

3.3.5 Example Estimates

Using the models described earlier in this section, I obtain example estimates of access latencies and access dynamic energies for DRAM, PCM, NAND Flash, SSD, and HDD. The input characteristics of the technologies are provided in Tables 3.3 to 3.7. I consider DRAM revision G as the state-of-the-art DRAM in this thesis. DRAM and PCM have equal page sizes, and the page size of NAND Flash (4KB) is scaled down four times to match that of DRAM (1KB). This scaling decreases four times the NAND Flash E_R and E'_{PROG} energies, given by (3.32) and (3.35), respectively. I organize DRAM, PCM and NAND Flash into ranks of eight $\times 8$ devices. The access sizes of interest are 64B and 4KB per rank. SSD and HDD access latencies and dynamic energies are estimated only for 4KB accesses, since disk is typically not accessed at smaller granularities.

Table 3.8 shows the DRAM, PCM, and NAND Flash latencies of the 64B $read\,miss$ and $write\,miss$ accesses (accesses that require array reads) to the rank of eight $\times 8$ devices. PCM is slower than DRAM by a factor of several times: by 2.26× for 64B reads and by 4.23× for 64B writes. NAND Flash is slower than DRAM by several orders of magnitude: by 715× for 64B reads and by 4253× for 64B writes.

Table 3.10: *64B access dynamic energies of DRAM, PCM, and NAND Flash*

DRAM, PCM, and NAND Flash dynamic energies are per rank of eight devices
Ratios (×) normalized to respective dynamic energies of DRAM revision G

	$read\,miss$	$write\,miss$
DRAM	21nJ	24nJ
PCM	1.41×	1.40×
NAND Flash	120×	1065×

Table 3.11: *4KB access dynamic energies of DRAM, PCM, NAND Flash, SSD, and HDD*

DRAM, PCM, and NAND Flash dynamic energies are per rank of eight devices
Ratios (×) normalized to respective dynamic energies of DRAM revision G

	$read\,miss$	$write\,miss$
DRAM	408nJ	630nJ
PCM	1.02×	1.14×
NAND Flash	7.0×	42×
SSD	65×	89×
HDD	19976×	9136×

Table 3.9 shows the DRAM, PCM, and NAND Flash latencies of the 4KB $read\,miss$ and $write\,miss$ accesses to the rank of eight ×8 devices (each device executes 64 column accesses), and the SSD and HDD latencies of 4KB reads and writes that miss in the disk cache. The large access size amortizes the differences in the array read and write latencies among the technologies: PCM is slower than DRAM only by 1.13× for reads and by 1.53× for writes, and NAND Flash is slower than DRAM by 72× for reads (instead of 715× in case of the short 64B access) and by 693× for writes (instead of 4253×). The access latency gap between DRAM and SSD is about 150× for disk reads and writes that miss in the disk cache. The SSD cache reduces the gap for reads down to about 100× (not shown in Table 3.9). The access latency gap between DRAM and HDD is close to four orders of magnitude for disk writes and above four orders of magnitude for disk reads that miss in the disk cache. HDD cache dramatically reduces the gap for disk reads down to about 443× (not shown in Table 3.9).

The estimates of access dynamic energies in Tables 3.10 and 3.11 follow the same trend as the estimates of access latencies discussed above. SSD and HDD normalized dynamic energies of reads hitting in the disk cache are 41× and 457×, respectively (not shown in Table 3.11).

3.3.6 Discussion

There exist a number of models for estimating timing and electrical characteristics of different memory technologies [7, 63, 70–72]. I choose to develop custom models, specific to the purposes of this thesis, that can be used for obtaining characteristics of devices employing different memory technologies but using the same interface type. The energy and power estimates represent the worst case, since datasheets tabulate worst-case measurements of currents. This is acceptable for the purposes of this thesis, because the access dynamic energy gaps between the memory technologies are preserved.

3.4 Summary

Memory technologies differ by their basic cell properties, device organization, and device operation. The primary property of interest in this thesis is bit density, because it enables increasing memory capacity. Technologies like PCM and NAND Flash offer $4\times$ and $16\times$ higher bit densities than DRAM, respectively. It is assumed that bit density does not depend on the I/O interface of a memory device, thus for simplicity a single high-speed interface, DDR3, is employed for DRAM, PCM, and NAND Flash. Technologies with higher bit densities than DRAM typically suffer from worse performance and energy efficiency, but they are still faster and more energy efficient than disk, as is shown by Tables 3.9 and 3.11. The access latency and dynamic energy gaps between PCM and DRAM are small compared to those between NAND Flash and DRAM, but PCM has a four times lower bit density than NAND Flash. DRAM, PCM, and NAND Flash are the representative technologies used later in this thesis for composing hybrid main memories.

4 Hybrid Memory Systems Taxonomy

In the quest for higher-performance, more energy efficient memory systems, unconventional memory systems have been investigated. The baseline is a multi-core, high-performance computing system with a cache hierarchy and on-chip memory controllers, main memory, and a disk system accessible via I/O (Figure 4.1). Main memory is built from conventional DRAM devices connected to the memory controllers via memory channels. Main memory is *hybrid* if it is divided into two or more partitions where each partition is optimized for specific purposes, e.g., performance, energy efficiency, capacity, or cost. Today's diversity of memory technologies, e.g., DRAM, PCM, and NAND Flash, has created opportunities for research and implementation of hybrid main memory systems. As a result, a large body of work on hybrid systems has been created. Unfortunately, it lacks systematization, and this complicates positioning new proposals within the existing body of work. Due to the development of memory technologies, inspiring results of contemporary work, and the continuous increase in demand for higher memory system performance, energy efficiency, and capacity, the field of hybrid main

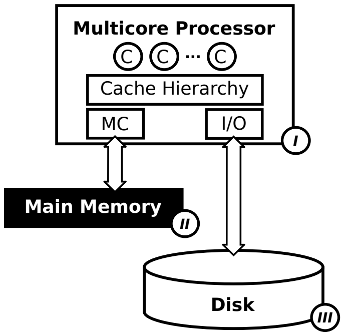


Figure 4.1: *Baseline system*

memory systems is expected to continue expanding. In this chapter, I survey a body of work on hybrid main memory systems not older than ten years and classify systems by organization in accordance with a novel taxonomy.

4.1 Notation

Various hybrid main memory system organizations have been proposed in the literature. I classify these by their *logical* and *physical* organization. A hybrid system is logically divided into partitions. Logical partitions can be located on the same or on different physical levels, explained below.

I classify hybrid systems as *flat* or *hierarchical* by their logical topology. In the former, each partition can be accessed independently. In the latter, lower partitions can be accessed via higher partitions in much the same way as levels in a conventional cache hierarchy. For clarity in describing the next taxonomy category, additional terms are introduced. A *data migration* is a data move between two partitions, and a *data replication* is a data copy. A *data promotion* is a data migration or replication to a partition with higher performance in terms of access latency or energy efficiency in terms of access dynamic energy. A *migrating* hybrid system migrates data on promotion, and a *replicating* one replicates. In the context of hierarchical organizations, migrating systems are exclusive and replicating systems are inclusive. I identify three *physical levels* by the distance from the processor(s): 1) on-chip (inside the processor package), 2) off-chip but before I/O, and 3) after I/O. The physical levels are labeled with ***I***, ***II***, and ***III*** in

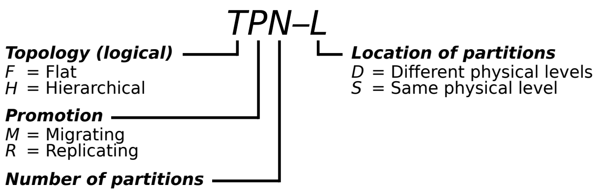


Figure 4.2: *TPN–L notation*

Figure 4.1, respectively. With respect to the physical organization, I distinguish between hybrid systems that have logical partitions on the same or on different levels.

To categorize prior work in the field of hybrid systems, I introduce a *TPN–L* notation, explained in Figure 4.2. In addition, I describe specifics of each organization, e.g., memory technologies used for building partitions from, or if a partition of a hierarchical system can be bypassed for specific types of accesses. Evaluation results provided here should be taken within the context of the corresponding work. The reader can refer to the original publications to clarify authors' assumptions, evaluation approaches, and results.

4.2 Classification

According to the *TPN–L* notation, hybrid systems are categorized into flat and hierarchical on the top level. Section 4.2.1 describes the diversity of flat system organizations, and Section 4.2.2 the diversity of hierarchical ones.

4.2.1 Flat Systems

I first describe an example of a flat hybrid system with partitions on different physical levels. According to the *TPN–L* notation, Dong et al. [27] propose an *FM2–D* system (flat, migrating, with two partitions of different physical levels), where one partition is built from custom DRAM and the other from conventional DRAM. The authors bridge the processor/memory performance gap by augmenting conventional DRAM main memory with customized high-performance DRAM placed inside the processor package. They consider on-package DRAM capacities up to 1GB because of limited power supply and heat dissipation capabilities. They conclude that using the on-package DRAM as Last Level Cache (LLC) is not worthwhile because of the small difference between the hit latency and miss penalty and the negligible reduction of the cache miss rate compared to a smaller LLC. Their hybrid system maps statically the first gigabyte of program memory addresses to the on-package DRAM. When the program's working set fits completely in the on-package DRAM, the system may offer better performance than conventional main memory with DRAM LLC of the same capacity as the on-package DRAM partition. The authors investigate coarse-grain pure-hardware and fine-grain OS-assisted dynamic data mapping schemes, too. Their main conclusions are: 1) data migration granularity should be adaptive to different workload types, and 2) off-package memory access traffic reduction of 83% can be achieved on average for workloads used in their study. Their hybrid system organization with dynamic data mapping suffers a power overhead from two to 62$\times$ compared to conventional DRAM main memory. The authors do not evaluate the effect of increasing main memory capacity on the number of disk accesses.

Phadke and Narayanasamy [23] bridge the processor/memory performance gap with an *FM3–S* system, where one partition each is built from DRAM optimized either for latency, bandwidth, or power. Unlike Dong et al. [27], they reduce power consumption. The authors classify programs at design-time into three groups based on memory access behavior: latency-sensitive, bandwidth-sensitive, or latency/bandwidth-insensitive (data can be mapped to the low-power partition without hurting performance). They implement in the OS a dynamic data mapping scheme that matches the program type with available memory resources. Compared to conventional DRAM main memory of the same capacity, their hybrid memory improves performance by 13.5% and reduces memory power by 20%, but increases the number of disk accesses by 1%.

Unlike Dong et al. [27] and Phadke and Narayanasamy [23], Sudan et al. [19] bridge the memory/disk performance gap with their *FM2–S* system, where both partitions are built from DRAM, but one partition is aggressively power-managed (at the granularity of a rank) and the other is not. The partitions are labeled *cold* and *hot* tiers, respectively. The power dissipation of one hot rank is the same as six cold ranks, thus main memory capacity can be increased within a fixed power budget by making some hot ranks cold. For workloads with working set sizes exceeding baseline main memory capacity, this reduces the number of disk accesses and consequently improves system performance, although the average main memory access latency increases. Employing the same DRAM devices in both tiers allows for dynamic resizing of the hot tier depending on an actual workload working set size, so that the average main memory access latency is not penalized when extra capacity is not needed. Compared to a system where all ranks are hot, their system supports up to three times more DRAM within the same power budget. Despite an average main memory latency increase, the system performance improves by 250%.

Similarly to Sudan et al. [19], Mogul et al. [20] increase the main memory capacity, but with a different memory technology. Their *FM2–S* system has one DRAM and one large NOR Flash partitions. The authors adaptively map data accessed mostly for reads into the NOR Flash partition, observing that the fraction of such data is large enough in server programs.

Unlike Mogul et al. [20], Zhang and Li [28] build the second partition of their *FM2–S* system from PCM. Both partitions are 3D-stacked on top of the processor die. They use 2GB PCM as main memory and the DRAM partition as a 128MB OS-managed buffer in front of it. Data are initially mapped to PCM, and write-intensive data are adaptively promoted. Moving main memory inside the processor package improves the average main memory access latency and bandwidth, but increases the memory/disk performance gap. Compared to a baseline system with DRAM-only main memory inside the processor package, their hybrid system of the same capacity achieves power savings of 54% with a performance degradation of 6%. The power savings reduce thermal constraints of the processor package and thus allow for an average speedup of 7%. Assuming Multi-Level Cell (MLC) PCM with two bits per cell, PCM capacity doubles (4GB), and this reduces the average main memory access latency by 15% compared to the baseline DRAM-only system half the capacity (2GB).

Similarly to Zhang and Li [28], Ramos et al. [18] build the second partition of their *FM2–S* system from PCM. The DRAM partition is used as a buffer for PCM, and both partitions are managed with an adaptive hardware data mapping scheme that promotes not only write-intensive but also frequently accessed data. The PCM lifetime is estimated below five years, and the average energy-delay-squared product of their system is 24% smaller than of a hybrid system with static data mapping, and 13% smaller than of two state-of-the-art hybrid systems [16, 28]. The authors do not evaluate the effect of a greater main memory capacity on the number of disk accesses.

Fang et al. [24] consider the same system organization as Ramos et al. [18]. Motivated by the differences among the characteristics of DRAM and memory technologies like PCM, they propose a universal memory protocol instead of DDR3 [41]. Their hybrid system just illustrates the viability of their *universal memory architecture*. The system employs one DRAM and one PCM partitions of equal capacities and a static, even interleaving of program data across the DRAM and PCM devices. The authors conclude that the performance of the hybrid system is 12% lower than the performance of a DRAM-only system of the same total capacity. They state that the capacity of the PCM partition can be increased but do not evaluate it.

Unlike the previous authors, Yoon et al. [21] propose an *FR2–S* (replicating) system. Similarly to Zhang and Li [28], Ramos et al. [18], and Fang et al. [24], the partitions are built from DRAM and PCM. A data mapping scheme adaptively promotes frequently accessed data and data with high row buffer miss rates. The DRAM partition is organized as a 16-way set-associative cache for the PCM partition, but is bypassed for all accesses to data residing in the PCM partition only (i.e., to data not replicated to the DRAM partition). Both partitions are directly accessible and thus the system is flat. Similarly to Dong et al. [27], Ramos et al. [18], and Fang et al. [24], the authors exclude disk from evaluation. Compared to a baseline hybrid system where DRAM acts as a conventional cache for PCM, their system achieves 41% higher performance.

Summary

Dong et al. [27], Phadke and Narayanasamy [23], Sudan et al. [19], Mogul et al. [20], Zhang and Li [28], Ramos et al. [18], and Fang et al. [24] propose flat, migrating hybrid systems. Only Yoon et al. [21] propose a replicating system. Only Dong et al. [27] study a hybrid system with partitions on different physical levels. Only Phadke and Narayanasamy [23] divide their hybrid system into three partitions. Dong et al. [27], Phadke and Narayanasamy [23], and Sudan et al. [19] build partitions from either conventional or customized DRAM, while Mogul et al. [20], Zhang and Li [28], Ramos et al. [18], Fang et al. [24], and Yoon et al. [21] combine conventional DRAM with an NVM technology. Dong et al. [27] bridge the processor/memory performance gap at the cost of higher power dissipation. Phadke and Narayanasamy [23] and Fang et al. [24] target the same goal but manage to reduce memory power dissipation. Unlike Dong et al. [27] and Phadke and Narayanasamy [23], Sudan et al. [19], Mogul et al. [20], Ramos et al. [18], and Yoon et al. [21] bridge the memory/disk performance gap by increasing main memory capacity. Only Zhang and Li [28] place all main memory inside the processor package and improve main memory latency and bandwidth, thereby increasing the memory/disk performance gap.

4.2.2 Hierarchical Systems

The earliest work in this category is by Ekman and Stenstrom [14, 15]. Both partitions of their *HM2–D* system are built from DRAM. One partition is local to the processor and acts as a fully-associative cache for the other partition that is remote and accessible via a fiber optic link. The authors make remote as much local memory as possible within a fixed memory performance degradation thus reducing the cost and power of local memory. They conclude that only 30% of memory resources must be accessed at the local DRAM speed (about 100ns) for the workloads in their study. The remaining memory resources can be accessed at a speed one order of magnitude slower (up to 1μs). Their hybrid system has an average performance overhead of 1.2% compared to an all-local DRAM baseline of the same capacity. Disk is not a performance bottleneck since the workload is in-memory w.r.t. baseline main memory. The authors state that their organization could be generalized to 1) a hybrid system with fast and slow (compressed or aggressively power-managed) DRAM partitions located on the same physical level, i.e., the main memory level, and 2) a hierarchical, migrating system with *N* partitions.

Similarly to Ekman and Stenstrom [14, 15], Sudan et al. [17] and Badam and Pai [22] organize the DRAM partition of their *HM2–D* systems as a fully-associative cache for the second partition, but build the latter from NAND Flash. Unlike Ekman and Stenstrom [14, 15], the authors bridge the memory/disk performance gap by increasing main memory capacity with a large NAND Flash partition accessible via a Peripheral Component Interconnect express (PCIe) bus. The system improves the throughput of programs used in their study from 131% to 1740% compared to a system with the same amount of DRAM but no NAND Flash. Their hybrid memory manager (implemented in the OS) reduces write traffic to NAND Flash and thus increases its lifetime up to 32× compared to a system where NAND Flash is organized as a disk cache.

Ye et al. [13] bridge the memory/disk performance gap but, similarly to Ekman and Stenstrom [14, 15], evaluate systems with in-memory workloads only. They investigate both *HM2–S* and *HR2–S* systems, where one partition is composed of DRAM and the other of a slower memory technology (similar to NAND Flash). Compared to a system with DRAM-only main memory, the performance degradation of their hybrid system of the same capacity stays within 10% for the slow partition read latencies up to 40μs if the DRAM partition contains on average 75% of the workload's working set for their programs. A small write buffer can hide the slow partition write latencies of up to 1ms.

Similarly to Ye et al. [13], Bivens et al. [25] and Dube et al. [26] build the second partition of their *HM2–S* system from a hypothetical technology termed Storage-Class Memory (SCM) that is slower than DRAM (e.g., PCM or NAND Flash). The DRAM partition acts as a direct-mapped cache for the SCM partition. Similarly to Ye et al. [13], they consider a range of SCM access latencies. An individual SCM device in their system must have a read latency below 350ns and a minimum write bandwidth of 200MB/s. This places NAND Flash outside the feasible working region within the memory performance degradation of 5% compared to a DRAM-only system of the same capacity. Similarly to Ekman and Stenstrom [14, 15] and Ye et al. [13], the authors exclude disk from evaluation.

Unlike the previous authors, Qureshi et al. [16] implement a bypass mechanism in their *HR2–S* system. They state that disk is a major system performance bottleneck even if augmented with a NAND Flash based disk cache. They replace all DRAM of the baseline system with PCM of the same area but 4× greater capacity and augment it with a DRAM partition 3% its capacity that acts as a 16-way set-associative cache for it. The PCM partition is bypassed for reads from disk (data are transferred directly to DRAM) for all programs and for writebacks to disk for programs with low temporal locality of reference, such as streaming programs. Their hybrid system reduces the number of disk accesses by a factor of five, increases the system performance by a factor of three, provides energy savings of 55%, and incurs a 13% area overhead compared to the baseline DRAM-only system of a 4× smaller capacity. The PCM lifetime is about ten years in their system. Compared to a conventional DRAM system of the same capacity as the PCM partition, their system suffers less than a 10% performance degradation.

Summary

Ekman and Stenstrom [14, 15], Sudan et al. [17], Badam and Pai [22], Bivens et al. [25], and Dube et al. [26] propose hierarchical, migrating hybrid systems. Only Qureshi et al. [16] propose a replicating system with a bypass mechanism. Ye et al. [13] study both migrating and replicating systems. All of the hierarchical systems have two partitions. Ekman and Stenstrom [14, 15], Sudan et al. [17], and Badam and Pai [22] place the partitions on different physical levels. Only Ekman and Stenstrom [14, 15] build both partitions from DRAM, while the others employ DRAM and NVM. Ekman and Stenstrom [14, 15] reduce the conventional DRAM main memory capacity within a fixed performance degradation. Unlike them, Sudan et al. [17], Badam and Pai [22], Ye et al. [13], Bivens et al. [25], Dube et al. [26], and Qureshi et al. [16] bridge the memory/disk performance gap by increasing main memory capacity.

4.3 Summary

I have surveyed and classified hybrid main memory systems not older than 10 years based on their logical and physical organizations, according to a novel *TPN–L* notation. To the best of my knowledge, this is the first attempt to introduce such a classification.

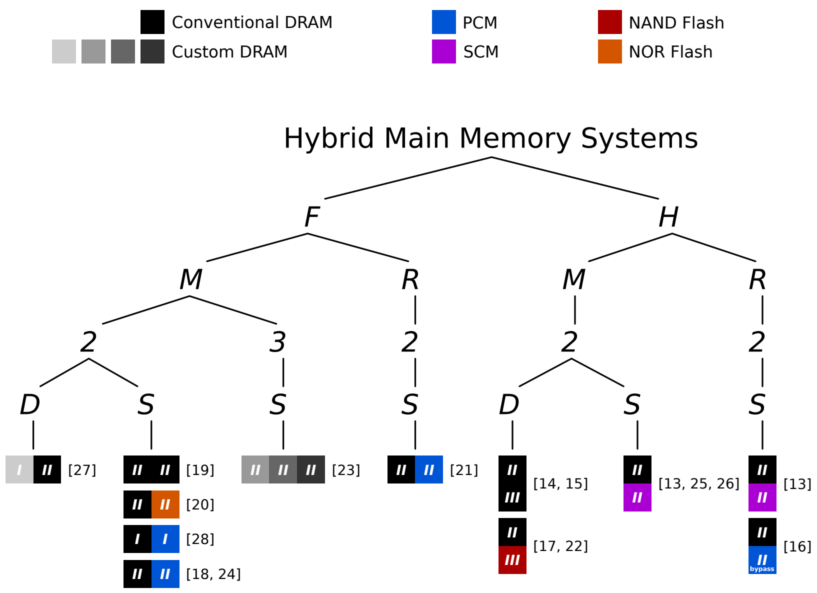


Figure 4.3: *TPN–L classification of contemporary hybrid memory systems*

Figure 4.3 illustrates the classification, where the color squares represent different memory technologies employed for building hybrid system partitions, as per the legend above the figure, the roman numbers (***I***, ***II***, and ***III***) inside the color squares match the physical levels in Figure 4.1, and each group of the color squares thus represents a specific hybrid memory organization. The references to the right from each group of the color squares cite the publications that propose the respective hybrid memory organizations.

Among the 14 different hybrid system proposals surveyed, I have identified four unique flat organizations and three unique hierarchical organizations. Classifying them further by the types of memory technologies and the physical locations of partitions, I have identified seven unique flat systems and five unique hierarchical systems. Among the seven flat systems the majority are migrating with two partitions on the same physical level. All of the five hierarchical systems have two partitions. Two hierarchical systems are migrating with partitions on different physical levels. Three hierarchical systems have their partitions on the same physical level, and among them one is migrating and two are replicating. One of the hierarchical, replicating systems employs a bypass mechanism.

5
Hybrid Memory Systems Design Methodology

NVM technologies introduce a new dimension to the system design space. They typically bridge the access latency, access dynamic energy, bit density, and bit cost gaps between the conventional main memory technology, DRAM, and backing store technologies, collectively called disk here. NVM technologies such as PCM and NAND Flash can be bit-denser and less expensive per bit than DRAM. The potential benefits of combining DRAM with such technologies instead of building DRAM-only main memory are: 1) a lower cost for main memory of the same capacity; and 2) a larger capacity for main memory of the same area, where area is expressed in the number of memory devices, and one DRAM device has the same area in mm^2 as one NVM device, as described in Section 5.1. For instance, if the baseline system has two channels, two Dual In-line Memory Modules (DIMMs) per channel, 16 memory devices per DIMM, the total area is 64 devices. An equal-area hybrid system would have 64 devices, too, some of which would be DRAM and some NVM. A larger main memory capacity can reduce the number of disk accesses and consequently reduce system-level execution time and/or energy.

I identify the following hybrid main memory design challenges: 1) *resource partitioning*, i.e., how much memory of each type to employ; 2) *resource allocation*, i.e., how to distribute the capacity of hybrid partitions among co-running programs; and 3) *data placement*, i.e., where in hybrid main memory to place program data. Each of the three challenges is present both at design-time and at run-time. Design-time Resource Partitioning (DRP) finds the amounts of different memory technologies that have to be installed in the system to satisfy a design goal best. Run-time resource partitioning adjusts these amounts dynamically, provided that memory technologies support such reconfiguration [30]. Design-time Resource Allocation (DRA) statically distributes available main memory capacity among programs in a given workload, and run-time resource allocation dynamically adjusts memory capacity allocated per program. Design-time data placement statically maps program data to hybrid main memory partitions, and Run-time Data Placement (RDP) does so dynamically to satisfy a run-time goal.

DRP is a nontrivial problem, and its best solution depends on many factors, including the workload, properties of memory technologies and disk, and characteristics of non-main-memory subsystems, such as the CPU. Speedups and energy savings compared to the DRAM-only baseline can vary widely. Thus, DRP is a fundamental problem for hybrid memory systems with a multi-dimensional design space.

DRP has been studied using simulators [14–16] and a virtual machine monitor [13]. Simulation typically has a large implementation overhead and consumes significant computational resources (hours for high-performance multicore systems with contemporary workloads). Prototyping hybrid main memory with a virtual machine monitor involves substantial implementation overhead, too, and restricts exploration to the host configuration (e.g., the total main memory capacity). Thus, these approaches complicate extensive design space exploration and impede finding the best solution to DRP.

In order to enable extensive DRP, I propose using analytic, system-level models for execution time and energy that require no lengthy simulations except for the one-time effort of creating program profiles, as described in Section 2.1. The models describe the fundamental processes of the memory hierarchy and indirectly represent implementation-dependent details by assumed parameters, as explained in Section 5.1. Essentially, they embody the standard memory hierarchy models applied in the context of hybrid main memory systems. The model for execution time is employed in Section 5.2 for driving a performance model named *Rock*. It provides first-order estimates of the memory system throughput of different hybrid system configurations. As a result, it emphasizes the importance of DRP. I tackle DRP in Section 5.3 by proposing a novel partitioning method named *Crystal* and driven by both the model for execution time and the model for energy. The chapter is summarized by Section 5.4.

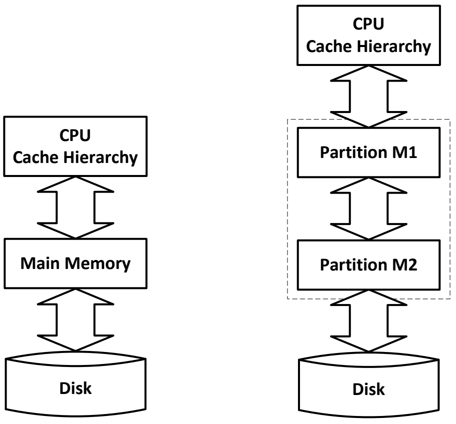
5.1 System-Level Models

For ease of demonstration I make several assumptions about the systems I model. Figure 5.1 depicts logical memory organizations, and Figure 5.2 shows details (the number of cores, main memory channels, and DIMM organization can be different). First, I assume that unlike the conventional main memory of Figure 5.1(a), hybrid memory consists of two exclusive partitions labeled $M1$ and $M2$ and organized hierarchically ($M2$ is only accessible via $M1$), as in Figure 5.1(b). This is an ***HM2–S*** system according to the ***TPN–L*** notation, introduced in Section 4.1.

Programs are represented by profiles recorded in the steady state, as described in Section 2.1. The system workload consists of one single-thread program per core, where the OS can be one of the programs. DRA statically distributes memory capacity among the programs according to a given policy, e.g., as described in Section 2.2. The program with the longest T_{CPU} of the workload programs is labeled p_α. The programs start at the same time and run concurrently, so the workload's T_{CPU} is set to that of p_α.

Program data are managed at the granularity of an OS page. A program can be not-in-memory w.r.t. the memory capacity slice allocated to it after DRA. If a requested page of a not-in-memory program is not in $M1$ of the baseline system or not in $M2$ of the hybrid system, it is paged in from disk. Each page-in (insertion) causes a page-out (eviction), since the miss curve represents steady-state behavior. In the hybrid system, pages evicted from $M1$ are migrated to $M2$. Only dirty pages of a not-in-memory program are paged out to disk, thus the writebacks for each program are defined by its write fraction (fWr). I choose this organization because it is simple to model and it does not waste memory capacity (like replicating systems do). The organization implies a data placement policy that is further referred to as *basic*.

For simplicity of device capacity comparison, I assume that DRAM and NVM devices have the same number of memory cells, and the cells have the same area in μm^2. The higher bit density of NVM is represented by multiple bits per cell. Rows of DRAM and NVM have equal bit capacity, thus an NVM row comprises fewer cells than a DRAM row,



(a) Baseline (b) Two-level main memory

Figure 5.1: *Memory hierarchy organizations*

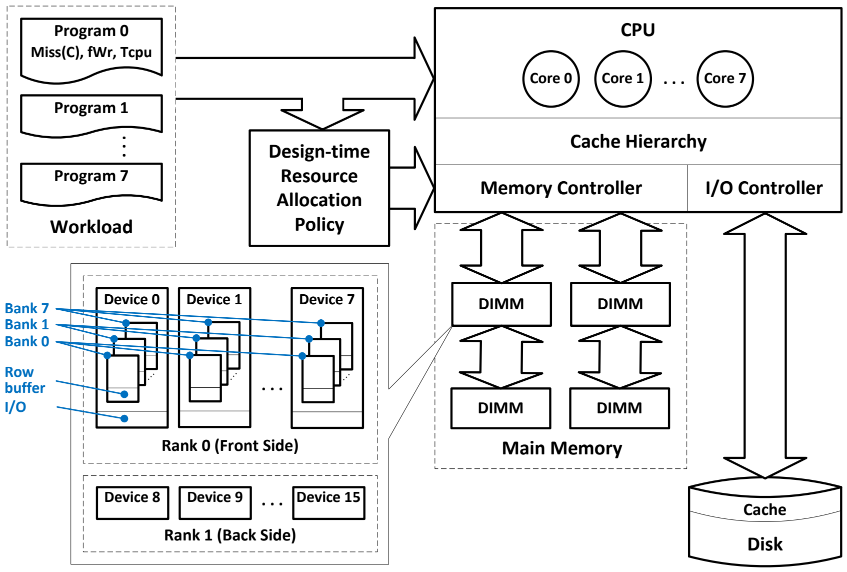


Figure 5.2: *Detailed diagram of system modeled*

Table 5.1: *Assumed model parameters*

Assumed parameter	**Description**
Row buffer hit rate	Row buffer hit rate for $M1$ reads and writes
Buffer disk writes	For each program, overlap its disk write time with T_{CPU} and main memory time
Disk cache hit rate	Disk cache hit rate for disk reads
Disk write propagation rate	Fraction of writes to main memory that propagate to disk
Overlap T_{CPU} and T_{mem}	Overlap T_{CPU} of program that defines T_{CPU} of entire workload with main memory and disk times of all other programs in workload

and an NVM device comprises more rows than a DRAM device. In addition, I assume equal peripheral circuits and data interfaces, hence the same static power. This way, the areas of one DRAM and one NVM device are equal, and equal-area main memories have the same number of devices, regardless of type.

Lastly, I introduce a number of assumed model parameters to indirectly represent execution details, as described below. *Row buffer hit rate* models the execution details that decrease the average main memory access latency and dynamic energy. For instance, the parameter models access reordering and bank- and channel-level parallelism utilization by the memory controller. The *Buffer disk writes* parameter models an ideal write buffer in the I/O controller by overlapping each program's disk write time with its T_{CPU} and main memory time. *Disk cache hit rate* models prefetching to the disk cache, I/O controller cache, and disk access reordering. It affects solely disk reads, since only they are likely to hit in the cache [5]. *Disk write propagation rate* models write coalescing in main memory, such that only a fraction of the total number of writes to main memory propagate to disk. The *Overlap T_{CPU} and T_{mem}* parameter overlaps the workload's T_{CPU} with main memory and disk times of all the workload programs except p_{α}. This overlap represents a limit case resulting in the shortest execution time.

Table 5.2: *Variables in Equations* (5.1) *to* (5.12)

$Miss(C)$, fWr, and T_{CPU} are from program profile

Variable	**Description**
n	Number of programs constituting system workload
$A_{\{M1,M2\}}$	Area (in memory devices) of $M1$ and $M2$, respectively
$C_{\{M1,M2\}}$	Capacity of $M1$ and $M2$, respectively
$N_{\{M1,M2,D\}}$	Number of accesses to $M1$, $M2$, and disk, respectively: $N_{M1} = Miss(0)$, $N_{M2} = Miss(C_{M1})$, and $N_D = Miss(C_{M1} + C_{M2})$
$\{T,E\}_{M1\,acc}$	Latency and dynamic energy, respectively, per $M1$ access (acc), where access can be read (Rd) or write (Wr), and size of access is one cache line
$\{T,E\}_{X\rightarrow Y}$	Latency and dynamic energy, respectively, reading one OS page from memory part X, transferring, and writing it to memory part Y
$T_{D\,\{Rd,Wr\}}$	Latency of reading (Rd) and writing (Wr), respectively, one OS page from/to disk
$dwpr$	Assumed disk write propagation rate
$P_{\substack{non-\\mem}}$	System power excluding main memory and disk
$P_{\substack{\{M1,M2,D\}\\stat}}$	Static power of $M1$, $M2$, and disk, respectively. For $M1$ and $M2$, depends on their respective areas and technologies
$P_{\substack{\{M1,M2\}\\maint}}$	Maintenance power of $M1$ and $M2$, respectively (e.g., refresh power of DRAM). Depends on respective areas and technologies of $M1$ and $M2$

Equations (5.1) and (5.2) describe the models for the execution time and dynamic energy of a single-thread program in the baseline system of Figure 5.1(a):

$$\begin{aligned} T_{base} = T_{CPU} &+ N_{M1}\cdot ((1 - fWr)\cdot T_{M1\,Rd} + fWr\cdot T_{M1\,Wr}) \\ &+ N_D\cdot (T_{D\rightarrow M1} + dwpr\cdot fWr\cdot T_{M1\rightarrow D}) \end{aligned} \tag{5.1}$$

$$\begin{aligned} E_{\substack{base\\dyn}} = N_{M1}&\cdot ((1 - fWr)\cdot E_{M1\,Rd} + fWr\cdot E_{M1\,Wr}) \\ &+ N_D\cdot (E_{D\rightarrow M1} + dwpr\cdot fWr\cdot E_{M1\rightarrow D}) \end{aligned} \tag{5.2}$$

where the entire main memory constitutes partition $M1$, and the variables are described in Table 5.2. The number of accesses to main memory (N_{M1}) by each program is obtained from the program's miss curve at capacity 0. The latencies and dynamic energies per $M1$

read and write are calculated by simple equations according to the assumed *Row buffer hit rate* value. E.g., $T_{M1\,Rd} = rbhr\cdot T_{M1\,Rd\,Hit} + (1 - rbhr)\cdot T_{M1\,Rd\,Miss}$, where $rbhr$ is the row buffer hit rate, and $T_{M1\,Rd\,Hit}$ and $T_{M1\,Rd\,Miss}$ are the latencies of servicing one read hitting and missing in the row buffer, respectively. By varying the row buffer hit rate I vary the actual latencies and dynamic energies per $M1$ access. Partition $M2$ is accessed only at a page granularity, thus the row buffer hit rate is constant, and a separate parameter is not needed. The latencies and dynamic energies per disk read are calculated according to the assumed *Disk cache hit rate* value in the same way, and I omit separate equations for brevity. All the access, migration, and paging latencies and dynamic energies include the latencies and dynamic energies of data transfers. For instance, the latency of a page-out from $M1$ to disk ($T_{M1\to D}$) comprises the latencies of reading and transferring the page from $M1$ to the I/O controller and the latencies of transferring and writing it from the I/O controller to disk.

Equations (5.1) and (5.2) represent the fundamental processes of the memory hierarchy. The models for execution time and dynamic energy of the hierarchical, migrating system of Figure 5.1(b) are derived by simply adding partition $M2$ and are given by (5.3) and (5.4), respectively:

$$\begin{aligned} T_{hyb} = {} & T_{CPU} + N_{M1}\cdot((1 - fWr)\cdot T_{M1\,Rd} + fWr\cdot T_{M1\,Wr}) \\ & + N_{M2}\cdot(T_{M2\to M1} + T_{M1\to M2}) \\ & + N_D\cdot(T_{D\to M2} + dwpr\cdot fWr\cdot T_{M2\to D}) \end{aligned} \tag{5.3}$$

$$\begin{aligned} E_{\substack{hyb\\dyn}} = {} & N_{M1}\cdot((1 - fWr)\cdot E_{M1\,Rd} + fWr\cdot E_{M1\,Wr}) \\ & + N_{M2}\cdot(E_{M2\to M1} + E_{M1\to M2}) \\ & + N_D\cdot(E_{D\to M2} + dwpr\cdot fWr\cdot E_{M2\to D}) \end{aligned} \tag{5.4}$$

where the variables are described in Table 5.2.

In order to estimate system-level execution time and energy, I distribute partition capacities among the workload programs according to a DRA policy of choice and apply (5.5) to (5.12). Table 5.2 describes the variables in the equations. For each program I calculate execution time, buffering disk writes if the respective parameter is enabled:

$$T' = \begin{cases} T & \text{if } \textit{Buffer disk writes} = \textit{no} \\ T_{BDW} & \text{otherwise} \end{cases} \tag{5.5}$$

$$T_{BDW} = \begin{cases} \underset{total}{T_{D\,Rd}} + \underset{total}{T_{D\,Wr}} & \text{if } \underset{total}{T_{D\,Wr}} \geq T - \underset{total}{T_{D\,Rd}} - \underset{total}{T_{D\,Wr}} \\ T - \underset{total}{T_{D\,Wr}} & \text{otherwise} \end{cases}$$

where $\underset{total}{T_{D\,Rd}} = N_D \cdot T_{D\,Rd}$, $\underset{total}{T_{D\,Wr}} = N_D \cdot dwpr \cdot fWr \cdot T_{D\,Wr}$, and T is given by (5.1) for the baseline system and by (5.3) for the hybrid system. The conditions for calculating T_{BDW} are as follows. $T - \underset{total}{T_{D\,Rd}} - \underset{total}{T_{D\,Wr}}$ represents the sum of the program's T_{CPU} and main memory time. If the disk write time is greater than or equal to the program's T_{CPU} and main memory time, the former overlaps the latter and defines the result time in addition to the disk read time, that cannot be overlapped. This gives $T_{BDW} = \underset{total}{T_{D\,Rd}} + \underset{total}{T_{D\,Wr}}$. Otherwise, i.e., if the disk write time is less than the program's T_{CPU} and main memory time, the latter overlaps the former and thus defines the result time in addition to the disk read time. This gives $T_{BDW} = T - \underset{total}{T_{D\,Wr}}$. Next, I sum the execution times of the programs overlapping their T_{CPU} to obtain the execution time of the entire workload:

$$T_{sys} = \max_{0 \leq i < n} T_{CPU_i} + \sum_{0 \leq i < n} T'_i - T_{CPU_i} \tag{5.6}$$

where T' is defined by (5.5). Workload's T_{CPU} and T_{mem} are overlapped if the respective parameter is enabled:

$$T'_{sys} = \begin{cases} T_{sys} & \text{if } \textit{Overlap } T_{CPU} \textit{ and } T_{mem} = \textit{no} \\ \underset{OCM}{T_{sys}} & \text{otherwise} \end{cases} \tag{5.7}$$

$$\underset{OCM}{T_{sys}} = \begin{cases} T_\alpha & \text{if } T_{CPU_\alpha} \geq \sum_{0 \leq i < n,\; i \neq \alpha} T'_i - T_{CPU_i} \\ T_{sys} - T_{CPU_\alpha} & \text{otherwise} \end{cases}$$

where α is such that $T_{CPU_\alpha} = \max_{0 \leq i < n} T_{CPU_i}$, and T_{sys} is defined by (5.6). Next, the overhead of ZQ-calibration is added, if a high-speed data interface is employed. ZQCS commands are assumed to overlap for all ranks, so the overhead of ZQ-calibration is calculated by simply dividing execution time over the ZQCS period and multiplying by the latency of the ZQCS command:

$$T''_{sys} = \begin{cases} T'_{sys} + \frac{T'_{sys}}{t_{ZQI}} \cdot t_{ZQCS} & \text{if high-speed data interface employed} \\ T'_{sys} & \text{otherwise} \end{cases} \tag{5.8}$$

where T'_{sys} is given by (5.7), t_{ZQI} is the ZQCS period, and t_{ZQCS} is the ZQCS latency. Finally, DRAM refresh is added, if DRAM is employed by at least one partition. REF commands are assumed to overlap for all ranks, so the latency overhead of DRAM refresh is calculated similarly to the above overhead of ZQCS commands, giving the following equation for system-level execution time:

$$T'''_{sys} = \begin{cases} T''_{sys} + \frac{T''_{sys}}{t_{REFI}} \cdot t_{RFC\,(MIN)} & \text{if DRAM employed} \\ T''_{sys} & \text{otherwise} \end{cases} \tag{5.9}$$

where T''_{sys} is given by (5.8), t_{REFI} is the REF period, and $t_{RFC\,(MIN)}$ is the REF latency. Then, I calculate system-level static and maintenance energies given by (5.10) and (5.11), respectively:

$$E_{\substack{sys\\stat}} = T'''_{sys} \cdot \left(P_{\substack{non-\\mem}} + P_{\substack{M1\\stat}}\left(A_{M1}\right) + P_{\substack{M2\\stat}}\left(A_{M2}\right) + P_{\substack{D\\stat}} \right) \tag{5.10}$$

$$E_{\substack{sys\\maint}} = T'''_{sys} \cdot \left(P_{\substack{M1\\maint}}\left(A_{M1}\right) + P_{\substack{M2\\maint}}\left(A_{M2}\right) \right) \tag{5.11}$$

where T'''_{sys} is given by (5.9). The maintenance power of $M1$ or $M2$ can be, e.g., the refresh power of DRAM, or nothing in case of PCM or NAND Flash. The area of $M2$ (A_{M2}) is zero for the baseline system. Lastly, system-level execution energy is calculated:

$$E_{\substack{sys\\total}} = E_{\substack{sys\\stat}} + E_{\substack{sys\\maint}} + \sum_{0 \le i < n} E_{dyn_i} \tag{5.12}$$

where $E_{\substack{sys\\stat}}$ is given by (5.10), $E_{\substack{sys\\maint}}$ by (5.11), and E_{dyn_i} denotes the dynamic energy of each program in the workload given by (5.2) for the baseline system and by (5.4) for the hybrid system.

5.2 Rock

Rock is a performance model for hybrid systems, inspired by the Roofline model [73]. Rock plots estimates of memory system throughput of different hybrid configurations executing a given workload. The workload is fixed and the system is optimized. Rock is driven by (5.1), (5.3), and (5.5) to (5.9), and thus provides first-order throughput estimates. Throughput is calculated as the total amount of data accessed by the workload (the number of accesses below Last Level Cache (LLC) multiplied by the access size, i.e., the cache line size) over system-level execution time. As a result, Rock illustrates how memory system throughput can be boosted by different memory system optimizations for a given workload, and highlights the importance of Design-time Resource Partitioning (DRP).

5.2.1 Experimental Setup

Workloads

Programs are represented by `CG`, `lbm`, `mcf`, `sjeng`, and `soplex` (their respective profiles are described in Section 2.4). I consider two pairs of workloads and systems: 1) 16 instances of `sjeng`, denoted by *WA*, running on a 16-core system and 2) one each of `mcf`, `lbm`, `CG`, and `soplex`, denoted by *WB*, running on a 4-core system. The programs are assigned one per core, as described in Section 5.1.

System Configuration

For each of the workloads, the maximum area of main memory is defined by the number of DRAM DIMMs required to fit the entire working set of the workload. Each DIMM is dual-rank with eight memory devices per rank, as illustrated in Figure 5.2. I scale the device capacity down by a factor of four in order to increase the number of DIMMs required for workloads WA and WB. This gives the maximum main memory area of six DIMMs. Hybrid configurations are such that the area of the NVM partition is varied from 0% to 100% of the total main memory area. The partitioning granularity, i.e., the area by which DRAM can be replaced with NVM, is one DIMM, since I optimistically assume mixed-technology channels. Table 5.3 summarizes the system configuration. The number of cores is 16 for WA and four for WB. The cache line size defines the size of main memory accesses. The OS page size defines the granularity of the program miss curve and the size of accesses to $M2$ and disk. Table 5.4 shows the assumed model parameters representing the worst case for memory system throughput: no row buffer hits and no disk

Table 5.3: *System configuration for Rock*

Number of cores	16 (WA), 4 (WB)	**Partitioning granularity**	1DIMM
Cache line size	64B	**Datasheet device capacity**	128MB
OS page size	4KB	**Capacity scale-down factor**	4×
Number of DIMMs	6	**Scaled device capacity**	32MB
Devices per DIMM	16	**Scaled DIMM capacity**	512MB

Table 5.4: *Model parameters for Rock*

Row buffer hit rate	0%	**Buffer disk writes**	No
Disk cache hit rate	0%	**Overlap T_{CPU} and T_{mem}**	No
Disk write propagation rate	100%		

Table 5.5: *Selected memory characteristics for Rock*

Main memory numbers are per rank of eight devices

Ratios (×) normalized to respective DRAM numbers

		DRAM	**PCM**	**NAND Flash**	**SSD**	**HDD**
Bit density		1×	4×	16×	Don't care	
Latency	64B Read	35ns	2.26×	715×	Not used	
	64B Write	61ns	4.23×	4253×		
	4KB Read	350ns	1.13×	72×	157×	19386×
	4KB Write	376ns	1.53×	693×	146×	8492×

cache hits, all writes propagate to disk and are not buffered, and the workload's T_{CPU} does not overlap with its memory and disk time.

The 12.8Gb/s DDR3 (×8 DDR at 800MHz) interface is employed by all main memory devices. NVM technologies are represented by PCM and NAND Flash. Their timing characteristics are estimated using the models described in Section 3.3. Although I scale the device capacity, I use the original datasheet timing characteristics for DRAM and NAND Flash. I model the ZQ-calibration latency overhead of the DDR3 interface and the refresh latency overhead of DRAM using (5.8) and (5.9) and the t_{ZQI}, t_{ZQCS}, t_{REFI}, and $t_{RFC\,(MIN)}$ values from Table 3.3. Both the memory and I/O controller delays are assumed 5ns. The 6Gb/s SATA interface is employed by both SSD and HDD, and the models described in Section 3.3 are used for estimating their access latencies. Selected memory characteristics are summarized in Table 5.5. The NVM and disk numbers given as ratios in the table are normalized to the respective DRAM numbers. PCM and NAND Flash are assumed four and 16 times denser than DRAM, respectively. The access latencies of the memory technologies have been discussed in Section 3.3.5.

I consider systems employing DRAM, PCM, and SSD, and systems employing DRAM, NAND Flash, and HDD. Each configuration is evaluated using the high-utility, low-utility, and utility-agnostic DRA policies described in Section 2.2. The implementations of the policies are agnostic to memory technology type and consider only the total capacity of main memory. The capacity of $M1$ is distributed first. The initial slice size is 32MB for each program and the slice delta is 1MB (recall the algorithm in Figure 2.2). The baseline Run-time Data Placement (RDP) policy is the basic one implied by the system organization described in Section 5.1. A hypothetical, *ideal* RDP policy renders hybrid main memory built from the $M1$ memory technology but of the aggregate capacity of partitions $M1$ and $M2$, and thus represents the data placement of an ideal flat hybrid system. I employ it to demonstrate memory system throughput attainable if a hybrid system could serve all accesses at the DRAM speed but retaining its hybrid capacity.

5.2.2 Results

Figures 5.3 to 5.7 show results of applying Rock to workloads WA and WB (defined in the beginning of Section 5.2.1). The X axis of the figures shows the total area of main memory, ranging from one to six DIMMs. The Y axis of the figures shows the area of partition $M2$, ranging from zero to six DIMMs. The area of $M1$ is the difference between the total area and the area of $M2$. Thus, memory technology configurations are represented by (x, y) points. Configurations where $M1$ area is zero are such that NVM completely replaces DRAM and main memory comprises only one partition. The Z axis of the figures shows memory system throughput. The surfaces with green-to-yellow curves show throughput attainable if the high-utility DRA policy is employed, and the surfaces with red-to-yellow curves show that if the low-utility DRA policy is employed. The surface curves are projected onto the XZ plane as contour plots. The blue vertical bars show throughput attainable if the utility-agnostic DRA policy is employed. Throughput attainable if the ideal RDP policy is employed is shown by the cyan vertical bars. The white curves show the equal-area partitioning options.

Figure 5.3 shows results for WA and systems with variable amounts of DRAM and PCM backed by SSD. Since WA consists of multiple instances of the same program

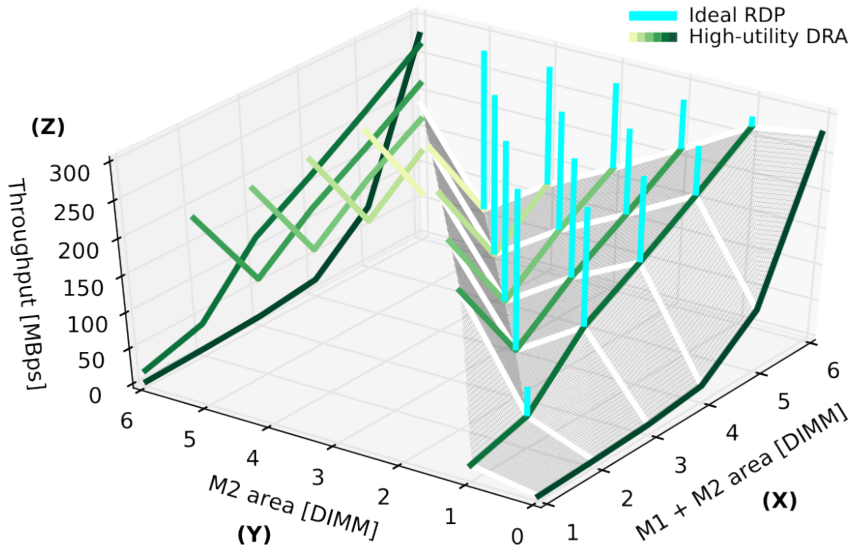


Figure 5.3: *Rock for WA and systems with PCM and SSD*

(sjeng), the high-utility, low-utility, and utility-agnostic DRA policies result in equal capacity distributions, and thus results are shown only for the high-utility policy. sjeng's miss curve decreases monotonically, as illustrated by Figure 2.4. Thus, memory system throughput increases as the capacity of the DRAM-only system is increased, as shown by the green curve for $M2$ area equal to zero ($y = 0$). The throughput of the system with six DRAM DIMMs (point $(6, 0)$) is the maximum throughput because disk accesses are eliminated (WA becomes in-memory) and entire main memory is build from DRAM.

When the total area is one DIMM, replacing DRAM with PCM increases throughput (points $(1, 0)$ vs. $(1, 1)$) because the benefit of reducing the number of disk accesses is greater than the penalty for accessing PCM. The throughput at point $(1, 1)$ is visually similar to the throughput at point $(4, 0)$, because the total capacities are equal (one PCM DIMM has the same capacity as four DRAM DIMMs) but not large enough to make WA in-memory, and disk accesses dwarf the difference between the access latencies of DRAM and PCM.

When the total area is two DIMMs, replacing one DRAM DIMM with one PCM DIMM increases throughput (points $(2, 0)$ vs. $(2, 1)$). But the workload is still not-in-memory, thus disk accesses limit throughput attainable even if the ideal RDP policy is

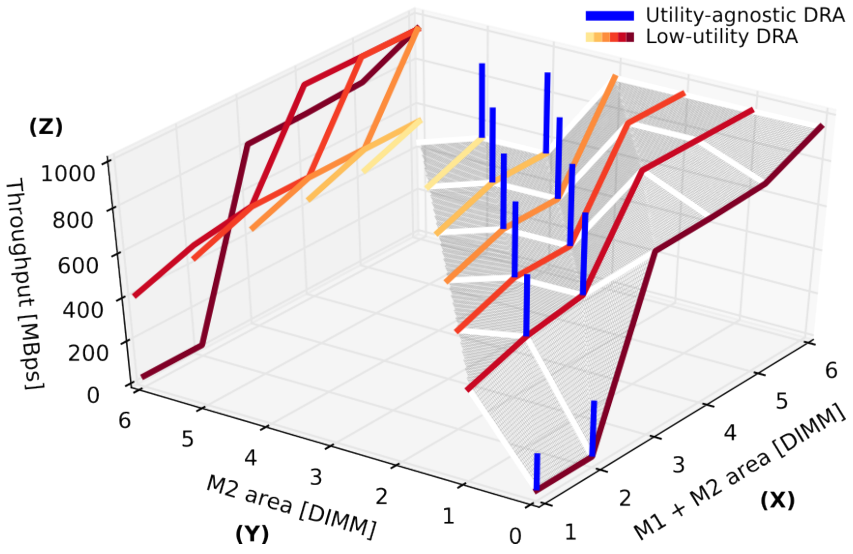


Figure 5.4: *Rock for WB and systems with PCM and SSD (part I)*

employed. This highlights the importance of provisioning enough main memory for the workload and thus eliminating disk accesses in the steady state. For instance, when both DRAM DIMMs are replaced by PCM DIMMs (points $(2, 1)$ vs. $(2, 2)$), the workload becomes in-memory and throughput increases significantly.

When the total area is three DIMMs, replacing one DRAM DIMM with one PCM DIMM increases throughput since the workload becomes in-memory (points $(3, 0)$ vs. $(3, 1)$). Replacing two DRAM DIMMs with PCM reduces throughput (points $(3, 1)$ vs. $(3, 2)$) because the number of page migrations between the DRAM and PCM partitions increases. Replacing all three DRAM DIMMs with PCM boosts throughput (points $(3, 2)$ vs. $(3, 3)$), because the benefit of eliminating migrations between the DRAM and PCM partitions is greater than the penalty for servicing all accesses from PCM. The ideal RDP policy maximizes throughput for the hybrid configurations (points $(3, 1)$ and $(3, 2)$) because disk accesses are eliminated and the entire memory is assumed to be accessed at the DRAM speed. The throughput values of the systems with the total main memory area of four and five DIMMs can be explained similarly. When the total area is six DIMMs, the capacity of the DRAM-only system is large enough to eliminate

Table 5.6: *Low-utility DRA for WB when $M2$ area is zero*

M1 area [DIMM]	M1 capacity slices [MB]			
	mcf	lbm	CG	soplex
1	32	396	52	32
2	32	399	418	175
3	463	403	419	251
4	975	403	419	251
5	1487	403	419	251
6	1674	403	419	251

disk accesses. The workload becomes in-memory and thus does not benefit from larger capacity, and replacing DRAM DIMMs with PCM reduces throughput.

Figure 5.4 shows results for WB and system configurations with variable amounts of DRAM and PCM backed by SSD. Since WB consists of instances of different programs (`mcf`, `lbm`, `CG`, `soplex`), the high-utility, low-utility, and utility-agnostic DRA policies result in different capacity distributions. The results for the low-utility policy are shown by the surface and for the utility-agnostic policy by the blue bars.

The miss curves of `mcf`, `lbm`, `CG`, and `soplex` have distinct plateaus, as illustrated by Figure 2.4, and throughput increases steeply when one or more plateaus get fit into DRAM, as detailed below. Table 5.6 shows the capacity slices allocated per program by the low-utility DRA policy when main memory is DRAM-only ($M2$ area is zero). Consider increasing the total area from two to three DRAM DIMMs (points $(2, 0)$ vs. $(3, 0)$ in Figure 5.4 and rows 2 and 3 of Table 5.6). Throughput increases significantly because the first plateau of `mcf` (about 100MB) and the entire working sets of `lbm` (403MB), `CG` (419MB), and `soplex` (251MB) get fit into DRAM. Adding more DRAM DIMMs reduces the number of accesses to disk initiated by `mcf` (rows 4-6 of Table 5.6) and results in a much smaller throughput increase.

When the total area is one DIMM, replacing DRAM with PCM (points $(1, 0)$ vs. $(1, 1)$) increases the total capacity, and this increases throughput, because the benefit of reducing the number of disk accesses of `mcf` and eliminating the disk accesses of `lbm`, `CG`, and `soplex` is greater than the penalty for accessing PCM directly.

Likewise, when the total area is two DIMMs, replacing one DRAM DIMM with a PCM DIMM increases throughput (points $(2, 0)$ vs. $(2, 1)$): The benefit of reducing the number of disk accesses of `mcf` and eliminating the disk accesses of `lbm` and `soplex` is greater than the overhead of page migrations between the DRAM and PCM partitions.

Table 5.7: *Low-utility DRA for WB when total area is six DIMMs*

		Capacity slices [MB]							
Row	M2 area [DIMM]	`mcf`		`lbm`		`CG`		`soplex`	
		M1	M2	M1	M2	M1	M2	M1	M2
1	0	1674	0	403	0	419	0	251	0
2	1	1487	187	403	0	419	0	251	0
3	2	975	699	403	0	419	0	251	0
4	3	463	1211	403	0	419	0	251	0
5	4	32	1642	399	4	418	1	175	76
6	5	32	1642	396	7	52	367	32	219
7	6	0	1674	0	403	0	419	0	251

Replacing all two DRAM DIMMs with PCM slightly reduces throughput (points $(2,1)$ vs. $(2,2)$): The penalty for accessing PCM directly is greater than the benefit of eliminating disk accesses and the overhead of migrations between DRAM and PCM.

When the total area is three DIMMs, replacing one DRAM DIMM with one PCM DIMM reduces throughput (points $(3,0)$ vs. $(3,1)$). This is so because the low-utility DRA policy does not allocate enough DRAM to `mcf` to fit its first plateau and the overhead of page migrations between the hybrid partitions is added, despite that the workload becomes in-memory (two DRAM DIMMs and one PCM DIMM provide enough capacity to fit the entire WB working set). Replacing two DRAM DIMMs with PCM does not significantly increase the number of migrations between the partitions, and throughput does not visibly change (points $(3,1)$ vs. $(3,2)$). Replacing all three DRAM DIMMs reduces throughput (points $(3,2)$ vs. $(3,3)$), because the penalty for accessing PCM directly is greater than the benefit of eliminating migrations between the partitions.

When the total area is four DIMMs, replacing one DRAM DIMM with one PCM DIMM increases throughput (points $(4,0)$ vs. $(4,1)$), because disk accesses are eliminated and the amount of DRAM is sufficient to fit the first plateau of `mcf` and the entire working sets of `lbm`, `CG`, and `soplex`. Replacing two DRAM DIMMs with PCM (points $(4,1)$ vs. $(4,2)$) steeply reduces throughput, because the low-utility DRA policy does not allocate enough DRAM to fit the first plateau of `mcf`, and the overhead of page migrations between the hybrid partitions is added. At this point, WB does not benefit from more capacity, and replacing more DRAM DIMMs with PCM reduces throughput further. The steep decrease of throughput at points $(5,3)$ and $(6,4)$ is explained the same way as that at point $(4,2)$, described above.

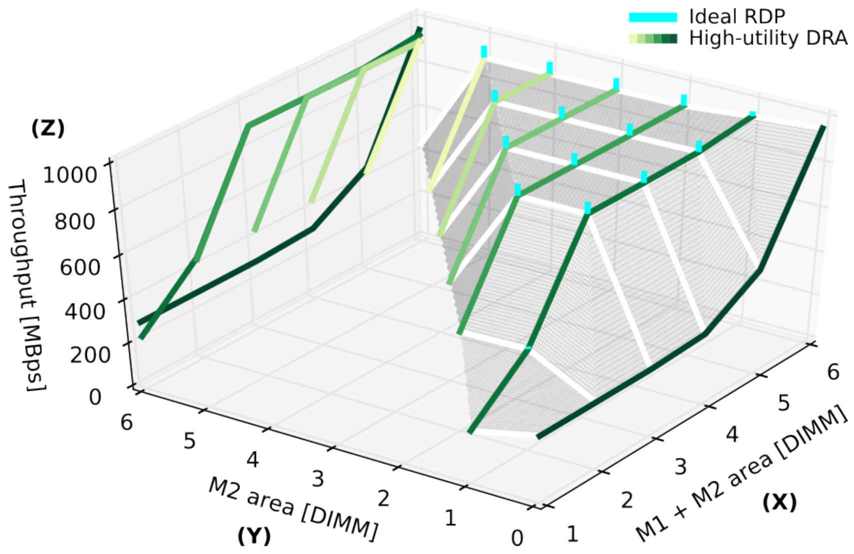


Figure 5.5: *Rock for WB and systems with PCM and SSD (part II)*

When the total area is six DIMMs, replacing one to three DRAM DIMMs with PCM results in the visibly similar throughput values (points $(6, 0)$ vs. $(6, 1)$ to $(6, 3)$), as explained below. Table 5.7 shows the capacity slices allocated per program by the low-utility DRA policy when the total area of main memory is six DIMMs. Rows 1-4 correspond to the configurations when the PCM partition area is zero to three DIMMs, respectively. For these configurations, the capacity of the DRAM partition is large enough to fit the first plateau of `mcf` (about 100MB) and the entire working sets of `lbm` (403MB), `CG` (419MB), and `soplex` (251MB). There are no disk accesses, and the number of page migrations between the hybrid partitions increases relatively slowly as DRAM DIMMs get replaced with PCM DIMMs, because the second plateau of `mcf` is almost flat.

The throughput values attainable when the utility-agnostic DRA policy is employed are shown by the blue error bars in Figure 5.4. The policy manages to achieve higher throughput than the low-utility policy because it allocates enough DRAM to `mcf` to fit its first plateau.

Figure 5.5 shows results for the high-utility DRA policy by the surface and the results for ideal RDP policy by the cyan bars. Increasing DRAM-only capacity (the green curve for $y = 0$) gradually increases throughput, as detailed below. Table 5.8 shows the

Table 5.8: *High-utility DRA for WB when $M2$ area is zero*

M1 area [DIMM]	M1 capacity slices [MB]			
	`mcf`	`lbm`	`CG`	`soplex`
1	187	33	41	251
2	699	33	41	251
3	1211	33	41	251
4	1674	43	80	251
5	1674	216	419	251
6	1674	403	419	251

capacity slices allocated per program by the high-utility DRA policy when main memory is DRAM-only ($M2$ area is zero). The high-utility policy fits into DRAM the entire working set of `soplex` (251MB) and allocates enough DRAM to `mcf` to fit its first plateau (about 100MB) already when the total area is just one DIMM (point $(1, 0)$ in Figure 5.5 and row 1 of Table 5.8). Adding more DRAM DIMMs gradually reduces the number of disk accesses of `mcf` and thus gradually increases throughput (points $(2, 0)$ to $(4, 0)$ in Figure 5.5 and rows 2-4 of Table 5.8). Increasing the total area to five DIMMs fits the entire working set of `CG` (419MB) and results in a visible throughput boost (points $(4, 0)$ vs. $(5, 0)$ in Figure 5.5 and row 5 of Table 5.8). Finally, making WB in-memory at point $(6, 0)$ maximizes the throughput.

Comparing to the results in Figure 5.4, the high-utility policy achieves better results than the low-utility policy and similar results to the utility-agnostic policy at points $(1, 0)$ and $(2, 0)$, because it fits the first plateau of `mcf` into DRAM. It achieves worse results at points $(3, 0)$, $(4, 0)$ and $(5, 0)$, because it favors allocating DRAM to `mcf` instead of fitting the entire working sets of `lbm` and `CG` (recall the example of how the high-utility DRA policy can perform worse than the low-utility DRA policy illustrated by Figure 2.3).

When the total area is one DIMM, replacing the single DRAM DIMM with a PCM DIMM reduces throughput (points $(1, 0)$ vs. $(1, 1)$ in Figure 5.5), because the benefit of reducing disk accesses is smaller than the penalty for accessing PCM directly. The throughput value is lower than that attainable when the low-utility policy or the utility-agnostic policy is employed, because the high-utility policy favors allocating PCM to `mcf` and does not fit the entire working sets of `lbm` and `CG`.

When the total area is two DIMMs, replacing one DRAM DIMMs with one PCM DIMMs increases throughput (points $(2, 0)$ vs. $(2, 1)$ in Figure 5.5), because the benefit of eliminating disk accesses for `mcf` and `CG` is greater than the overhead of page migrations

Table 5.9: *High-utility DRA for WB when total area is three DIMMs*

		Capacity slices [MB]							
		`mcf`		`lbm`		`CG`		`soplex`	
Row	M2 area [DIMM]	M1	M2	M1	M2	M1	M2	M1	M2
1	0	1211	0	33	0	41	0	251	0
2	1	699	975	33	370	41	378	251	0
3	2	187	1487	33	370	41	378	251	0
4	3	0	1674	0	403	0	419	0	251

between the DRAM and PCM partitions. Since WB is still not-in-memory w.r.t. the combined capacity of one DRAM DIMM and one PCM DIMM, disk accesses limit throughput attainable even if the ideal RDP policy is employed (the cyan bar is barely visible at point $(2, 1)$). Replacing both DRAM DIMMs with PCM happens to result in visibly the same throughput (points $(2, 1)$ vs. $(2, 2)$): The benefit of eliminating disk accesses and migrations between the hybrid partitions happens to be similar to the penalty for accessing PCM directly, so they cancel out each other.

Table 5.9 shows the capacity slices allocated per program by the high-utility DRA policy, when the total area of main memory is three DIMMs. Replacing one DRAM DIMM with one PCM DIMM increases throughput steeply (points $(3, 0)$ vs. $(3, 1)$). This is so because WB becomes in-memory (the sum of the capacity slices in row 2 of Table 5.9 equals the aggregate working set size of WB), and the overhead of page migrations between the hybrid partitions is dwarfed by the benefit of eliminating disk accesses. The throughput value at point $(3, 1)$ is close to the maximum (point $(6, 0)$), as indicated by the relatively short cyan bar of the ideal RDP policy. Thus, the ideal RDP policy offers a relatively small throughput boost compared to the throughput attainable by the basic RDP policy implied by the hybrid system organization. Replacing two DRAM DIMMs with PCM does not visibly reduce throughput (points $(3, 1)$ vs. $(3, 2)$), because the number of migrations between the hybrid partitions increases relatively little. Replacing all three DRAM DIMMs with PCM decreases throughput steeply (points $(3, 2)$ vs. $(3, 3)$), because the penalty for accessing PCM directly is greater than the benefit from eliminating migrations between the hybrid partitions. The remainder of the surface in Figure 5.5 offers no new insights, so I omit describing it for brevity.

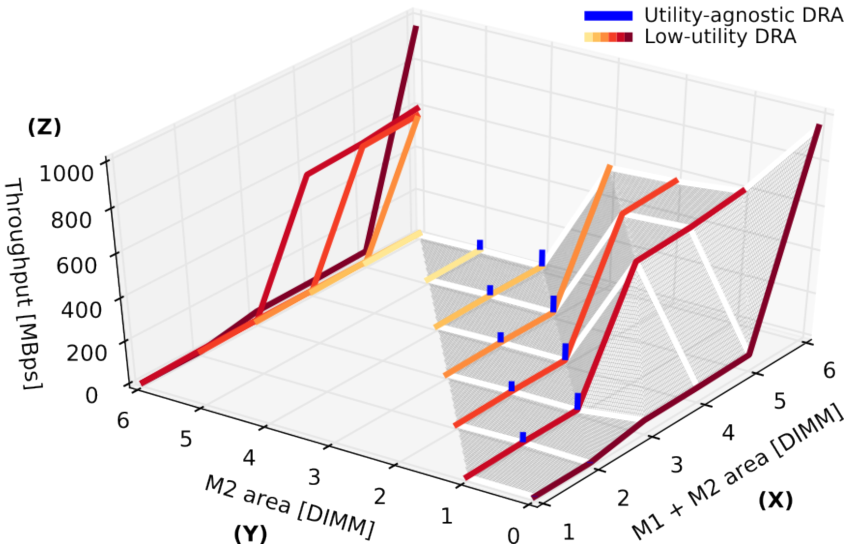


Figure 5.6: *Rock for WB and systems with NAND Flash and HDD (part I)*

Figure 5.6 shows results for WB and system configurations employing the low-utility and utility-agnostic DRA policies with variable amounts of DRAM and NAND Flash backed by HDD. Some similarities can be observed between Figure 5.6 and Figure 5.4, but Figure 5.6 highlights the negative impact of the long latencies of NAND Flash: The throughput values attainable by the systems with at least one NAND Flash DIMM are much lower than those attainable by the systems employing PCM in Figure 5.4. Likewise, the two figures illustrate how the long latencies of HDD result in a much lower memory system throughput for all configurations where disk is accessed, compared to the respective configurations with SSD. Increasing the total area by adding DRAM (the red curve for $y = 0$ in Figure 5.6) increases throughput slowly up to point $(5, 0)$. The steep throughput increase (points $(5, 0)$ vs. $(6, 0)$) emphasizes the importance of eliminating disk accesses.

When the total area is one DIMM, replacing the single DRAM DIMM with a NAND Flash DIMM provides enough main memory capacity to make WB in-memory (NAND Flash is $16\times$ bit-denser than DRAM). However, this does not visibly increase throughput (points $(1, 0)$ vs. $(1, 1)$). This is so because the benefit of eliminating disk accesses does not dwarf the penalty for accessing NAND Flash directly. When the total area is two

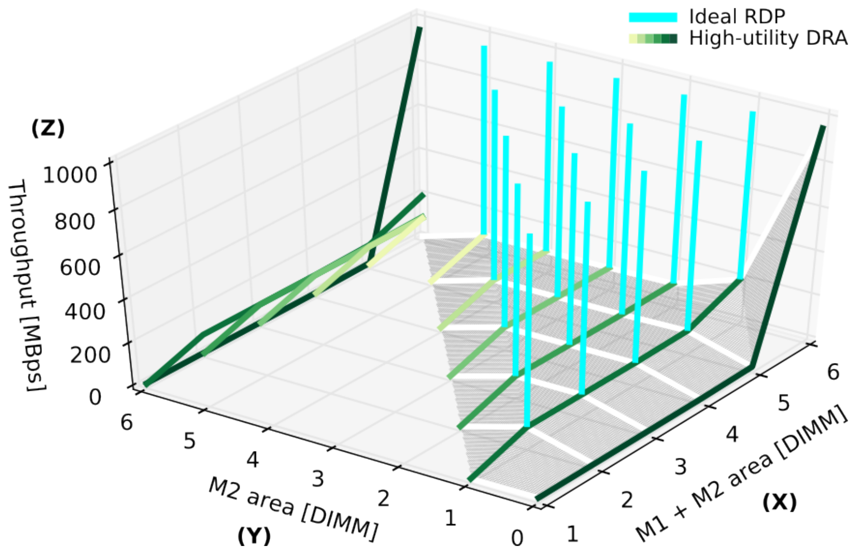


Figure 5.7: *Rock for WB and systems with NAND Flash and HDD (part II)*

DIMMs, replacing one DRAM DIMM with one NAND Flash DIMM does not visibly increase throughput (points $(2, 0)$ vs. $(2, 1)$), because the benefit of reducing the number of disk accesses of `mcf` and eliminating the disk accesses of `lbm` and `soplex` does not dwarf the overhead of page migrations between the DRAM and NAND Flash partitions. When the total area is three DIMMs, replacing one DRAM DIMM with one NAND Flash DIMM decreases throughput (points $(3, 0)$ vs. $(3, 1)$) for the same reason as such partitioning decreases throughput in Figure 5.4. The remainder of the figure resembles a scaled version of Figure 5.4 and offers no new insights.

Figure 5.7 shows results for WB and system configurations employing the high-utility DRA policy with variable amounts of DRAM and NAND Flash backed by HDD. The throughput values attainable by employing the ideal RDP policy are shown by the cyan bars. The surface resembles a scaled version of the surface in Figure 5.5. The long access latencies of NAND Flash and HDD are emphasized by the lower throughput values at all points except $(6, 0)$, where the DRAM capacity is large enough to make WB in-memory. Since NAND Flash is $16\times$ bit-denser than DRAM, the system at point $(2, 1)$ fits the entire working set of WB, and the ideal RDP policy achieves the maximum throughput for the workload.

5.2.3 Discussion

The three DRA policies considered in this paper are distinct but do not represent limit cases. Adding DRA polices that result in the minimum and maximum throughput for each system configuration would provide more insights. Additional insights can be obtained by varying the assumed model parameters. Further, Rock can be extended to an energy-efficiency model for hybrid memory systems by employing the model for system-level execution energy presented in Section 5.1.

5.2.4 Concluding Remarks

To the best of my knowledge, Rock is the first performance model for hybrid memory systems. It is driven by the system-level execution time model presented in Section 5.1, and provides first-order estimates of memory system throughput. Rock is quick and enables exhaustive design-space exploration. It can yield important insights, e.g.:

- Partitioning main memory area between DRAM and NVM can significantly boost memory system throughput, for instance, as between points $(4, 0)$ and $(4, 1)$ in Figures 5.3, 5.5, and 5.6;
- The choice of the DRA policy can significantly affect the result of Design-time Resource Partitioning (DRP), as between points $(3, 0)$ and $(3, 1)$ in Figure 5.4, where throughput decreases if the low-utility policy is employed and increases if the utility-agnostic policy is employed; and
- Throughput boost attainable by the ideal RDP policy can be limited because of disk accesses, as at point $(2, 1)$ in Figures 5.3 and 5.5, or because the total number of accesses below LLC dwarfs the overhead of page migrations between the hybrid partitions, as on top of the surface in Figure 5.5.

DRP is the fundamental design challenge of hybrid memory design. Rock shows that DRP can improve throughput significantly, and the best area partitioning between DRAM and NVM is not trivial. This motivates the subject of the next section, a novel DRP method named *Crystal*.

5.3 Crystal

Hybrid main memory offers a larger capacity than an equal-area DRAM-only memory, where area is expressed in the number of memory devices. According to Section 2.3, not-in-memory workloads benefit from a larger memory capacity as long as their working sets contain pages with reuse distances greater than or equal to the baseline DRAM-only capacity and less than the larger capacity of hybrid memory, where the capacities are expressed in the number of pages. Partitioning main memory for such workloads has a high potential of increasing the performance of memory systems, as has been demonstrated by Rock in Section 5.2.2. Finding the right amount of DRAM and NVM to be installed in a system executing a specific workload is a nontrivial problem. Its solution depends on many factors, and the performance and energy efficiency of equal-area system configurations with different area partitioning between DRAM and NVM can vary dramatically. The goals of such Design-time Resource Partitioning (DRP) can be different, e.g., to minimize system-level execution time or energy. As has been stated in the beginning of this chapter, DRP has been studied by means of detailed simulation [14–16] and prototyping [13]. These approaches typically imply high efforts and thus impede extensive partitioning, required for finding the best design points. This section presents Crystal, a DRP method that enables such extensive partitioning and quickly identifies promising design points for further detailed evaluation, as described below.

The main idea behind Crystal is to frame design-time main memory partitioning as an optimization problem in which the minimum of a target metric (execution time or energy) is sought. The trend of that metric is of more interest than absolute values, and the precision of detailed simulation or prototyping is unnecessary. I propose driving Crystal by the first-order, analytic, system-level models for execution time and energy presented in Section 5.1. The models do not require lengthy simulations, thus Crystal enables rapid, exhaustive search by examining all partitioning options at the partitioning granularity. Results of applying Crystal to a number of system configurations and workloads are presented in Section 5.3.3. I validate Crystal by means of sensitivity analysis, varying the model parameters in broad ranges in Section 5.3.4.

5.3.1 Complexity of Equal-Area Partitioning

A target metric might have several minima as a function of DRAM and NVM partition capacities, when the goal is to find the best memory area partitioning between the two technologies. For instance, consider the execution time of a workload whose miss

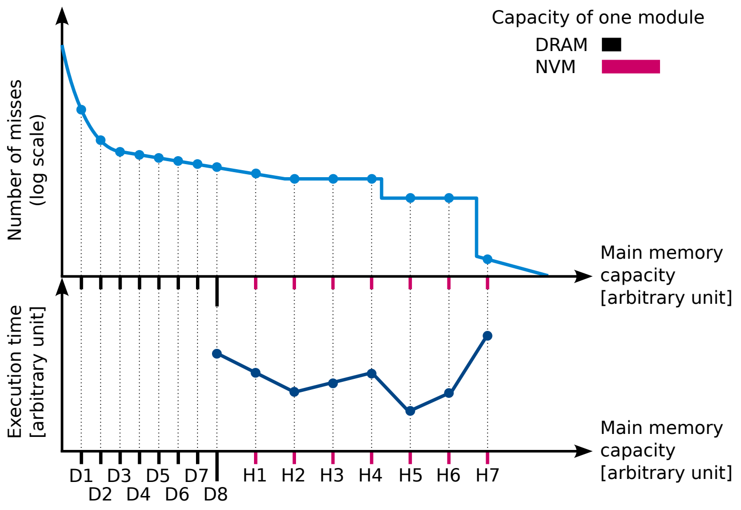


Figure 5.8: *Equal-area partitioning and multiple local minima of execution time*

curve decreases from 0 to 1GB, stays flat up to 2GB forming a flat plateau, and then decreases again. Increasing memory capacity from 1GB to 2GB increases execution time, because DRAM is replaced with slower NVM, but the number of disk accesses does not change. There is one local minimum at 1GB and one more at a capacity above 2GB. The workload's miss curve might have multiple flat plateaus, each creating a local target metric minimum.

Figure 5.8 illustrates multiple local minima of execution time as a result of equal-area partitioning. The figure shows a miss curve in the top part of the graph and execution time in the bottom part. Execution time is expressed in arbitrary units. The horizontal axis represents total main memory capacity. The baseline DRAM-only capacity is indicated by point $D8$ on the horizontal axis i.e., it is eight DRAM modules, where *module* is an arbitrary area unit for DRAM and NVM. The area of one DRAM module equals that of one NVM module, but the capacity of one NVM module is greater than that of one DRAM module. Consider replacing one DRAM module with one NVM module: the DRAM partition capacity becomes $D7$, the NVM partition capacity becomes $H1 - D7$, the total capacity becomes $H1$, the miss curve decreases between points $D8$ and $H1$, and execution time decreases, too. Consider replacing one more DRAM module: the DRAM partition capacity becomes $D6$, the NVM partition capacity becomes $H2 - D6$, the total capacity becomes $H2$, the miss curve decreases between points $H1$ and $H2$, and execution time decreases, too. Replacing one more DRAM module increases execution time, because the miss curve does not decrease between points $H3$ and $H2$. The number of disk accesses is the same as for the previous configuration, but the number of page migrations between the DRAM and NVM partitions increases. Thus, there is a local minimum at point $H2$. Replacing more DRAM modules reveals one more local minimum at point $H5$. When only one DRAM module is left (the DRAM partition capacity is $D1$), the overhead of migrations between the hybrid partitions is greater than the benefit of reducing the number of disk accesses, and execution time at point $H7$ is greater than the execution time of the DRAM-only baseline.

All of potentially multiple minima must be found in order to identify the hybrid configuration that matches a partitioning goal best. Besides minimizing execution time or energy, the goal can include additional criteria such as minimizing system cost and maximizing its lifetime. Crystal uses the models of Section 5.1 for estimating the execution time and energy of all hybrid configurations at the partitioning granularity of choice. Then Crystal identifies the configuration that matches the partitioning goal best. The search time complexity is linear as a function of the number of hybrid configurations.

5.3.2 Experimental Setup

Workloads

Programs are represented by profiles described in Section 2.4. I vary the system workload according to:

$$Wi = p_{large} \times (n - i) + p_{small} \times i \text{ for } 0 \leq i \leq n \tag{5.13}$$

where n denotes the number of cores; p_{large} denotes one of `CG`, `lbm`, or `mcf`; p_{small} denotes `sjeng` or `soplex` (thus the program sets are: `mcf/sjeng`, `mcf/soplex`, `lbm/sjeng`, `lbm/soplex`, `CG/sjeng`, and `CG/soplex`); i denotes the system workload index (which is also the number of cores that run the p_{small} program). For instance, for a system with eight cores workload W0 ($i = 0$) for the `mcf/sjeng` set consists of eight instances of `mcf`, giving it the largest working set and behavior defined by `mcf`. Likewise, W1 consists of seven instances of `mcf` and one instance of `sjeng`. W8 consists of eight instances of `sjeng` and thus has the smallest working set for the `mcf/sjeng` set and behavior defined by `sjeng`. The programs are assigned one per core, as explained in Section 5.1.

Table 5.10: *System configuration for Crystal*

Number of cores	8	**Partitioning granularity**	1DIMM
Cache line size	64B	**Datasheet device capacity**	128MB
OS page size	4KB	**Datasheet system capacity**	8GB
Number of DIMMs	4	**Capacity scale-down factor**	$4\times$
Devices per DIMM	16	**Scaled device capacity**	32MB
		Scaled DIMM capacity	512MB
		Scaled system capacity	2GB

Table 5.11: *Default model parameters for Crystal*

DRAM revision	G [8]	**Row buffer hit rate**	0%
Non-memory power	50W	**Buffer disk writes**	No
Overlap T_{CPU} and T_{mem}	No	**Disk cache hit rate**	0%
		Disk write propagation rate	100%

System Configuration

Table 5.10 shows the system configuration, explained below. The cache line size defines the size of main memory accesses. The OS page size defines the granularity of the program miss curve and the size of accesses to $M2$ and disk. I scale down the capacity of the DRAM-only baseline so that some workloads become not-in-memory. The partitioning granularity, i.e., the area by which DRAM can be replaced with NVM, is one DIMM, since I optimistically assume mixed-technology channels. There are four DIMMs in total, one DIMM is always DRAM in order to implement hybrid main memory with two partitions, hence the partitioning options are one to three NVM DIMMs plus the DRAM-only option.

Table 5.11 summarizes the default model parameters, described below. The default DRAM revision is G [8]. Non-memory power is the system power excluding main memory and disk. The assumed model parameters are as follows: no row buffer hits and no disk cache hits, all writes propagate to disk and are not buffered, and the workload's T_{CPU} does not overlap with its memory and disk time.

The 12.8Gb/s DDR3 ($\times 8$ DDR at 800MHz) interface is employed by all main memory devices, and they are organized in two fully-populated channels (two DIMMs per channel). Each DIMM is dual-rank with eight devices per rank, and each device has eight banks with dedicated row buffers, as illustrated in Figure 5.2. The timing, power, and energy numbers for DRAM, PCM, and NAND Flash are estimated by the models

Table 5.12: *Characteristics of DRAM revisions F and G*

Numbers per rank of eight devices
RB – row buffer

			Revision F		Revision G	
			RB miss	**RB hit**	**RB miss**	**RB hit**
Latency	**[ns]**	64B Read	35.00	18.75	Same as revision F	
		64B Write	61.25	18.75		
Dynamic energy	**[nJ]**	64B Read	40.49	11.24	20.77	6.14
		64B Write	43.65	14.41	24.23	9.61
Power	**[mW]**	Static	840		540	
		Refresh	32		21	

presented in Section 3.3. Although I scale the device capacity, I use the original datasheet numbers. I model the ZQ-calibration latency overhead of the DDR3 interface and both the latency and power overheads of DRAM using (5.8), (5.9), and (3.21) and the t_{ZQI}, t_{ZQCS}, t_{REFI}, $t_{RFC\,(MIN)}$, I_{DD2N}, and I_{DD5B} values from Tables 3.3 and 3.4.

The default DRAM revision is G, and the less energy efficient revision F [8] is used for the sensitivity analysis presented in Section 5.3.4. Table 5.12 summarizes my estimates of selected DRAM numbers. All of the estimates are provided for a rank of eight $\times 8$ devices. Both revisions share the same timing characteristics, and hitting in the row buffer shortens the latencies of 64B reads and writes by about 1.9$\times$ and 3.3$\times$, respectively, as shown by the first two rows of the table. The dynamic energies of 64B reads and writes of revision F are almost 2$\times$ greater than those of revision G because of the larger currents, as per Table 3.4. Hitting in the row buffer reduces the dynamic energy of a 64B read by about 3.6$\times$ for revision F and by about 3.4$\times$ for revision G, as shown by the third row of Table 5.12. The dynamic energy of a 64B write is reduced by about 3$\times$ for revision F and by 2.5$\times$ for revision G if the writes hit in the row buffer, as shown by the fourth row of Table 5.12. The static power of revision F is about 1.5$\times$ greater than that of revision G, as shown by the fifth row of Table 5.12. The refresh power estimates of both DRAM revisions are only about 4% of their respective static power estimates, as shown by the last row of Table 5.12.

Table 5.13 summarizes selected characteristics of DRAM, PCM, NAND Flash, SSD, and HDD. The DRAM, PCM, and NAND Flash estimates are provided for a rank of eight $\times 8$ devices. The NVM and disk estimates given as ratios ($\times$) are normalized to the respective estimates for DRAM revision G. The 6Gb/s SATA interface is employed

Table 5.13: *Selected memory characteristics for Crystal*

Main memory numbers are per rank of eight devices

Ratios (×) normalized to respective DRAM revision G numbers

		DRAM	PCM	NAND Flash	SSD	HDD
Bit density		1×	4×	16×	Don't care	
Latency	4KB Read	350ns	1.13×	72×	157×	19386×
	4KB Write	376ns	1.53×	693×	146×	8492×
Dynamic energy	4KB Read	408nJ	1.02×	7×	65×	19976×
	4KB Write	630nJ	1.14×	42×	89×	9136×
Static power		1×	1×	1×	0.03W	3.00W

Table 5.14: *Hybrids and their baselines*

M1	M2	Disk	DRA policy
a) DRAM	b) PCM	d) SSD	f) Low-utility
	c) NAND Flash	e) HDD	g) High-utility

Label	Organization	Baseline	Label	Organization	Baseline
I-L	`a-b-e-f`	`a-e-f`	III-L	`a-b-d-f`	`a-d-f`
I-H	`a-b-e-g`	`a-e-g`	III-H	`a-b-d-g`	`a-d-g`
II-L	`a-c-e-f`	`a-e-f`	IV-L	`a-c-d-f`	`a-d-f`
II-H	`a-c-e-g`	`a-e-g`	IV-H	`a-c-d-g`	`a-d-g`

by both disk types. PCM and NAND Flash are assumed 4× and 16× bit-denser than DRAM, respectively. The latency and dynamic energy estimates come from Section 3.3.5 and have been discussed there. I assume that the static (background) power of NVM is the same as that of DRAM, as per Section 5.1. The SSD static power is measured after ten minutes of the system being idle [11], thus it is relatively small (e.g., smaller than the static power of a single NAND Flash rank discussed above), but such inaccuracy can be tolerated, as is shown in Section 5.3.4. The latency and dynamic energy of a 4KB read from the disk cache are 100× and 41× for SSD, respectively, and 443× and 457× for HDD, respectively (normalized to those of a 4KB read from DRAM revision G).

Hybrid Organizations

I apply Crystal to eight hybrid systems, or simply *hybrids* for short, shown in Table 5.14. The baselines for the hybrids are DRAM-only systems with respective disk types and DRA policies. Both the high- and low-utility DRA policies initially allocate 32MB of

DRAM to each program, and the capacity slice delta is 1MB (recall the algorithm in Figure 2.2). The implementations of the policies are agnostic to memory technology type and consider only the total main memory capacity. The capacity of $M1$ is distributed first.

5.3.3 Results

I set the goal to partition main memory area between DRAM and NVM such that system-level execution time and energy are within 20% of their respective global minima and the area of the NVM partition is minimized. Due to the first-order nature of the models in the heart of Crystal, I regard speedups and energy savings below 1.5-2$\times$ as noise and dismiss them. Minimizing the area of the NVM partition keeps system cost and lifetime closest to those of the baseline system. Replacing one DRAM DIMM with one NVM DIMM does not reduce main memory cost: For one NVM DIMM to cost less than one DRAM DIMM, NVM must cost less per bit for at least as many times as it is bit-denser than DRAM. In this study this does not hold even for NAND Flash, that is 16$\times$ bit-denser but only about 10$\times$ less expensive than DRAM. Since NVM such as PCM and NAND Flash suffer from lower write endurance than DRAM, the less NVM is employed the better it is for system lifetime.

The bottom parts of the graphs in Figures 5.9 to 5.12 show best hybrid configurations as NVM area normalized to total main memory area. The configurations satisfy the partitioning goal for the default model parameters (Table 5.11) and workloads W0-W8 (recall (5.13) in the beginning of Section 5.3.2) from different program sets. NVM area ranges from 0 to 75%: Main memory comprises four DIMMs in total, hence one DIMM is 25%, and one DIMM is always DRAM to implement a hybrid with two partitions. DRAM area is the difference between total main memory area and NVM area. The configurations found for execution time as the target metric are shown by bars, and those for execution energy are shown by diamonds. The top parts of the graphs show improvements—speedups (by bars) and energy savings (by diamonds)—attainable by the best hybrid configurations. Energy savings are very close to speedups for all the hybrids and workloads, thus I show the two metrics together.

Figure 5.9 shows results for the hybrids with HDD partitioned for workloads from the `mcf/soplex` set. The hybrids employing NAND Flash (II-L and II-H) offer significant improvements for W0-W1 and W3-W4 (17-19$\times$), but the hybrids employing PCM (I-L and I-H) do not. This is so because hybrids II-L and II-H provide enough main memory capacity to fit the entire working sets of these workloads, and the benefit of eliminating

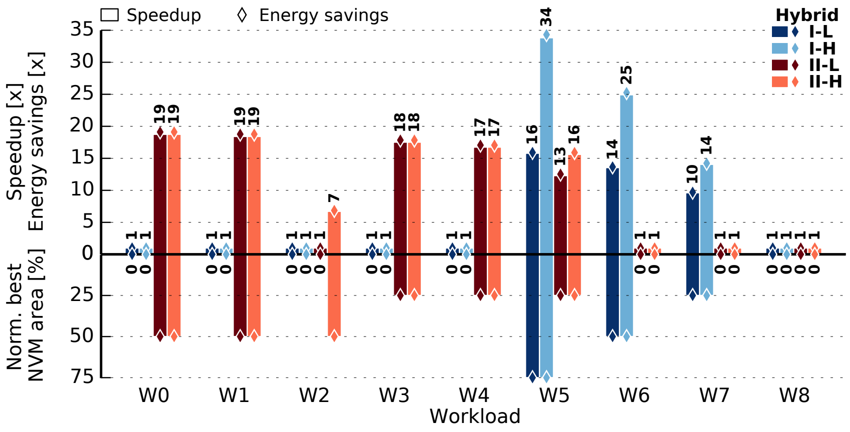


Figure 5.9: *Crystal for hybrids with HDD and* `mcf`*/*`soplex` *set*

HDD accesses is greater than the overhead of accessing the NAND Flash partition. PCM bit density is 4× lower than that of NAND Flash, and the main memory capacity of hybrids I-L and I-H is too small to eliminate HDD accesses even if the area of the PCM partition is maximized to 75%. The overhead of accessing the PCM partition is greater than the benefit of reducing the number of HDD accesses by replacing DRAM with PCM. Normalized PCM area is 0%, i.e., the respective DRAM-only baselines are better than hybrids I-L and I-H for these workloads. The improvements for hybrids II-L and II-H are equal, because the amount of DRAM is such that the high- and low-utility DRA policies result in equal distributions of main memory capacity.

Hybrid II-L is worse than its baseline for W2 in Figure 5.9, despite that the hybrid can provide enough main memory capacity to fit the entire working set of the workload. This is so because the low-utility DRA policy, employed by the hybrid, does not allocate enough DRAM to the `mcf` instances to fit their first plateau (Figure 2.4). `soplex` has a lower utility and wins the DRAM capacity. This results in a great number of page migrations between the DRAM and NAND Flash partitions initiated by the `mcf` instances. These migrations take more time and expend more energy than the baseline HDD accesses. Hybrid II-H attains significant improvements (about 7×) thanks to the high-utility DRA policy, that allocates enough DRAM to the `mcf` instances to fit their first plateau.

Hybrids I-L and I-H attain great improvements (10-34×) for W5-W7 because the overhead of accessing the PCM partition is less than the benefit of eliminating HDD

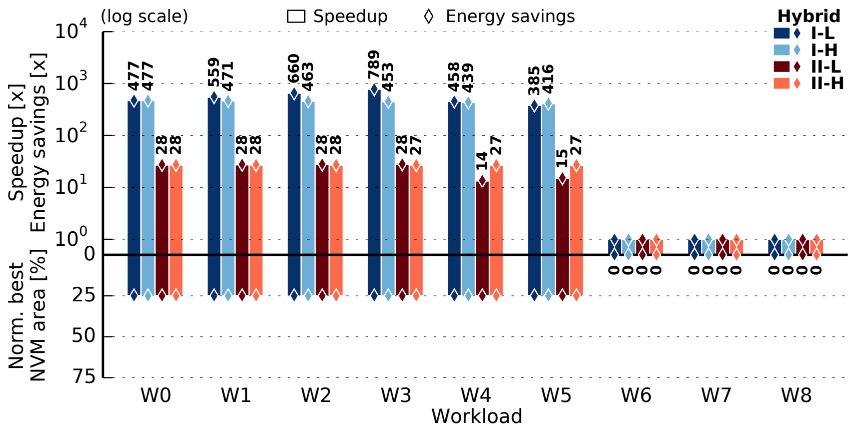


Figure 5.10: *Crystal for hybrids with HDD and* `lbm/sjeng` *set*

accesses. The hybrids offer greater improvements than hybrids II-L and II-H for W5, since PCM is faster and more energy efficient than NAND Flash. Hybrids II-L and II-H are worse than their respective baselines for W6-W7, because the overhead of accessing the NAND Flash partition is greater than the benefit of eliminating HDD accesses. The four hybrids make no sense for W8, because the workload is in-memory w.r.t. the baseline main memory capacity.

Figure 5.10 shows results for the hybrids employing HDD and partitioned for workloads from the `lbm/sjeng` program set. The results can be explained similarly to those in Figure 5.9. The hybrids attain great improvements for W0-W5 (above $14\times$), because they provide enough main memory capacity to fit the entire workload working sets, and the benefit of eliminating HDD accesses is greater than the overhead of accessing the NVM partition. Hybrids I-L and I-H offer greater improvements than hybrids II-L and II-H, since PCM is faster and more energy efficient than NAND Flash. The four hybrids are worse than their respective baselines for W6-W8, because these workloads are in-memory w.r.t. the baseline main memory capacity.

Hybrid I-L offers greater speedups than hybrid I-H for W1-W4, although the execution times of both hybrid I-L and its baseline are greater than those of hybrid I-H and its baseline, respectively. Hybrid I-L offers lower speedups than hybrid I-H for W5, because the execution times of their respective baselines are similar, but the

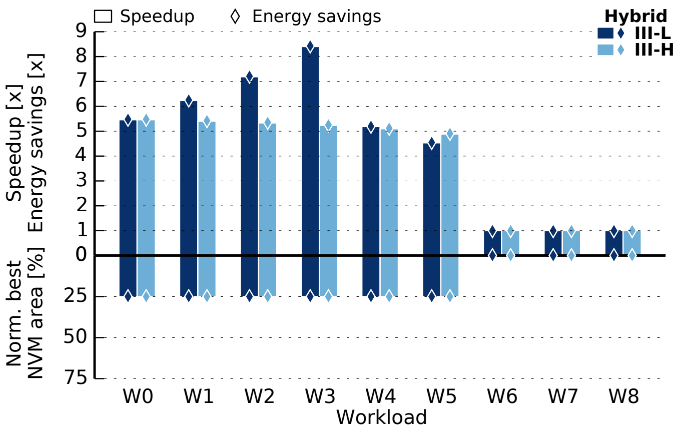


Figure 5.11: *Crystal for hybrids III-L and III-H and* `lbm/sjeng` *set*

execution time of hybrid I-L is greater than that of hybrid I-H. The difference in energy savings offered by the hybrids can be explained in the same way.

As for systems with SSD backing store, the hybrids employing PCM (III-L and III-H) are worse than their respective baselines for the workloads containing `mcf`. The hybrids employing NAND Flash (IV-L and IV-H) are never better than their respective baselines for all the workloads, because the access latency and access dynamic energy of NAND Flash are too close to those of SSD, and the overhead of accessing the NAND Flash partition is always greater than the benefit of reducing the number of SSD accesses.

Figure 5.11 shows results for the hybrids employing PCM and SSD and partitioned for workloads from the `lbm/sjeng` program set. The results resemble those for hybrids I-L and I-H in Figure 5.10. The improvements in Figure 5.11 are significant for W0-W5 (4-8$\times$) but smaller than those offered by hybrids I-L and I-H in Figure 5.10, since SSD is closer to DRAM than HDD in terms of access latency and access dynamic energy.

The results for workloads containing `CG` and respective workloads containing `lbm` are similar, because `CG` and `lbm` have similar miss curves (recall Figure 2.4). For instance, compare Figure 5.12 and Figure 5.10: The best hybrid configurations are the same, although `CG` affects the improvements by having a much smaller fraction of writes and almost double T_{CPU} compared to `lbm` (recall Table 2.1).

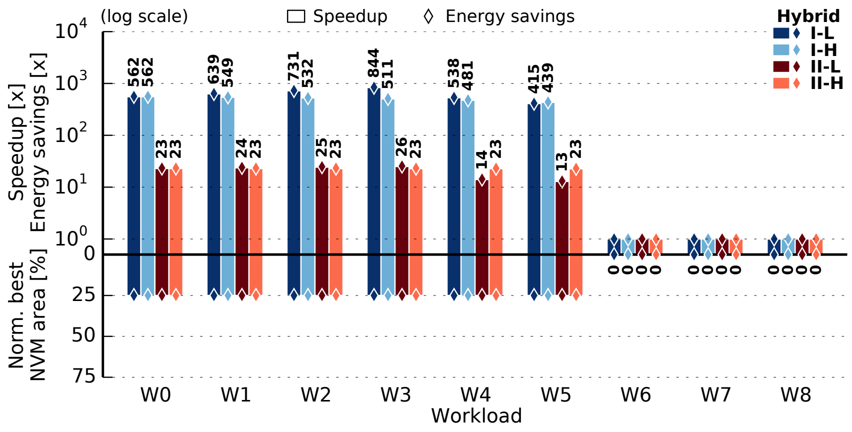


Figure 5.12: *Crystal for hybrids with HDD and* `CG/sjeng` *set*

Summary

Intuitively, hybrid memory offers speedups and energy savings compared to an equal-area DRAM-only memory if the benefit of reducing the number of disk accesses is greater than the overhead of page migrations between the hybrid partitions. However, for specific workloads such beneficial hybrid configurations are nontrivial: Speedups and energy savings can be relatively small (e.g., below 1.5×) and can significantly depend on DRA (e.g., the case of W2 in Figure 5.9). Crystal comes in handy in such situations and quickly identifies promising hybrid configurations.

For instance, Crystal shows how the bit density of NAND Flash enables hybrids that offer significant speedups and energy savings for W0-W4 in Figure 5.9, while the bit density of PCM is not high enough to enable hybrids offering any improvements for the same workloads. The higher speed and energy efficiency of PCM (compared to those of NAND Flash) offer no advantage is such cases. The higher endurance of PCM is not considered here, since NAND Flash serves as a representative technology in terms of bit density and access characteristics only. Memory technologies with similar bit densities and access characteristics but much higher endurance (e.g., Resistive Random Access Memory (RRAM) [59]) may gain maturity in the future.

For all the memory technologies, hybrids, and workloads considered in this study, hybrid configurations that minimize execution time also minimize execution energy, and speedups are similar to energy savings for each given hybrid and workload. Execution

Table 5.15: *Model parameters for sensitivity analysis of Crystal*

DRAM revision	F, G [8]	**Row buffer hit rate [%]**	0, 99
Non-memory power [W]	25, 50, 100	**Buffer disk writes**	No, Yes
Overlap T_{CPU} and T_{mem}	No, Yes	**Disk cache hit rate [%]**	0,50

time and energy follow the same trend for the technologies, since NVM access latencies and dynamic energies are worse than those of DRAM but better than those of disk. Ultimately, partitioning can be done for execution time even if the actual target metric is energy, and this further simplifies the use of Crystal.

5.3.4 Validation

I validate Crystal by means of sensitivity analysis, showing that partitioning results are insensitive to the implementation-dependent details represented indirectly, as is described in Section 5.1. I vary the model parameters in broad ranges covering a large space of execution scenarios. Table 5.15 shows the parameter values. I consider the two DRAM device revisions introduced in Section 3.3, a low and a high row buffer hit rate, three non-memory power values, toggling the *Buffer disk writes* and the *Overlap T_{CPU} and T_{mem}* parameters, and a low and a high disk cache hit rate, thus producing 96 combinations in total. In order to limit the size of the validation space, *Disk write propagation rate* is set to $(100 - \textit{Row buffer hit rate})\,\%$, since writes that hit in the row buffer of $M1$ are guaranteed to not propagate as writes to disk. Thus, the row buffer hit rates of 0% and 99% yield disk write propagation rates of 100% and 1%, respectively. I assume that the propagation rate of 100% represents the worst case in terms of execution time and energy, and that the rate of 1% represents the best case. Each of the eight hybrids is partitioned for W0-W8 from each of the six program sets (`mcf/sjeng`, `mcf/soplex`, `lbm/sjeng`, `lbm/soplex`, `CG/sjeng`, and `CG/soplex`) and each of the 96 parameter combinations. The results show that the best partitioning for each given hybrid and workload is stable for all the parameter combinations, as is discussed below by the example of three selected hybrid/workload combinations.

Figures 5.13 to 5.15 show results for W4 from the `lbm/sjeng` set and hybrids II-L, I-L, and III-L, respectively. The horizontal axis shows the 96 parameter combinations (0-95), encoded by the colors from the legend on the right. For instance, parameter combination 0 denotes DRAM revision F, the non-memory power of 25W, no overlapping of T_{CPU} and T_{mem}, the row buffer hit rate of 0%, no buffering of disk writes, and the

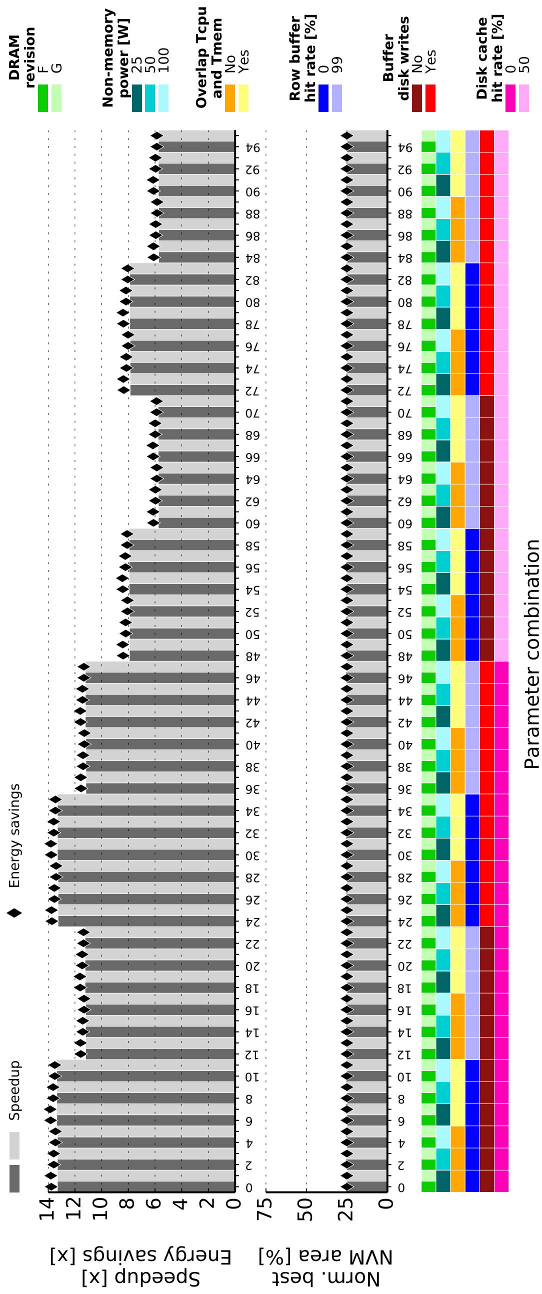


Figure 5.13: *Sensitivity of Crystal for hybrid II-L and W4 from* `lbm/sjeng` *set*

disk cache hit rate of 50%. The bottom parts of the graphs show the best configurations for each parameter combination as the best NVM area normalized to the total main memory area. The bars of alternating shades (darker for the even parameter combinations and lighter for the odd ones) show the best configurations found for execution time as the target metric, and the diamonds show those for execution energy as the target metric. The top parts of the graphs show corresponding improvements: speedups (by bars) and energy savings (by diamonds).

The improvements in Figures 5.13 to 5.15 are rather insensitive to DRAM revision, i.e., to the variation of DRAM electrical characteristics expressed as the operating currents in Table 5.12. In addition, the results are rather insensitive to non-memory power, that I vary from one-half to double the default value of 50W and observe relatively insignificant changes in energy savings. This is why, e.g., the inaccuracy of SSD idle power is tolerable (recall the remark in Section 5.3.2). The robustness of the best hybrid configuration (25% NVM in Figures 5.13 to 5.15) to such parameter variations indicates that Crystal can be applied early in the design process, when accurate estimates of system component characteristics are not yet available.

The speedups and energy savings offered by hybrid II-L in Figure 5.13 are most sensitive to the disk cache hit rate (parameter combinations 0-47 vs. 48-95) and to the row buffer hit rate (parameter combinations 0-11 vs. 12-23, 24-35 vs. 36-47, etc.). Disk cache hits significantly reduce the execution time and energy of the baseline, but do not affect those of the hybrid, since it provides enough main memory capacity to make the workload in-memory and thus eliminate its disk accesses. Hence the high disk cache hit rate results in the lower speedups and energy savings. The high row buffer hit rate yields the low disk write propagation rate, that significantly reduces the number of disk writes in the baseline system thus reducing its execution time and energy. However, the disk write propagation rate does not affect the execution time and energy of the hybrid, since the workload is in-memory w.r.t. its main memory capacity and disk is not accessed. At the same time, the high row buffer hit rate equally reduces the execution times and energies of the baseline and the hybrid systems. In addition, its effect is dwarfed by the disk accesses in the baseline system and the migrations between the DRAM and NAND Flash partitions in the hybrid. Hence the high row buffer hit rate results in the lower speedups and energy savings.

The improvements attainable by hybrid I-L in Figure 5.14 are sensitive to the overlap of T_{CPU} and T_{mem} (parameter combinations 0-5 vs. 6-11, 12-17 vs. 18-23, etc.), row buffer hit rate, and disk cache hit rate. Overlapping T_{CPU} and T_{mem} does not

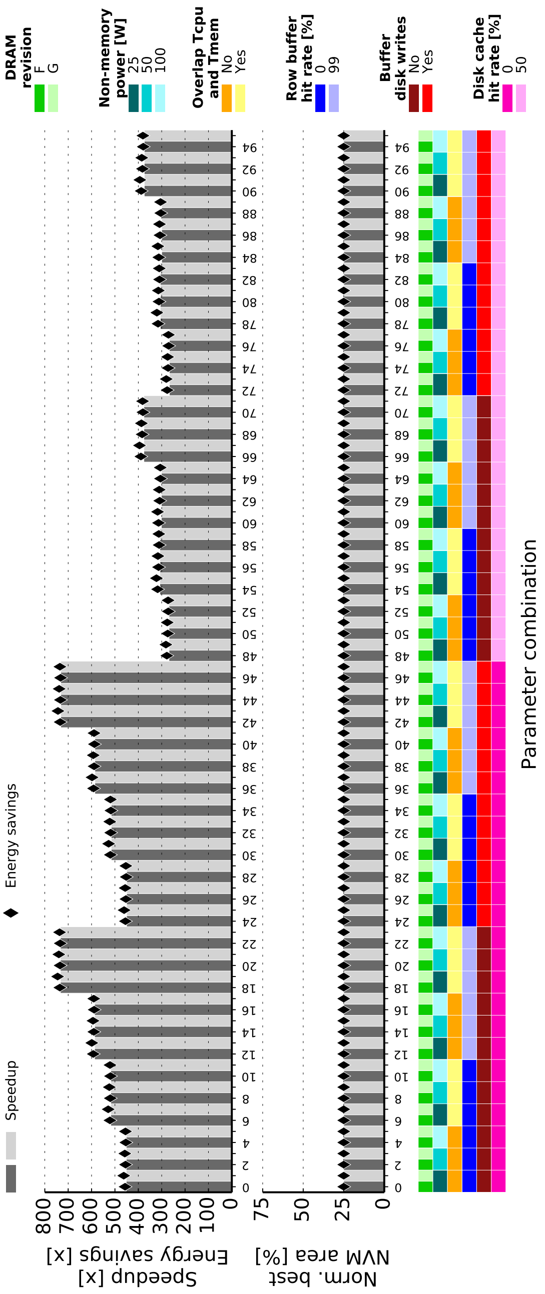


Figure 5.14: *Sensitivity of Crystal for hybrid I-L and W4 from* `lbm's jeng set`

significantly reduce baseline execution time and energy, because disk accesses dwarf the result of the overlap. On the contrary, the overlap does reduce the execution time and energy of the hybrid, since its main memory capacity is large enough to eliminate disk accesses. Hence enabling the parameter increases the improvements. Similarly, disk accesses dwarf the effect of high row buffer hit rate in the baseline system, even though the corresponding low disk write propagation rate significantly reduces the number of disk writes. But since disk is not accessed in the hybrid, and the migrations between the DRAM and PCM partitions do not dwarf the effect of the high row buffer hit rate, it significantly reduces the execution time and energy of the hybrid. Hence, unlike the above result for hybrid II-L, the high row buffer hit rate results in greater improvements.

The speedups and energy savings attainable by hybrid III-L in Figure 5.15 are sensitive to the overlap of T_{CPU} and T_{mem}, row buffer hit rate, disk write buffering (parameter combinations 0-23 vs. 24-47 and 48-71 vs. 72-95), and disk cache hit rate. Unlike the above case of hybrid I-L, disk write buffering reduces the improvements more than increasing disk cache hit rate does: Hitting in the SSD cache reduces the disk access latency not as dramatically as hitting in the HDD cache. The effect of buffering disk writes is greater in systems with SSD than in systems with HDD, because SSD accesses dwarf T_{CPU} and main memory accesses not as much as HDD accesses do.

The sensitivity analysis shows that the partitioning results are most sensitive to the number of accesses to main memory and disk initiated by the workload. Each disk access and each page migration between the DRAM and NVM partitions dwarf many execution details at higher levels in the system. The assumed model parameters do not affect the best partitioning even if varied in broad ranges. Though, they do affect speedups and energy savings within understandable margins. The best hybrid configurations for each hybrid and workload are stable among all the 96 parameter combinations. This holds both at the partitioning granularity of one DIMM and at a hypothetical, two times finer granularity (for brevity, I do not show more results than those in Figures 5.13 to 5.15 because they offer no new insights). Thus, the first-order nature of the models does not restrict the applicability of Crystal.

5.3.5 Related Work

Qureshi et al. [16] replace all DRAM devices with PCM devices and add a DRAM cache to create a hierarchical, replicating (inclusive) system. The DRAM cache does not add capacity and constitutes an area overhead. Main memory is backed by HDD with a

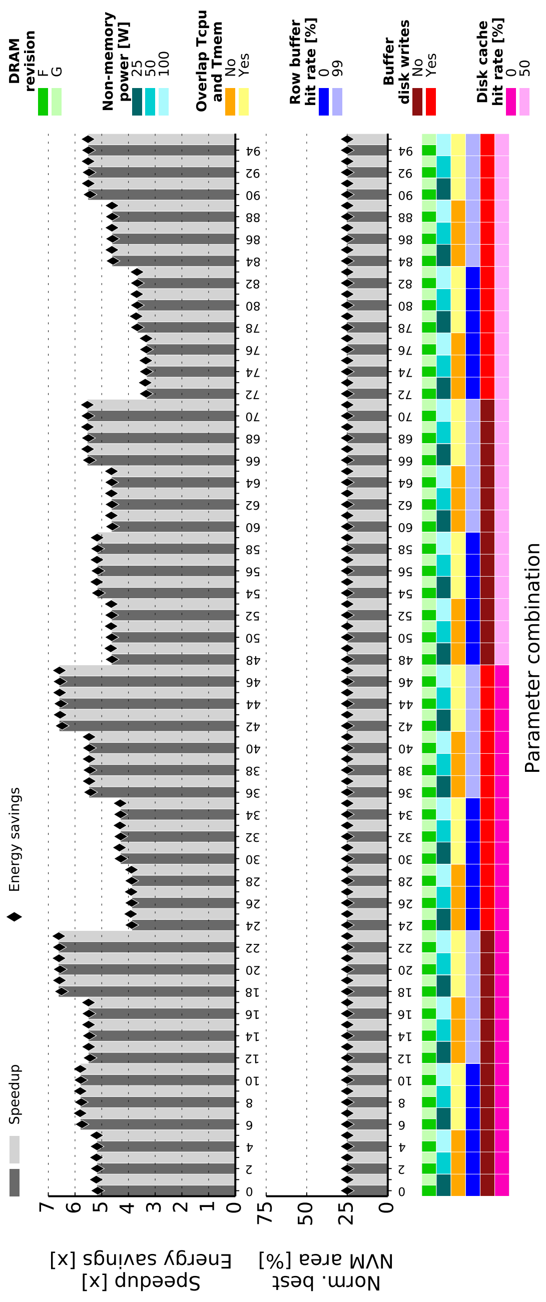


Figure 5.15: *Sensitivity of Crystal for hybrid III-L and W4 from* `lbm/sjeng` *set*

NAND Flash cache. Using a detailed simulator, the authors model candidate hybrids for not-in-memory (w.r.t. their DRAM-only baseline) workloads. The models in the heart of Crystal can be easily modified to represent hierarchical, replicating systems, and Crystal can be applied to evaluate more configurations with less computational effort.

Ekman and Stenstrom [14, 15] reduce main memory cost by replacing part of its fast DRAM with a less expensive, slower technology, and organizing hierarchical, migrating systems.They study in-memory workloads with a detailed simulator, identifying the fraction of each working set that must remain in fast DRAM in order to maintain execution times similar to those of the baseline system. Such partitioning can be done with Crystal by setting the goal to "for a fixed main memory capacity, minimize DRAM within a given degradation of execution time".

Ye et al. [13] study partitioning for in-memory workloads employing NVM with a broad range of access characteristics. Like Ekman and Stenstrom [14, 15], they set a goal to find a hybrid configuration with acceptable performance degradation. Unlike me, they partition main memory capacity, i.e., the total main memory capacity is fixed for all of their candidate configurations. They emulate hybrid memory on a real machine with a custom virtual machine monitor. The total main memory capacity is limited by that of the host machine, and Crystal avoids this limitation.

Jacob et al. [74] propose an analytic, first-order model for finding the number and the capacities of levels in an n-level, replicating (inclusive) memory hierarchy for a given cost (monetary budget). Unlike me, the authors impose no area constraint, assume equal bit densities of memory technologies, treat all accesses as reads, and roughly approximate miss curves abstracting away plateaus, that are crucial for partitioning (recall the example in Section 5.3.1).

Yavits et al. [75] study cache hierarchy partitioning in 3D chip multi-processors. For given area and power budgets, their analytic model finds the optimal number and area of the cache levels. Unlike me, the authors do not consider hybrid technology hierarchies and specialize the model on on-chip caches.

Choi et al. [76] propose a model for finding an optimal placement of program data in a hybrid that is flat (both the DRAM and NVM partitions are accessed directly). The model does not include disk and is thus applicable for in-memory workloads only. The authors partition main memory capacity (i.e., the total capacity is fixed) and do not consider area. Their model involves integer linear programming and, although not discussed by the authors, implies a limit on the working set size for which the optimal placement can be practically found. On the contrary, Crystal is holistic (models the entire system and is thus applicable for both in-memory and not-in-memory workloads) and practical (considers main memory area and is applicable for workloads with realistic working set sizes).

5.3.6 Concluding Remarks

Finding the right balance between DRAM and NVM is fundamental for hybrid main memory design. Detailed simulation or prototyping involve high effort impeding comprehensive design space exploration required for finding the best partitioning. I propose Crystal, an analytic approach to the design-time resource partitioning of hierarchical, migrating hybrid systems. Crystal employs simple models that deliver first-order estimates of system-level execution time and energy. I validate Crystal by means of sensitivity analysis and show that partitioning results are robust to the inaccuracies of the models. I make the following contributions and observations:

- Crystal helps designers identify promising design points on which to focus detailed evaluation via simulation or prototyping. E.g., Crystal shows how for a practical partitioning goal and specific workloads higher performance and energy efficiency can be achieved by employing NVM with the speed and energy consumption of NAND Flash instead of a much faster and more energy efficient NVM technology like PCM. This makes Crystal a valuable addition to the system designer's toolbox.
- Simple models and coarse estimates of system component characteristics are sufficient for finding the best hybrid configuration. This illustrates the robustness of the best configuration to variations of system characteristics and makes Crystal applicable early in the design process, when accurate numbers might be not yet available.
- For the current state of technologies, system-level execution time can be used for partitioning even if the actual target metric is system-level execution energy: Both metrics follow the same trend, and minimizing execution time minimizes execution energy. This speeds the partitioning process, since the model for execution time is simpler than that for energy.

Crystal is holistic, practical, and enables early and rapid partitioning, thus powering exhaustive design space exploration. To the best of my knowledge, Crystal is the first design-time hybrid memory partitioning method that is applicable for both in-memory and not-in-memory workloads.

5.4 Summary

Hybrid main memory increases complexity of already complex contemporary high-performance systems. Different memory technologies and ways to organize and manage them within hybrid main memory add new dimensions to the system design space. The simulation or prototyping of hybrid memory systems imply high effort and thus impede obtaining insights and finding the best solutions to such problems as DRP.

I have proposed using light-weight, analytic, system-level models for representing hybrid memory systems. The models provide first-order estimates of system-level execution time and energy. They power Rock, a performance model for hybrid systems, and Crystal, a exhaustive DRP method. Although execution time and energy estimates are first-order, their accuracy is sufficient for obtaining insights using Rock and finding the best area partitioning using Crystal.

6 Conclusion

The memory demand created by growing working set sizes of system workloads can be addressed by increasing the capacity of main memory. The type of workloads that benefit from main memory capacity increase is introduced in Section 2.3. When the area budget of main memory is fixed, its capacity can be increased by employing memory technologies that are bit-denser than the baseline main memory technology, DRAM. Such technologies are typically slower and less energy efficient than DRAM but still faster and more energy efficient than disk. In addition, they can be cheaper per bit than DRAM but more expensive than disk. Their prominent representatives are NVM technologies like PCM and NAND Flash, introduced in detail in Chapter 3. Combining DRAM with NVM into hybrid main memory has the potential of enjoying the advantages of both DRAM and NVM, i.e., the speed and energy efficiency of DRAM and the aggregate capacity of DRAM and NVM. However, in practice each access to NVM incurs a performance and energy penalty. Thus, hybrid main memory can offer speedups and energy savings (compared to an equal-area DRAM-only baseline) only on the system-level when its larger capacity helps to reduce the number of disk accesses. If a performance degradation and/or an energy efficiency degradation are acceptable and if NVM costs less per bit than DRAM, hybrid main memory can offer a lower cost compared to an equal-capacity DRAM-only baseline. The complexity of hybrid main memory design is that NVM introduces a new dimension to the system design space. The best amounts of different memory technologies employed for building the partitions of hybrid main memory and their organization are nontrivial. They depend on many factors, e.g.: workloads, the design goal, the characteristics of memory technologies, disk, and non-memory subsystems. The performance and energy efficiency of different hybrid memory designs can vary widely.

The complexity is well illustrated by the plurality of hybrid memory systems that have been proposed to date. The first problem is that this diversity of hybrid systems is large and lacks systematization. This complicates positioning new hybrid memory proposals within the existing body of work. The second problem is that there is a lack of quick, system-level models for exploring the large design space of hybrid memory systems. For instance, partitioning the area of main memory between DRAM and NVM is a fundamental hybrid memory design challenge that requires a method driven by such models. Using simulators implies significant implementation and computation efforts which impede extensive design space exploration. Likewise, prototyping implies substantial implementation effort and design space restrictions such as the total main memory capacity of the host machine. Simulation or prototyping are the right approaches for the detailed evaluation of the most promising hybrid configurations. However, quicker models are needed for enabling early and extensive design space exploration for finding such promising configurations.

The next section reviews the contributions of this thesis addressing the above problems and discusses its findings. Section 6.2 outlines the prospects of future work that will leverage the contribution of this thesis to Design-time Resource Partitioning (DRP) and will address such important hybrid memory design challenges as run-time resource allocation and Run-time Data Placement (RDP), introduced in the beginning of Chapter 5.

6.1 Contributions

The first contribution of this thesis is the taxonomy of hybrid memory systems presented in Chapter 4. It illustrates the diversity of hybrid systems not older than 10 years by classifying them by four key organizational criteria using the *TPN–L* notation: 1) the logical *T*opology, i.e., if hybrid memory is flat, such that its partitions can be accessed directly, or if it is hierarchical, such that partitions on lower levels can be accessed only via partitions on higher levels; 2) the *P*romotion policy, i.e., if data are migrated (moved) or replicated (copied) when placed into a partition with higher performance and/or energy efficiency; 3) the *N*umber of partitions; and 4) if the partitions are located on the same or on different physical *L*evels of the system, where the levels are: the processor chip/package, main memory channels, and after I/O. Using the notation, I identify seven unique organizations. Classifying them further by the types of memory technologies (presented in Chapter 3) and the physical locations of partitions, I identify 12 unique systems. I find that the most popular organizations are flat and migrating. This is understandable, since flat organizations allow implementing sophisticated RDP policies that decrease the performance and energy overhead of accessing lower partitions, and migrating organizations do not waste capacity as replicating ones do. Hierarchical organizations are well represented, too, by the five unique systems identified. Such systems are simpler to model than flat systems, and two have been prototyped [13, 17, 22]. For workloads that benefit from a main memory capacity increase (as described in Section 2.3), hierarchical organizations allow finding if hybrid main memory can help to attain speedups and/or energy savings on the system level (compared to a DRAM-only baseline of a lower capacity) when the overhead of accessing lower partitions is the largest. This way, they represent the worst case in terms of system-level execution time and energy. Flat organizations leverage these findings and further improve the performance and energy efficiency of hybrid systems by reducing the overhead of accessing lower partitions.

The second contribution of this thesis is the models for system-level execution time and energy of hierarchical hybrid systems presented in Section 5.1 and enabling early, rapid, and extensive design space exploration. The models embody the standard memory hierarchy equations but in the context of multi-level, hybrid main memory. They are analytic, first-order and thus are faster than memory system simulators. Implementation-dependent details are represented indirectly by the assumed model parameters. The models employ the workload methodology described in Chapter 2. It requires detailed program execution and analysis just once, when recording program profiles. The program profiles are reused throughout design space exploration thus reducing the computation effort. The models utilize the estimates of memory technology characteristics obtained with the device-level models derived from datasheets and tuned to the purposes of this thesis as described in Section 3.3. For instance, DRAM, PCM, and NAND Flash devices are modeled to share the same type of I/O interface.

The system-level models power Rock, the performance model for hybrid memory systems presented in Section 5.2. Rock provides first-order estimates of memory system throughput attainable for specific workloads. Rock is simple, quick, and insightful. For instance, Rock shows how partitioning the area of main memory between DRAM and NVM (such as PCM or NAND Flash) can significantly boost memory system throughput. In addition, Rock illustrates how the result of DRP can substantially depend on the choice of a Design-time Resource Allocation (DRA) policy. One prominent example described in Section 5.2.2 is when two hybrid systems with the same partition areas but different DRA policies (the low-utility and utility-agnostic policies introduced in Section 2.2) have a higher and a lower throughput, respectively, compared to the throughput of an equal-area DRAM-only baseline. Further, Rock shows how the throughput boost offered by a sophisticated RDP policy can be limited, because the total latency of accesses to main memory and/or disk can dwarf the overhead of accessing the NVM partition. Rock illustrates that in such cases even employing a hypothetical, ideal RDP policy (that completely eliminates the overhead of accessing the NVM partition) does not boost the memory system throughput significantly. In other words, it might be a better solution in such cases to employ a hierarchical organization and a basic RDP policy (e.g., those described in the beginning of Section 5.1), which are simpler to implement than a flat organization and a sophisticated RDP policy.

Rock shows a great potential of DRP for improving memory system performance and that the result of DRP substantially depends on workloads, DRA, and the characteristics of memory technologies. Thus, Rock motivates the final contribution of this thesis: Crystal, the DRP method presented in Section 5.3. Crystal is powered by the models for both system-level execution time and energy and finds for a given workload the partitioning of main memory area between DRAM and NVM that meets the design goal best. Crystal is simple yet holistic, quick, and practical. It frames partitioning as an optimization problem where the trends of execution time and energy matter more than absolute numbers, such that the first-order nature of the system-level models does not restrict its applicability. This is supported by an extensive validation of Crystal by means of sensitivity analysis, covering a large space of execution scenarios and showing that the partitioning results are robust to the model parameter variation in broad ranges. In addition, the sensitivity analysis shows that Crystal can be applied early in the design process when accurate estimates of system component characteristics are not yet available. For instance, the variation of system-level static power (including the static power of the CPU, main memory, and disk) is shown to have no impact on the best partitioning for a given workload. Likewise, the results of Crystal are robust to a variation in the power and dynamic energy of DRAM devices as is illustrated by the example of two different DRAM device revisions introduced in Section 3.3. This is important since the power variability in contemporary DRAM devices can be significant [77]. The conclusion is that simple models and coarse estimates of system component characteristics are sufficient for finding the best area partitioning between DRAM and NVM in hierarchical, migrating hybrid memory systems.

Next, I observe that for the current state of memory technologies (represented by DRAM, PCM, NAND Flash, SSD, and HDD in this thesis) system-level execution time can be used for partitioning even if the actual target metric is system-level execution energy. This is understandable, since PCM fits in the gap between DRAM and both SSD and HDD, and NAND Flash fits in the gap between DRAM and HDD without overlaps in terms of access latency and access dynamic energy. Using execution time as the target metric instead of energy further simplifies the use of Crystal, since the model for system-level execution time is simpler than that for energy.

Most importantly, Crystal identifies promising design points for detailed evaluation. It partitions hierarchical hybrid memories, that represent the worst case in terms of system-level execution time and energy compared to flat hybrid memories. Thus, the results of Crystal can be a starting point in building a hybrid main memory system. In addition, Crystal shows how for specific workloads memory technologies as slow and energy inefficient as NAND Flash can enable hybrid systems with higher performance and energy efficiency compared to those of hybrid systems employing technologies as fast and energy efficient as PCM. This highlights that for specific workloads bit density is a crucial characteristic that must be considered when designing hybrid main memory.

Design-time resource partitioning is the fundamental problem of hybrid memory system design, and this thesis addresses it by proposing Crystal. Resource allocation and data placement are considered but not investigated in detail in this thesis. They are the subject of my future work, described in the next section.

6.2 Future Work

Resource allocation is a memory system design challenge that is present both at design-time and at run-time. This thesis has considered several design-time resource allocation policies in order to investigate design-time resource partitioning. Run-time resource allocation is essential for dynamic optimization of main memory utilization. It has been investigated in the context of conventional DRAM-only systems [78]. However, run-time resource allocation for hybrid main memories is different in a number of respects. For instance, it has to be efficient for managing multiple partitions (instead of the single partition of conventional main memory). In addition, it has to be efficient for managing large capacities (enabled by hybrid memories). I am planning to propose a practical solution to run-time resource allocation for hierarchical, migrating hybrid systems employing a basic run-time data placement policy first. Next, I will try to complete the picture by proposing an adaptive run-time data placement policy in the context of the above run-time resource allocation solution.

Bibliography

[1] Int. Solid-State Circuits Conf., "ISSCC 2013 Tech Trends," `http://isscc.org/doc/2013/2013_Trends.pdf`, 2013.

[2] Int. Technology Roadmap for Semiconductors, "Process Integration, Devices, and Structures," `http://www.itrs.net/Links/2011itrs/2011Chapters/2011PIDS.pdf`, 2011.

[3] S. Raoux, G. Burr, M. Breitwisch, C. Rettner, Y. Chen, R. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H. L. Lung, and C. Lam, "Phase-change random access memory: A scalable technology," *IBM J. of Research and Development*, vol. 52, no. 4/5, pp. 465–479, July 2008.

[4] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, *Phase Change Memory: From Devices to Systems*, vol. 6 of *Synthesis Lectures on Computer Architecture*, Nov. 2011.

[5] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*, 2007.

[6] Int. Technology Roadmap for Semiconductors, "Executive Summary," `www.itrs.net/Links/2011itrs/2011Chapters/2011ExecSum.pdf`, 2011.

[7] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Laboratories Technical Report No. 2009-085, 2009.

[8] Micron Technology, Inc., "Micron® 1Gb: $\times 4$, $\times 8$, $\times 16$ DDR3 SDRAM features," Datasheet, `http://www.micron.com`, 2006.

[9] Micron Technology, Inc., "Micron® 16Gb, 32Gb, 64Gb, 128Gb asynchronous/synchronous NAND features," Datasheet, `http://www.micron.com`, 2009.

[10] B. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. Int. Symp. on Computer Architecture*, June 2009, pp. 2–13.

[11] Tom's Hardware, "SSD Charts 2013," `http://www.tomshardware.com/charts/ssd-charts-2013/benchmarks,129.html`, 2013.

[12] Tom's Hardware, "HDD Charts 2013," `http://www.tomshardware.com/charts/hdd-charts-2013/benchmarks,134.html`, 2013.

[13] D. Ye, A. Pavuluri, C. Waldspurger, B. Tsang, B. Rychlik, and S. Woo, "Prototyping a hybrid main memory using a virtual machine monitor," in *Proc. Int. Conf. on Computer Design*, Oct. 2008, pp. 272–279.

[14] M. Ekman and P. Stenstrom, "A case for multi-level main memory," in *Workshop on Memory Performance Issues, co-located with Int. Symp. on Computer Architecture*, June 2004, pp. 1–8.

[15] M. Ekman and P. Stenstrom, "A cost-effective main memory organization for future servers," in *Proc. Int. Parallel and Distributed Processing Symp.*, Apr. 2005, pp. 1–10.

[16] M. Qureshi, V. Srinivasan, and J. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. Int. Symp. on Computer Architecture*, June 2009, pp. 24–33.

[17] K. Sudan, A. Badam, and D. Nellans, "NAND-Flash: Fast storage or slow memory?," in *Non-Volatile Memory Workshop*, Mar. 2012, pp. 1–2.

[18] L. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *Proc. Int. Conf. on Supercomputing*, May 2011, pp. 85–95.

[19] K. Sudan, K. Rajamani, W. Huang, and J. B. Carter, "Tiered memory: An iso-power memory architecture to address the memory power wall," *Transactions on Computers*, vol. 99, no. PrePrints, 2012.

[20] J. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for NVM+DRAM hybrid main memory," in *Workshop on Hot Topics in Operating Systems*, May 2009, pp. 1–5.

[21] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, "Row buffer locality-aware data placement in hybrid memories," SAFARI Technical Report No. 2011-005, Sept. 2011.

[22] A. Badam and V. Pai, "SSDAlloc: Hybrid SSD/RAM memory management made easy," in *Proc. Symp. on Networked Systems Design and Implementation*, Mar. 2011, pp. 1–14.

[23] S. Phadke and S. Narayanasamy, "MLP aware heterogeneous memory system," in *Proc. Design, Automation and Test in Europe*, Mar. 2011, pp. 1–6.

[24] K. Fang, L. Chen, Z. Zhang, and Z. Zhu, "Memory architecture for integrating emerging memory technologies," in *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, Oct. 2011, pp. 403–412.

[25] A. Bivens, P. Dube, M. Franceschini, J. Karidis, L. Lastras, and M. Tsao, "Architectural design for next generation heterogeneous memory systems," in *Int. Memory Workshop*, May 2010, pp. 1–4.

[26] P. Dube, M. Tsao, D. Poff, L. Zhang, and A. Bivens, "Program behavior characterization in large memory systems," in *Proc. Int. Symp. on Performance Analysis of Systems Software*, Mar. 2010, pp. 113–114.

[27] X. Dong, Y. Xie, N. Muralimanohar, and N. Jouppi, "Simple but effective heterogeneous main memory with on-chip memory controller support," in *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Nov. 2010, pp. 1–11.

[28] W. Zhang and T. Li, "Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures," in *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, Sept. 2009, pp. 101–112.

[29] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache management," *Computer Architecture Letters*, vol. 11, no. 2, pp. 61–64, 2012.

[30] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montano, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *Proc. Int. Symp. on Computer Architecture*, June 2010, pp. 153–162.

[31] D. Knyaginin, S. A. McKee, and G. N. Gaydadjiev, "A hybrid main memory systems taxonomy," in *Memory Architecture and Organization Workshop, co-located with Embedded Systems Week*, Oct. 2012, pp. 1–6.

[32] D. Knyaginin, G. N. Gaydadjiev, and P. Stenstrom, "Crystal: A design-time resource partitioning method for hybrid main memory," in *Workshop on Reproducible*

Research Methodologies, co-located with Int. Symp. on High Performance Computer Architecture, Feb. 2014, pp. 1–6.

[33] P. J. Denning, “The working set model for program behavior,” *Commun. ACM*, vol. 11, no. 5, pp. 323–333, May 1968.

[34] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, “Evaluation techniques for storage hierarchies,” *IBM Systetms J.*, vol. 9, no. 2, pp. 78–117, June 1970.

[35] 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 Computer Architecture News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[36] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, “Pin: Building customized program analysis tools with dynamic instrumentation,” in *Proc. Conf. on Programming Language Design and Implementation*, June 2005, pp. 190–200.

[37] N. Nethercote and J. Seward, “Valgrind: A framework for heavyweight dynamic binary instrumentation,” in *Proc. Conf. on Programming Language Design and Implementation*, June 2007, pp. 89–100.

[38] SPEC, “SPEC CPU2006,” `http://www.spec.org/cpu2006`, 2006.

[39] NASA, “NAS Parallel Benchmarks,” `http://www.nas.nasa.gov/Resources/Software/npb.html`, 2010.

[40] G. Hamerly, E. Perelman, J. Lau, and B. Calder, “Simpoint 3.0: Faster and more flexible program analysis,” in *J. of Instruction Level Parallelism*, 2005.

[41] JEDEC Solid State Technology Association, “DDR3 SDRAM standard,” `http://www.jedec.org/standards-documents/docs/jesd-79-3d`, July 2010.

[42] Micron Technology, Inc., “Micron® 64Gb, 128Gb, 256Gb, 512Gb asynchronous/synchronous NAND features,” Datasheet, `http://www.micron.com`, 2009.

[43] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan, “Efficient scrub mechanisms for error-prone emerging memories,” in *Proc. Int. Symp. on High Performance Computer Architecture*, Feb. 2012, pp. 1–12.

[44] N. H. Seong, S. Yeo, and H.-H. S. Lee, "Tri-level-cell phase change memory: Toward an efficient and reliable memory system," in *Proc. Int. Symp. on Computer Architecture*, June 2013, pp. 440–451.

[45] Micron Technology, Inc., "Micron® LPDDR2 PCM $\times$16 + LPDDR2 SDRAM $\times$16," Datasheet, `http://www.micron.com`, 2010.

[46] Micron Technology, Inc., "Micron® 128Mb P8P parallel PCM features," Datasheet, `http://www.micron.com`, 2005.

[47] Everspin Technologies, "2M$\times$8 MRAM Memory," Datasheet, `http://www.everspin.com`, 2013.

[48] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, "A novel nonvolatile memory with spin torque transfer magnetization switching: spin-RAM," in *Proc. Int. Electron Devices Meeting*, Dec. 2005, pp. 459–462.

[49] S. Chung, K.-M. Rho, S.-D. Kim, H.-J. Suh, D.-J. Kim, H. Kim, S. Lee, J.-H. Park, H.-M. Hwang, S.-M. Hwang, J. Y. Lee, Y.-B. An, J.-U. Yi, Y.-H. Seo, D.-H. Jung, M.-S. Lee, S.-H. Cho, J.-N. Kim, G.-J. Park, G. Jin, A. Driskill-Smith, V. Nikitin, A. Ong, X. Tang, Y. Kim, J.-S. Rho, S.-K. Park, S.-W. Chung, J.-G. Jeong, and S. J. Hong, "Fully integrated 54nm STT-RAM with the smallest bit cell dimension for high density memory application," in *Proc. Int. Electron Devices Meeting*, Dec. 2010, pp. 12.7.1–12.7.4.

[50] H. Lee, P. Chen, T. Y. Wu, Y. Chen, C. Wang, P. Tzeng, C. H. Lin, F. Chen, C. Lien, and M. J. Tsai, "Low power and high speed bipolar switching with a thin reactive Ti buffer layer in robust HfO2 based RRAM," in *Proc. Int. Electron Devices Meeting*, Dec. 2008, pp. 1–4.

[51] H. Lee, Y. Chen, P. Chen, P. Gu, Y. Hsu, S. Wang, W. Liu, C. Tsai, S. Sheu, P. Chiang, W. Lin, C. Lin, W. Chen, F. Chen, C. Lien, and M. Tsai, "Evidence and solution of over-RESET problem for HfOX based resistive memory with sub-ns switching speed and high endurance," in *Proc. Int. Electron Devices Meeting*, Dec. 2010, pp. 1–4.

[52] S. R. Lee, Y.-B. Kim, M. Chang, K. M. Kim, C. B. Lee, J. H. Hur, G.-S. Park, D. Lee, M.-J. Lee, C.-J. Kim, U.-I. Chung, I.-K. Yoo, and K. Kim, "Multi-level switching of triple-layered TaOx RRAM with excellent reliability for storage class memory," in *Proc. Symp. on VLSI Technology*, June 2012, pp. 71–72.

[53] Fujitsu Semiconductor Ltd., "Fujitsu FRAM Guide Book," `http://www.fujitsu.com`, 2005.

[54] Fujitsu Semiconductor Ltd., "Fujitsu FRAM 1Mb (128K$\times$8)," Datasheet, `http://www.fujitsu.com`, 2011.

[55] H. Shiga, D. Takashima, S. Shiratake, K. Hoya, T. Miyakawa, R. Ogiwara, R. Fukuda, R. Takizawa, K. Hatsuda, F. Matsuoka, Y. Nagadomi, D. Hashimoto, H. Nishimura, T. Hioka, S. Doumae, S. Shimizu, M. Kawano, T. Taguchi, Y. Watanabe, S. Fujii, T. Ozaki, H. Kanaya, Y. Kumura, Y. Shimojo, Y. Yamada, Y. Minami, S. Shuto, K. Yamakawa, S. Yamazaki, I. Kunishima, T. Hamamoto, A. Nitayama, and T. Furuyama, "A 1.6GB/s DDR2 128Mb chain FeRAM with scalable octal bitline and sensing schemes," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 2009, pp. 464–465.

[56] M. Qureshi, M. Franceschini, and L. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *Proc. Int. Symp. on High Performance Computer Architecture*, Jan. 2010, pp. 1–11.

[57] Y. Itoh, M. Momodomi, R. Shirota, Y. Iwata, R. Nakayama, R. Kirisawa, T. Tanaka, K. Toita, S. Inoue, and F. Masuoka, "An experimental 4Mb CMOS EEPROM with a NAND structured cell," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 1989, pp. 134–135.

[58] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND Flash Memories*, 2010.

[59] T.-Y. Liu, T. H. Yan, R. Scheuerlein, Y. Chen, J. Lee, G. Balakrishnan, G. Yee, H. Zhang, A. Yap, J. Ouyang, T. Sasaki, S. Addepalli, A. Al-Shamma, C.-Y. Chen, M. Gupta, G. Hilton, S. Joshi, A. Kathuria, V. Lai, D. Masiwal, M. Matsumoto, A. Nigam, A. Pai, J. Pakhale, C. H. Siau, X. Wu, R. Yin, L. Peng, J. Y. Kang, S. Huynh, H. Wang, N. Nagel, Y. Tanaka, M. Higashitani, T. Minvielle, C. Gorla, T. Tsukamoto, T. Yamaguchi, M. Okajima, T. Okamura, S. Takase, T. Hara, H. Inoue, L. Fasoli, M. Mofidi, R. Shrivastava, and K. Quader, "A 130.7mm2 2-layer 32Gb ReRAM memory device in 24nm technology," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 2013, pp. 210–211.

[60] Open NAND Flash Interface Working Group, "Open NAND Flash interface specification," `http://onfi.org/specifications`, 2011.

[61] G. Campardo, R. Micheloni, and D. Novosel, *VLSI-Design of Non-Volatile Memories*, 2005.

[62] Numonyx, B.V., “Numonyx® Axcell™ P30-65nm Flash memory 512-Mbit, 1-Gbit, 2-Gbit,” Datasheet, `http://www.micron.com`, 2010.

[63] Micron Technology, Inc., “System power calculators,” `http://www.micron.com/products/support/power-calc`, 2013.

[64] Micron Technology, Inc., “Calculating memory system power for DDR,” Technical Note, `http://www.micron.com`, 2001.

[65] Micron Technology, Inc., “Calculating memory system power for DDR2,” Technical Note, `http://www.micron.com`, 2004.

[66] Micron Technology, Inc., “Calculating memory system power for DDR3,” Technical Note, `http://www.micron.com`, 2007.

[67] D. Schmidt and N. Wehn, “DRAM power management and energy consumption: A critical assessment,” in *Proc. Symp. on Integrated Circuits and System Design*, Aug. 2009, pp. 1–5.

[68] Micron Technology, Inc., “DDR3 ZQ calibration,” Technical Note, `http://www.micron.com`, 2008.

[69] Micron Technology, Inc., “Increasing NAND Flash performance overview,” Technical Note, `http://www.micron.com`, 2004.

[70] T. Vogelsang, “Understanding the energy consumption of dynamic random access memories,” in *Proc. Int. Symp. on Microarchitecture*, Dec. 2010, pp. 363–374.

[71] K. Chandrasekar, B. Akesson, and K. Goossens, “Improved power modeling of DDR SDRAMs,” in *Proc. Conf. on Digital System Design*, Aug. 2011, pp. 99–108.

[72] X. Dong, C. Xu, Y. Xie, and N. Jouppi, “NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 7, pp. 994–1007, July 2012.

[73] S. Williams, A. Waterman, and D. Patterson, “Roofline: An insightful visual performance model for multicore architectures,” *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.

[74] B. Jacob, P. Chen, S. Silverman, and T. Mudge, “An analytical model for designing memory hierarchies,” *Computers, IEEE Transactions on*, vol. 45, no. 10, pp. 1180–1194, Oct. 1996.

[75] L. Yavits, A. Morad, and R. Ginosar, “3D cache hierarchy optimization,” in *Proc. Int. 3D Systems Integration Conf.*, Oct. 2013, pp. 1–5.

[76] J.-H. Choi, S.-M. Kim, C. Kim, K.-W. Park, and K. H. Park, “OPAMP: Evaluation framework for optimal page allocation of hybrid main memory architecture,” in *Proc. Int. Conf. on Parallel and Distributed Systems*, Dec. 2012, pp. 620–627.

[77] M. Gottscho, A. Kagalwalla, and P. Gupta, “Power variability in contemporary DRAMs,” *Embedded Systems Letters*, vol. 4, no. 2, pp. 37–40, June 2012.

[78] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, “Dynamic tracking of page miss ratio curve for memory management,” in *Proc. Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004, pp. 177–188.