UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA



TESIS DOCTORAL

Data Placement in NVM Hybrid Scratchpad-Cache Hierarchies

Asignación de datos en jerarquías híbridas scratchpad-cache con NVM

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Tommaso Marinelli

DIRECTORES

José Ignacio Gómez Pérez
Francky Catthoor

UCM DOCTORAL SCHOOL
Facultad de Informática

UNIVERSIDAD COMPLUTENSE MADRID

KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Data Placement in NVM Hybrid Scratchpad-Cache Hierarchies

ES: *Asignación de datos en jerarquías híbridas scratchpad-cache con NVM*

NL: *Het plaatsen van data in een hybride geheugenhiërarchie van non-volatiele caches en scratchpads*

Tommaso Marinelli

Supervisors:
Prof. José Ignacio Gómez Pérez
Prof. Francky Catthoor

Dissertation presented in partial fulfillment of the requirements for the dual degree of *Doctor en Ingeniería Informática (PhD)* at Universidad Complutense de Madrid and *Doctor of Engineering Science (PhD): Electrical Engineering* at KU Leuven

May 2024



UNIVERSIDAD COMPLUTENSE MADRID

KU LEUVEN

Universidad Complutense de Madrid - Programa de Doctorado en Ingeniería Informática, Facultad de Informática
KU Leuven - Doctoral Programme in Engineering Science, Faculty of Engineering Science

Data Placement in NVM Hybrid Scratchpad-Cache Hierarchies

ES: *Asignación de datos en jerarquías híbridas scratchpad-cache con NVM*
NL: *Het plaatsen van data in een hybride geheugenhiërarchie van non-volatiele caches en scratchpads*

Tommaso MARINELLI

Examination committee:
Prof. José Ignacio Gómez Pérez, supervisor
Prof. Francky Catthoor, supervisor
Prof. Luis Piñuel Moreno
Prof. Wim Dehaene
Prof. Katzalin Olcoz Herrero
Prof. Jesús Javier Resano Ezcaray
(Universidad de Zaragoza)
Prof. Lieven Eeckhout
(Universiteit Gent)

Dissertation presented in partial fulfillment of the requirements for the dual degree of *Doctor en Ingeniería Informática (PhD)* at Universidad Complutense de Madrid and *Doctor of Engineering Science (PhD): Electrical Engineering* at KU Leuven

May 2024

© 2024 Universidad Complutense de Madrid – Facultad de Informática
Autopublicado, Tommaso Marinelli, C/ Profesor José García Santesmases 9, 28040 Madrid (Spain)

© 2024 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Tommaso Marinelli, Kasteelpark Arenberg 1 bus 2200, B-3001 Leuven (Belgium)

This thesis has been partially supported by the EU (FEDER) and Spanish MINECO (grant No. RTI2018-093684-B-I00), the Spanish CM (grant No. S2018/TCS-4423), the Spanish MCIN/AEI/10.13039/501100011033 and the ERDF (grant No. PID2021-123041OB-I00).

UNIVERSIDAD COMPLUTENSE MADRID

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña. Tommaso Marinelli,
estudiante en el Programa de Doctorado en Ingeniería Informática (RD 99/2011),
de la Facultad de Informática de la Universidad Complutense de Madrid, como autor/a de la tesis presentada para la obtención del título de Doctor y titulada:

Data Placement in NVM Hybrid Scratchpad-Cache Hierarchies

Asignación de datos en jerarquías híbridas scratchpad-cache con NVM

y dirigida por: José Ignacio Gómez Pérez, Francky Catthoor

DECLARO QUE:

La tesis es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, de acuerdo con el ordenamiento jurídico vigente, en particular, la Ley de Propiedad Intelectual (R.D. legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, modificado por la Ley 2/2019, de 1 de marzo, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), en particular, las disposiciones referidas al derecho de cita.

Del mismo modo, asumo frente a la Universidad cualquier responsabilidad que pudiera derivarse de la autoría o falta de originalidad del contenido de la tesis presentada de conformidad con el ordenamiento jurídico vigente.

En Madrid, a 15 de diciembre de 2023

SIGN



Fdo.: _____

Esta DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD debe ser insertada en la primera página de la tesis presentada para la obtención del título de Doctor.

Acknowledgements

The journey towards this PhD has been long, but rewarding. For this achievement, there is a series of people that I naturally want to thank.

First of all, I thank my supervisors, Prof. Gómez Pérez and and Prof. Catthoor, for the guidance and assistance I have received during all these years. Having two points of view on my work and potential directions has helped me approach things with a deeper critical thinking. I can only be grateful for the time that they have dedicated to checking my progresses and revising my manuscripts, other than for the support in always complex administrative tasks. But most of all, I can feel they really care about the people they have around.

I also want to thank my colleagues, with whom I've shared my day to day research life, but also many moments outside of the university. Their company made this journey a bit more lightweight and fun. During my studies in Madrid and in Leuven I have met many wonderful people, and to thank all of them would result in a very long list. I am confident that they know how much they have been important to me. I am also thankful for the long-lasting friendships that I have maintained despite the distance from my hometown, Torino.

Last but not least, I want to thank my family. They have always supported me in every decision I have taken, including when I chose to move abroad to pursue my life goals, and when I began this PhD adventure. Their love, which I fully reciprocate, and their teachings have contributed to make me the person I am.

Abstract

Traditional memory technologies are facing significant limitations as the scaling process advances. The SRAM technology, typically used in different levels of on-chip cache hierarchies, is starting to show integration problems and a high static power consumption due to leakage currents. In recent years, several non-volatile memory technologies have been developed (PCM, ReRAM, STT-MRAM, SOT-MRAM, etc.) which solve some of the issues of traditional technologies but with important trade-offs, especially in latency and write energy. In this research thesis, the candidate has explored different solutions to create an efficient memory system architecture. The problem has been approached by mixing different elements and creating heterogeneous memory organizations, both from a technology point of view (SRAM and STT-MRAM) and from a system management point of view (hardware controlled cache and software controlled scratchpad). The research has been accompanied by a detailed feasibility study of the implemented changes, analyzing the energy and performance impact with standard benchmark sets. In the case of hybrid cache-scratchpad architectures, a custom data assignment policy has been developed as well, to decide which portions of data would best benefit from being manually managed. This work has required to setup an extensive profiling methodology, to extract memory access information from target applications, with the goal of detecting patterns and metrics to use for the placement decisions. Furthermore, it has been assumed that the designer does not necessarily have access to the original source code of applications, which entails operating on binaries for both the profiling and policies enforcement phases. All the proposed modifications have been evaluated in the context of general-purpose embedded systems with

consumption constraints, such as mobile devices, edge nodes or gateways. High level software device modelling and system simulation tools have been employed for the experimental part, with extensive modifications to fully represent the target architectures. In all the proposed configurations, it has been shown that it is possible to obtain substantial energy gains, with acceptable typical-case performance penalization.

ES: Resumen

Las tecnologías de memoria tradicionales se enfrentan a limitaciones significativas a medida que avanza el proceso de escalado. La tecnología SRAM, típicamente utilizada en diferentes niveles de jerarquías de caché en el chip, está empezando a mostrar problemas de integración y un alto consumo estático de energía debido a corrientes de fuga. En los últimos años, se han desarrollado varias tecnologías de memoria no volátil (PCM, ReRAM, STT-MRAM, SOT-MRAM, etc.) que resuelven algunos de los problemas de las tecnologías tradicionales, pero con importantes contrapartidas, especialmente en latencia y energía de escritura. En este trabajo doctoral, el candidato ha explorado diferentes soluciones para crear una arquitectura eficiente de sistema de memoria. El problema se abordó mezclando diferentes elementos y creando organizaciones de memoria heterogéneas, tanto desde un punto de vista tecnológico (SRAM y STT-MRAM) como desde un punto de vista de la gestión del sistema (la tradicional *caché* controlada por hardware frente a una *scratchpad* controlada por software). La investigación incluye un detallado estudio de viabilidad de los cambios implementados, analizando el impacto en energía y rendimiento con conjuntos de pruebas estándar. En el caso de arquitecturas híbridas de caché-scratchpad, también se desarrolló una política personalizada de asignación de datos para decidir qué porciones de datos se beneficiarían mejor de una gestión software. Este trabajo ha requerido establecer una metodología de perfilado para extraer automáticamente información de los patrones de acceso a memoria de los distintos objetos de una aplicación, con el fin de desarrollar métricas que guíen las decisiones de emplazamiento de dichos objetos. Además, se ha asumido que el diseñador no tiene necesariamente acceso al código fuente original de

las aplicaciones, lo que implica operar en binarios tanto para el perfilado como para las fases de aplicación de políticas. Todas las modificaciones propuestas se han evaluado en el contexto de sistemas empotrados de propósito general con restricciones de consumo, como dispositivos móviles o dispositivos de *edge-computing*. Para la parte experimental, se utilizaron herramientas de modelado de dispositivos de software de alto nivel y herramientas de simulación de sistemas, con modificaciones extensas para representar completamente las arquitecturas objetivo. En todas las configuraciones propuestas, se ha demostrado que es posible obtener ganancias sustanciales de energía, con una penalización aceptable en el rendimiento en casos típicos.

NL: Beknopte samenvatting

Traditionele geheugentechnologieën worden geconfronteerd met aanzienlijke beperkingen naarmate het schaalproces vordert. De SRAM-technologie, doorgaans gebruikt in verschillende niveaus van on-chip cache-hiërarchieën, begint integratieproblemen te vertonen en heeft een hoog statisch energieverbruik als gevolg van toenemende lekstromen. In de afgelopen jaren zijn verschillende niet-volatiele geheugentechnologieën ontwikkeld (PCM, ReRAM, STT-MRAM, SOT-MRAM, enz.) die sommige problemen van traditionele technologieën oplossen, maar met belangrijke compromissen, vooral op het gebied van toegangstijd en schrijfenergie. In deze onderzoeksproefschrift heeft de kandidaat verschillende oplossingen onderzocht om een efficiënte architectuur voor geheugensystemen te creëren. Het probleem is benaderd door verschillende elementen te combineren en heterogene geheugenorganisaties te creëren, zowel vanuit een technologisch oogpunt (SRAM en STT-MRAM) als vanuit een systeembeheer oogpunt (een cache aangestuurd door hardware en een scratchpad aangestuurd door software). Het onderzoek is vergezeld van een gedetailleerde haalbaarheidsstudie van de geïmplementeerde veranderingen, waarbij de energie- en prestatie-impact is geanalyseerd met standaard benchmarksets. In het geval van hybride cache-scratchpad-architecturen is ook een aangepast beleid voor gegevenstoewijzing ontwikkeld om te beslissen welke delen van de gegevens het meest zouden profiteren van handmatig beheer. Voor dit werk moest een uitgebreide profileringmethodologie worden opgezet om informatie over geheugentoegang te extraheren uit doeltoepassingen, met als doel patronen en metingen te detecteren die kunnen worden gebruikt voor de plaatsingsbeslissingen. Bovendien is ervan uitgegaan dat de ontwerper niet

noodzakelijkerwijs toegang heeft tot de originele broncode van toepassingen, wat inhoudt dat er wordt gewerkt met binaire bestanden voor zowel de profilering als de handhaving van beleid. Alle voorgestelde wijzigingen zijn geëvalueerd in de context van algemene ingebedde systemen met energiebeperkingen, zoals mobiele apparaten, edge nodes of gateways. Voor het experimentele gedeelte zijn tools voor modellering van softwareapparaten op hoog niveau en systematische simulatietools gebruikt, met uitgebreide aanpassingen om de doelarchitecturen volledig weer te geven. In alle voorgestelde configuraties is aangetoond dat het mogelijk is aanzienlijke energiewinsten te behalen, met een acceptabele prestatievermindering.

List of Abbreviations

AI Artificial Intelligence 1

ALU Arithmetic Logic Unit 5

BBV Basic Block Vector 46

CAM Content-Addressable Memory 19

DBI Dynamic Binary Instrumentation 65, 68

DMA Direct Memory Access 75, 76

DRAM Dynamic Random-Access Memory 23

DTCO Design-Technology Co-Optimization 5

DVFS Dynamic Voltage and Frequency Scaling 7

EDP Energy-Delay Product 57

FLOPS Floating Point Instructions per Second 2

FPU Floating Point Unit 5

HPC High Performance Computing 2

ILP Integer Linear Programming 32

IoT Internet of Things 2, 73

ISA Instruction Set Architecture 42

ITRS International Technology Roadmap for Semiconductors 57

LLC Last-Level Cache 13, 16, 27, 28, 30, 31, 36, 44, 47–50, 52, 54, 55, 57, 61, 63, 86, 87, 89

LRU Least Recently Used 44

LSQ Load-Store Queue 43

ML Machine Learning 1

MMIO Memory-Mapped I/O 75, 78

MMU Memory Management Unit 32

MPKI Misses Per Kilo Instructions 47

MRAM Magnetoresistive RAM 25, 26, 86

MSHRs Miss Status Handling Registers 20, 37, 44

MTJ Magnetic Tunnel Junction 25, 36, 39

PCRAM Phase Change RAM 29

ReRAM Resistive RAM 29

ROB Reorder Buffer 43

RTL Register-Transfer Level 10

SOT-MRAM Spin-Orbit Torque MRAM 25, 27, 28, 89

SPM Scratchpad Memory 12, 21, 70, 71, 73–78, 81, 82, 88, 90

SRAM Static Random-Access Memory 11, 12, 23–27, 30, 31, 35, 36, 38, 39, 41, 42, 44, 47, 54–57, 61, 63, 87, 88

STCO System-Technology Co-Optimization 5

STT-MRAM Spin-Transfer Torque MRAM 11, 12, 25–28, 30, 31, 36–42, 44, 48, 50, 52, 54–57, 61, 63, 86, 87, 89

TLB Translation Lookaside Buffer 19

TSV Through Silicon Via 24

VCMA-MRAM Voltage-Controlled Magnetic Anisotropy MRAM 26, 89

Contents

Abstract v

ES: Resumen vii

NL: Beknopte samenvatting ix

List of Abbreviations xi

Contents xv

List of Figures xix

List of Tables xxi

1 Introduction 1

1.1 Context 1

1.1.1 Emerging applications and trends in computing 1

1.1.2 The limits of technological scaling 2

1.1.3 Microarchitectural improvements 4

1.1.4 The power and energy wall 6

1.1.5 Elements of typical computer systems 7

1.1.6 CPU caches 9

1.1.7 Architectural modeling and simulation 9

1.2 Motivation 11

1.3 Thesis contributions 12

1.4 Organization 13

2 Background **15**
2.1 Cache architecture and organization 15
2.1.1 General concepts 16
2.1.2 Temporal and spatial locality 17
2.1.3 Tags and data 18
2.1.4 Placement policies 18
2.1.5 Cache coherency 19
2.1.6 Non-blocking caches 20
2.2 Scratchpad memories 21
2.3 Memory technologies 23
2.3.1 The SRAM technology 23
2.3.2 Technology requirements for caches 24
2.3.3 Emerging memory technologies - MRAMs 25

3 Related Work **29**
3.1 STT-MRAM and hybrid caches 29
3.2 Scratchpad memories and data placement 31
3.3 Application profiling and characterization 33
3.4 Chapter conclusions 34

4 Hybrid Cache Subsystem Technology **35**
4.1 Technology of choice 36
4.2 Microarchitectural parameters 36
4.3 SRAM and STT-MRAM models 37
4.3.1 STT-MRAM in-house model 39
4.4 Simulated platform 42
4.4.1 CPU models 42
4.4.2 Cache configuration 44
4.5 Application set 45
4.6 Performance impact 47
4.6.1 Origin of cache conflicts 49
4.6.2 Effect of cache banks on performance 50
4.6.3 Effect of write buffer and MSHRs 51
4.6.4 Effect of hardware prefetcher 51
4.7 Energy consumption 52

4.7.1 In-house model 54
4.7.2 NVSim 55
4.8 Pareto analysis 58
4.9 Chapter conclusions 61

5 Application Memory Behaviour 64
5.1 Goals and constraints 64
5.2 Dynamic binary instrumentation 65
5.3 Memory regions of an application 66
5.4 Profiling methodology 67
5.4.1 Memory objects extraction 68
5.4.2 Detailed access trace 68
5.4.3 Pattern analysis 68
5.4.4 Relevant metrics 69
5.5 SPM placement algorithm 70

6 Heterogeneous On-Chip Memory Organization 73
6.1 Caches and scratchpads 73
6.2 COMPAD architecture 74
6.2.1 SPM 75
6.2.2 SGA-DMA 75
6.2.3 CAT 76
6.2.4 SG-MEMCTRL 77
6.3 Simulated platform 77
6.4 Application set 80
6.5 Simulation results 80
6.6 Chapter conclusions 82

7 Conclusion 86
7.1 Summary and final remarks 86
7.2 Future work 89

Bibliography 93

List of Figures

1.1 Microprocessor trend data [72] . . . 3
1.2 Abstraction layers of a computer system . . . 4
1.3 Von Neumann architecture . . . 8

2.1 Memory hierarchy - access time vs capacity . . . 16
2.2 Example of hybrid cache-SPM system (single scratchpad) . . . 22
2.3 Standard 6-transistors SRAM cell . . . 23
2.4 TSMC SRAM cell area per technology node [77] . . . 24
2.5 Magnetic Tunnel Junction (MTJ) . . . 26
2.6 Energy comparison between SRAM and STT-MRAM for varying sizes [21, 75] . . . 27

4.1 STT-MRAM 28nm cell design [50] . . . 38
4.2 STT-MRAM MTJ R-V curve [50] . . . 39
4.3 STT-MRAM memory array [67] . . . 40
4.4 STT-MRAM LLC architectural breakdown [67] . . . 41
4.5 Instructions classification [64] . . . 48
4.6 Cache MPKI [64] . . . 48
4.7 Relative slowdown per benchmark (8 banks) [64] . . . 48
4.8 Percentage and categorization of bank conflicts (STT-MRAM worst case, 8 banks) [64] . . . 49
4.9 Relative slowdown varying the number of banks (In-order, STT-MRAM worst case) [64] . . . 50

4.10 Effect of L1D write buffer and MSHRs increase (Out-of-order, STT-MRAM worst case, 8 banks) [64] 51
4.11 Stride prefetcher effect on performance (In-order, 8 banks) [64] 52
4.12 In-house accurate energy model - Average LLC energy consumption per $100 \cdot 10^6$ instructions (8 banks) [64] 53
4.13 LLC banks utilization of *607.cactuBSSN_s* (8 banks) [64] . . . 54
4.14 NVSim models - Average LLC energy consumption per $100 \cdot 10^6$ instructions (8 banks) [64] 58
4.15 Energy vs. performance with Pareto front for different number of LLC banks (In-order) [64] 62
4.16 Performance using STT-MRAM w/ 16 banks instead of SRAM w/ 4 banks (Out-of-order) [64] 63

5.1 Memory layout of a C/C++ application 67
5.2 Example of identifiable access patterns [63] 69

6.1 COMPAD architecture [63] 75
6.2 Dynamic data layout compaction [63] 76
6.3 Complete experimental analysis and simulation flow [63] 79
6.4 Analysis metrics [63] 84
6.5 Simulation results [63] 85

List of Tables

4.1 Simulated CPU models [64] 45
4.2 LLC latencies, in cycles [64] 45
4.3 Selected benchmarks (*train* set) [64] 47
4.4 STT-MRAM energy saving (8 banks - LLC only) [64] 55
4.5 NVSim energy models comparison (1MB cache, single bank, $0.5 <$ mantissa ≤ 5) [64] 59
4.6 NVSim STT-MRAM vs. SRAM consumption parameters ratios (1MB cache, single bank) [64] 60

6.1 CACTI caches and SPMs parameters [63] 79
6.2 Simulated system details (single core) [63] 79
6.3 Selected benchmarks (PolyBench/C Suite) [63] 80

Chapter 1

Introduction

1.1 Context

1.1.1 Emerging applications and trends in computing

The modern era of computing, and the impact it has on the society and people everyday life, is exciting like never before. Years of hardware and software improvements have led to a wide series of new applications we could not imagine of in the past. This is also driven by the mass diffusion of Artificial Intelligence (AI) and Machine Learning (ML), which try to replicate some of the mechanisms of the human brain to employ them in traditionally complex processing tasks, or to have a more functional human-machine interaction. These algorithms find application in a wide series of fields, not only intrinsically related to technology, e.g. healthcare, industry, finance, etc.

It is undeniable that technology can help people living a better life, and several emerging applications are a clear example of this. More precise surgery is possible with advanced robotics and telemedicine, the risk of car accidents can be reduced with automated driving systems, we can experience a better teaching and entertainment experience with augmented and virtual reality. All these

applications - and more - are driven by devices that have been growing faster, smaller and less energy hungry with the time.

In general, the growth in computational capabilities have benefited all kind of devices. On the one hand, we have increasingly powerful High Performance Computing (HPC) clusters to execute parallel or distributed workloads and solve complex problems, like physical simulations. The most powerful HPC clusters are stored in dedicated buildings, have an extreme high amount of cores and employ tailored solutions for data communication, power distribution and heat dissipation. At the current moment, we are starting to see in operation the first exascale HPC systems, able to execute a number of Floating Point Instructions per Second (FLOPS) in the order of magnitude of 10^{15} [90].

On the other hand, now more than ever technology is at the hand of common people. We are surrounded by "smart" devices, which are able to extract information from the environment or human interaction and send them over a network, all of this while being powered by batteries. This is the concept at the base of Internet of Things (IoT). A tightly related paradigm is the one of Edge Computing, which we can describe as the ability of low-power sensor nodes to perform part of the data processing before sending the results to one or more powerful devices. This effective decentralization and distribution of the computation tasks, which gives benefits in terms of latency and cost, is also one of the trend topics of the current technological era.

1.1.2 The limits of technological scaling

All the emerging trends described above could not have been possible without a steady advancement of the semiconductor industry. The path has not been without obstacles, though. Since the late '60s, when the first integrated MOSFETs and fabrication processes appeared, people have started to generate prediction trends on the scaling metrics and processor characteristics growth. Such (improperly called) "laws" set a pace for the industry to follow and are known to the specialized public even today. We recall especially a few ones:

- **Moore's law (1965, revised in 1975)** [66]

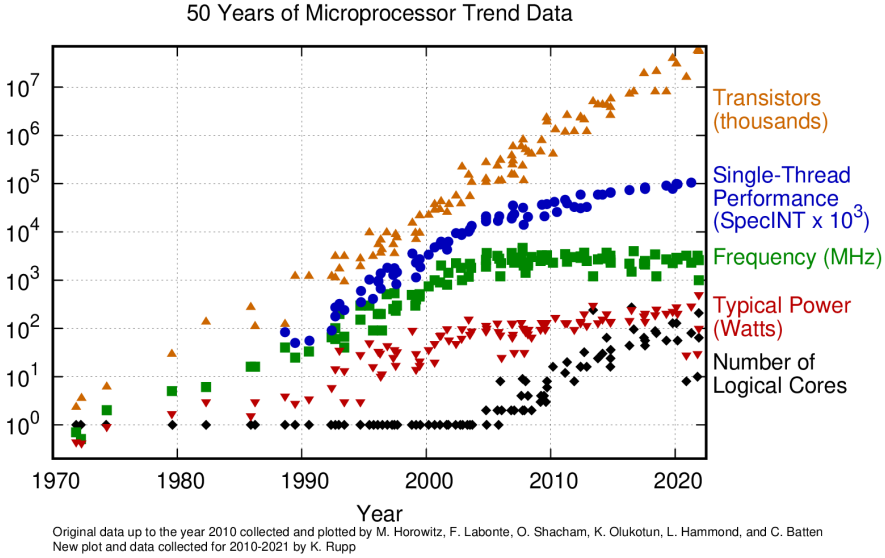


Figure 1.1: Microprocessor trend data [72]

According to Gordon Moore's (co-founder of Intel) previsions, the complexity of integrated circuits doubles approximately every two years. This complexity is measured as the number of transistors.

- **Dennard scaling (1974)** [23]

  This law, named after the engineer Robert Dennard, says that when transistors scale, the power density stays roughly the same. This is due to the combined effect of capacitance and voltage reduction, even taking into account the smaller transition time and thus higher frequency of the transistors.

Moore's law somehow still holds true up to this day. This is clearly visible in Figure 1.1, which represents an effort to keep track the processors metrics over the years, gathering data from popular high-performance architectures. The number of transistors shows an almost perfect exponential relationship with time. However, it is getting harder and harder to keep reducing the transistors

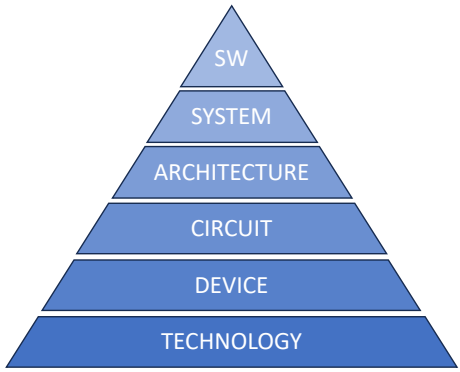


Figure 1.2: Abstraction layers of a computer system

size. The reasons are several, related to the complexity of the fabrication process and the quantum mechanics effects that happen with extremely small physical dimensions [55]. Consequently, it is becoming difficult to maintain the same number of transistors without increasing the chip area. This does not mean that the industry is running out of ideas to maintain Moore's law ongoing. New geometries like the nanosheet/forksheet transistors are being studied to go beyond the 7 nm production nodes [61], as well as vertically stacked structures like the CFET [70]. Nonetheless, it is clear that the point where technological scaling will not be physically or simply economically feasible is approaching.

If Moore's law has seen a long way since its introduction, the same can't be said of Dennard scaling. The turning point was around 2004, when it was noticed that it was not possible to scale the supply voltage without a significant increase of the leakage energy [86]. This has posed a limit to the achievable transistor operating frequency, as can be seen in the same Figure 1.1.

1.1.3 Microarchitectural improvements

Without the possibility to further increase the clock frequency, researchers have then found other ways to improve the performance of devices. One of the biggest achievements in that sense was the gradual introduction of multi-core architectures, where processing units are replicated and can execute instructions

independently. This helps reducing the execution time by running multiple workloads in parallel, or even parts of the same workload where there is little dependency among the instructions being executed. This concepts has been successfully applied to commercial general purpose CPUs: most of today's consumer processors have a physical core count between 4 and 8 [85]. At the same time, in HPC mainframes nowadays it is possible to find processors with more than one hundred cores per socket [49].

Another path that has been taken to optimize processors performance without necessarily increasing the frequencies is the specialization of the execution units: instead of having only a general-purpose Arithmetic Logic Unit (ALU), several other units were introduced to accelerate specific tasks. One of the first to come was the Floating Point Unit (FPU), a math co-processor used for calculations involving decimal numbers. It is employed in all the modern processors, except for resource-limited embedded systems. Other units include cryptography accelerators, vector co-processors, programmable hardware neural networks, etc.

We can make a few more examples of features that have made their way through modern CPUs to improve their performance, like out-of-order execution, branch prediction, hardware prefetching. However, the message we want to pass is simple: transistor scaling and improvement of CMOS technology nodes are not the only way of making our devices better, but there is a whole variety of possibilities at higher abstraction layers as well (Figure 1.2).

If we consider the entire abstraction stack, the best results can be achieved not developing each layer in complete isolation, but improving one layer based on changes from above (top-down) or below (bottom-up). This has led to the development of cross-layer optimization methodologies, like Design-Technology Co-Optimization (DTCO) [60, 29] and System-Technology Co-Optimization (STCO) [29, 73]. Part of the work of this thesis, which will be described in the next pages, can fit in this second category.

1.1.4 The power and energy wall

We have already briefly spoken about the difficulties of increasing operating frequencies without raising energy consumption - and thus dissipated power - above a reasonable threshold, while speaking about the Dennard scaling law. This problem is further exacerbated if we consider that the number of transistors per area is still increasing (as per Figure 1.1), as a consequence of our circuits getting more and more complex.

Even assuming a fixed performance level, the inability to scale down the supply voltage, and the cumulative sum of the leakage currents, are increasing the power density of our chips. If the power per area becomes too high, several issues can arise:

- The circuit dissipates more heat, so it needs more effective and bigger cooling solutions, as well as better power distribution circuitry.
- Higher operating currents may cause a decreased lifetime of the components, due to thermal stress and other physical effects like electromigration.
- Finally, as obvious as it may appear, a higher power consumption brings to a higher cost of keeping the computer running, which is problematic when we think about HPC clusters. Vice versa, in case of simpler embedded systems, a higher power consumption means a shorter device power-on time while operating on battery, with the necessity of more frequent recharges which also deteriorate battery over time.

For the aforementioned reasons, computer architects usually set a fixed power budget for the chips, then apply several techniques to avoid surpassing the imposed boundary.

One of the most popular energy saving methods is to selectively shutdown unused parts of the chip, in order to limit the overall consumption. At a circuit level, this can be done through clock gating, i.e. disconnecting the clock signal, or power gating, operating on the supply voltage itself. A popular term that has

emerged to define inactive parts of a chip is “dark silicon” [27]. It is estimated that with modern technological nodes the percentage of useful silicon can be as low as 10% at a given time [45].

Another way to keep the consumed energy under control is through task-aware power management, modifying the device power mode according to the workload being executed. Most modern processors implement a Dynamic Voltage and Frequency Scaling (DVFS) mechanism to use the minimum amount of power for a given application [99]. This works well for background processes or tasks where direct interaction with the user is not required.

A similar principle has been applied to heterogeneous systems with different core computation capabilities. In essence, some CPUs are composed of a cluster of low-energy cores - typically smaller, slower and with less local memory - and another cluster of high performance ones. This was a necessary introduction for the mobile world where efficiency is key, but is rapidly conquering the desktop world as well. Notable examples of this implementation are ARM processors with a big.LITTLE configuration [32]. Intel has replicated the idea starting from its 12th generation of desktop CPUs, with codename *Alder Lake*, featuring Performance (P) cores and Efficient (E) cores [40]. The role of the operating system to schedule and assign tasks to the right processor cluster is of extreme importance, to limit the power consumption without having a strong impact on the perceived performance.

Researchers are constantly trying to find solutions which can improve the energy behavior of modern chips and overcome the power/energy wall to a degree. As we have seen, there is not a single optimization path, but improvements can be done on several levels. The work that is described in this thesis has as a main goal the improvement of power efficiency of embedded application processors, with a focus on integrated CPU memory hierarchies.

1.1.5 Elements of typical computer systems

Most general-purpose computers are still based on ideas and concepts originated at the beginning of the computing era, in the middle of last century. The

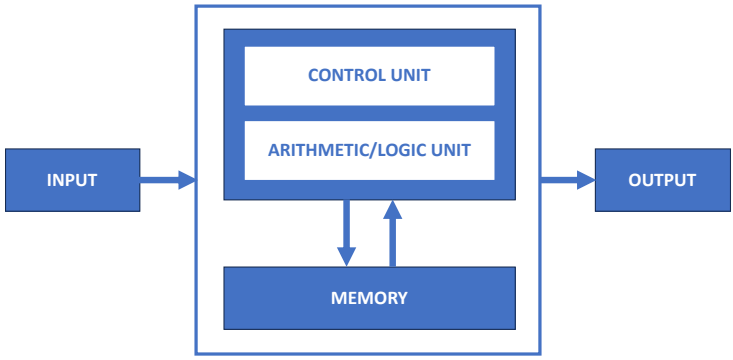


Figure 1.3: Von Neumann architecture

prototypical architecture of a digital computer was considered to be introduced by the mathematician John von Neumann in his *First Draft of a Report on the EDVAC* [94], and is therefore known today as *von Neumann architecture*. The key elements are represented in Figure 1.3:

- A central processing unit (CPU), which in turn is composed of two parts:
  - An arithmetic/logic unit (ALU), which performs the computation itself.
  - A control unit, which orchestrates the various units according to the instructions to be executed.

  The CPU also contains a series of registers, memory elements used to store the operands and results
- A main memory unit, to store both instructions and data.
- Input and output devices, to connect external peripherals and/or storage.

The application code and data are stored in the same memory and accessed similarly by the CPU, with a single access path. A popular variant of this model is the Harvard architecture, where the paths to access the code and data are separated and independent. Most modern processors can be classified as *modified* Harvard architectures, where the separation between code and data

can be relaxed in several ways. Clearly, a modern CPU architecture is way more complex than the models just described, nonetheless the working principles are still the same.

1.1.6 CPU caches

While being extremely flexible, one of the main limitations that soon emerged in von Neumann architectures - taking the name of *von Neumann bottleneck* - is the dependency of data communication with the main memory to perform computing. For several reasons, processors have grown faster with time compared to the main memory units (the so-called *memory wall*). Thus, in several occasions the processor has to stay idle while waiting for instructions or data being transferred from the memory.

In order to reduce this memory bottleneck, caches have been introduced in most processor designs. A cache is a memory device that has the purpose of store local copies of the content of the main memory. Such copies are closer to the execution units, making them faster to access and with a minor consumption of dynamic energy. Data to be stored in caches are selected according to principles of temporal and spatial locality, to maximize their reuse during the lifetime of the copy.

Caches are extremely important in modern systems, as they provide an effective way of mitigating high data access latencies. Due to their key role, they have attracted the attention of many researchers and are a good target of optimization to further improve execution metrics. This thesis revolves around CPU caches, exploring hybrid solutions to improve the energy consumption of the memory subsystem. The fundamentals of caches will be explored in Chapter 2.

1.1.7 Architectural modeling and simulation

Computer architects face a series of challenges when designing new components or modifying existing ones. The main challenge is how to estimate the impact of the proposed solution. Designing an entire component from scratch, traversing

the full stack from the original idea to the physical implementation, is simply not feasible in terms of time and costs.

A common approach for the evaluation of new ideas is to create a software model and simulate it. In this context, where complex interactions happen between the devices of a system, a simple analytical model is not always enough to perform accurate evaluations. Architectural-level and system-level simulators are widely employed by academia and industry for testing new designs. These simulators allow for a certain level of flexibility, providing the user with a way to program and extend them, and allowing to evaluate the impact of the modifications with benchmarks of choice (in form of binary or trace). Examples of popular system simulators include gem5 [10, 62], Sniper [14], ZSim [76]. Specialized simulator for specific parts of a system exist as well, like main memories (Ramulator [48], DRAMSim [71, 59]) or storage (MQSim [87]).

Different simulation techniques can be employed by a software, with important trade-offs between accuracy and speed [4]. For example, it is certainly possible to perform a cycle-accurate simulation of a device starting from its Register-Transfer Level (RTL) implementation, but this will be the slowest of the options, other than being less suited for preliminary evaluations. Event-based simulators like gem5 allow to reduce the amount of simulation overhead, by skipping cycles where no useful events happen. Sniper employs yet another technique, called interval simulation [31], which divides the execution in intervals separated by miss events and uses a mechanistic analytical model to compute timing information.

In general, simulators are powerful tools for architectural exploration, provided reasonable choices are made for the environment selection and that the models are described with a sufficient level of detail. Most of the experimental work of this thesis has been performed using the gem5 simulator. More details on the conducted memory-centric explorations and on the simulation contexts will be presented in Chapters 4 and 6.

1.2 Motivation

In this section we summarize some of the concepts expressed in the previous pages, narrowing down to the effective goals of the presented work.

The general idea of this thesis is to explore solutions to improve the energy consumption of high performance embedded CPUs, which are employed for example in the mobile and edge computing world. These processors are powerful enough to execute any kind of applications, nonetheless they must be extremely energy efficient since these devices do not usually possess a constant source of power.

The target of the proposed optimizations is the CPU memory subsystem. As it will be illustrated more in detail in Chapter 2, caches are an excellent way to reduce the data bottleneck by shortening the distance between the main memory and the processing elements, but they present some notable shortcomings:

- Static Random-Access Memory (SRAM), which is the technology that caches have been traditionally based on, is becoming increasingly difficult to scale. Additionally, cache static leakage is getting more pronounced, which poses a limit to the power efficiency of modern CPUs.
- Caches are transparently managed by an hardware controller, but this automation comes at the cost of energy efficiency in certain situations, due to the complexity of the logic circuitry and the rigidity of the control mechanism.

In this thesis, the problem is approached from two different perspectives:

- Analyzing the impact of emerging non-volatile memory technologies to replace SRAM, with a special focus on Spin-Transfer Torque MRAM (STT-MRAM). These memories, have some peculiarities that must be taken into account when designing a system, and may result in performance penalization. Some insights on possible bottleneck mitigation strategies will also be provided.

- Mixing caches with software-controlled Scratchpad Memories (SPMs). This includes the definition of strategies for data partitioning between the different memory types, starting from specific applications or domains. Considering that it is not always possible to recompile existing applications, the developed methodology only operates on the final application binaries.

In essence, in this thesis it is evaluated how several modifications to the CPU on-chip memory hierarchy can have a positive impact on the device performance and especially energy. The majority of the work relies on software modelling and high-level simulation tools to provide an estimation of the impact of the modifications.

1.3 Thesis contributions

The research contributions of this thesis can be summarized as follows:

- Implementation of a framework for architectural memory exploration, based on commonly used tools in research and industry. Adaptation of cache models to properly simulate non-volatile memory technologies and get meaningful insights.
- Microarchitectural exploration of STT-MRAM memory parameters to replace SRAM last-level caches in ARM application processors. Comparative analysis of performance and energy with respect to traditional SRAM solutions. Pareto-analysis of different configurations, highlighting solutions that are best suitable as a replacement to SRAM for most applications.
- Analysis of memory utilization of applications, extracting information at runtime from executed binary code. Pattern analysis and classification according to application access flow. Definition of strategies for data partition in hybrid systems based on pattern information.
- Proposal of a heterogeneous architecture comprising scratchpad memories together with caches, including support logic for semi-transparent data

management. Modelling of the architecture in the simulation environment. Experiment with loop-dominated benchmarks to show the potential of the proposed architecture.

1.4 Organization

The rest of the manuscript is organized in the following chapters:

- Chapter 2 (*Background*) is meant to provide more details on the technical aspects that are useful to better understand the contributions of this thesis.
- Chapter 3 (*Related Work*) presents the result of a literature exploration centered on the topics of this thesis, highlighting the most interesting related resources and the state of the art.
- Chapter 4 (*Hybrid Cache Subsystem Technology*) illustrates the experimental work about Last-Level Cache (LLC) technology replacement, with an in-depth analysis of the relative slowdown, impact on energy and potential parameter choices for a system of this kind.
- Chapter 5 (*Application Memory Behaviour*) explains the methodology adopted to profile applications and detect patterns in the memory access. Pattern information is then used in conjunction with special placement policies to partition data in the architecture described in the following chapter.
- Chapter 6 (*Heterogeneous On-Chip Memory Organization*) describes the architectural details of a potential heterogeneous cache-SPM architecture with data compaction support. It also illustrates the simulation flow and experimental results obtained with a series of standard loop-dominated benchmarks, modified to apply the previously defined mapping policies.
- Chapter 7 (*Conclusion*) sums up all the obtained results and insights, drawing some conclusive considerations and imagining potential future work.

Chapter 2

Background

2.1 Cache architecture and organization

The key topic of the thesis is to explore energy-efficient integrated memory solutions for modern processors. This has been done by improving on existing solutions, trying to propose optimizations that are not extremely costly or difficult to implement, from a technology, hardware and software perspective.

If we exclude the CPU registers, which consist of a small series of memory elements containing values immediately needed by the execution units, the most important memory unit in the CPU is the cache subsystem. Understanding how a cache is made and operates is extremely important to perform meaningful analyses and improve its characteristics. This section proposes a background on typical cache designs, describing the different logical elements that compose a cache and evaluating how they contribute to the energy and performance of a memory access.

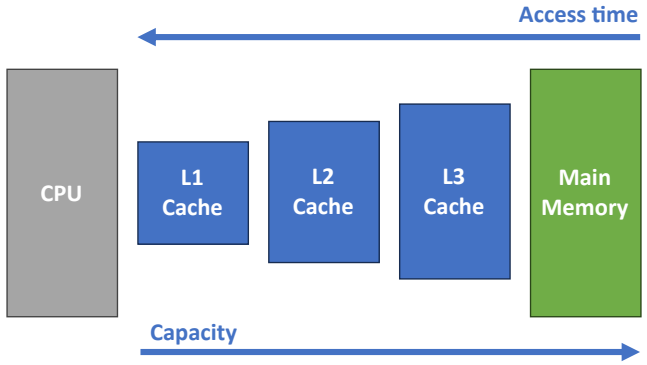


Figure 2.1: Memory hierarchy - access time vs capacity

2.1.1 General concepts

A cache is a device which is meant to reduce the distance between the data and the processing elements. It does so by storing copies of the content of the main memory, based on data locality principles. The cache is completely transparent to the application: when the CPU sends a request to the memory subsystem, the cache controller checks if a copy of the data is cached. In case it is present, a *hit* occurs, and the request is served by the cache itself, with a reduced cost in terms of performance and dynamic energy. Otherwise, the request is sent to the following memory level, which is the consequence of a *miss*.

In systems where performance plays an important role, large quantities of cache memory are present. However, since accessing large caches entails a higher latency/energy cost compared to smaller ones, it is common to employ a hierarchical structure, where there are multiple levels of different sizes throughout the path towards the main memory. The first level (L1) cache represents the memory closest to the processing elements, while the LLC is the closest to the main memory. Modern systems can reach up to 4 levels of cache [108], while this amount is reduced in less performance-critical devices, like embedded systems. A representation of a sample cache hierarchy is depicted in Figure 2.1. To give some reference values, accessing a L1 cache typically costs a few clock cycles (or a few nanoseconds, assuming 1 GHz of CPU frequency). Accessing the main memory, instead, can cost more than one hundred cycles.

A design choice which is extremely common is to split the first level cache in a *data cache* and an *instruction cache*. This allows to relieve the von Neumann bottleneck by providing separate caching for instructions and data, which reduces the amount of resource contention and partially separates the two access paths, at least until the following shared memory level. In case of multi-core processors, downstream caches can be either private to each core or shared among them.

Traditionally, CPU caches are SRAM-based and integrated in the processor package, but this is not always the case, especially for big levels [3]. Also, caches are not only employed in CPUs, but also in GPUs and sometimes mass storage, proving their importance in shortening the data distance for different elements of a system.

2.1.2 Temporal and spatial locality

Caches are limited in size and can only store a subset of the content of the main memory. As a general concept in computer science, when some set of data is accessed by an application, it is likely to be accessed again in a relatively small time frame. This property takes the name of *temporal locality*. A cache exploits this property by creating a copy of the data after a miss, and storing it until it needs to be deleted to make space for other data. The logic that is employed to decide which of the older data is removed is referred as *replacement policy*. A typical choice, in line with the temporal locality principle, is to get rid of the least recently accessed data (LRU or Pseudo-LRU policies). However, different replacement policies can have different effects according to the executed applications [109, 56].

Another common property exploited by the cache is the *spatial locality*. According to this principle, when a typical application accesses data from a certain memory position, it is likely to also access close locations in the near future. To exploit this property, the cache transfers and stores data with a granularity which is higher than a single word. This way, subsequent data locations are also copied in the cache for a more immediate access. The unit of the data transfer in caches is typically fixed and takes the name of *cache*

line or *block*. Most modern systems adopt a value of 64 bytes for the cache line size, even if this value has been changing over time and with different types of architectures. For example, Intel Itanium and IBM Power processors have a 128 bytes cache line, while the recent Apple M1 uses 64 bytes for the L1 cache and 128 bytes for the L2 cache [103].

2.1.3 Tags and data

Since caches are managed in hardware, the controller needs to verify for each access if a copy of the data exists or not in the current level. To achieve this goal, when a new line is inserted in the cache, a part of the address is stored together with the data copy, to be used as a fingerprint. This part takes the name of *tag*. To check if a successive request points to the same stored set of data, it is sufficient to compare the tag with a series of potential candidates inside the cache. If one of the tags matches, a hit is produced.

Tags can be stored together with the data or on a separate structure. In this case, we talk about *tag array* and *data array*. Having separate tags allows for higher speed, as the tag comparison can happen while fetching the line. It is worth mentioning that the size of the tag array is a fraction of the data, as only a few tens of bits are stored for each cache line (i.e. 64 bytes, typically). The tags access cost is therefore limited as well.

2.1.4 Placement policies

As a potential design choice, there are several ways a block of memory can be mapped to a cache, with different consequences.

- *Direct mapping*: Each address points to a single specific line in the cache. The position is calculated from a subset of the address bits, called *index*. This approach has the advantage of needing a single comparison (only one position is possible), but lacks flexibility and may generate a lot of conflict misses.

- *Fully associative*: This is the opposite approach. A line can be stored everywhere in the cache, and a number of comparators equal to the number of cache lines is needed to find the (potential) position of the data. In this case, there is no need for any index, and most of the address becomes the tag itself. Fully associative memories are expensive, so they are only employed for specific high-speed applications, e.g. as Content-Addressable Memories (CAMs).

- *Set-associative*: This is a trade-off between the solutions described above. The index identifies a set of cache lines, limited in size (typically between 2 and 16). This limits the amount of comparisons to perform, providing at the same time a certain degree of flexibility, as there are multiple potential position for the data. Most modern caches are set-associative. The number of lines composing each set can also be described as *number of ways* or *associativity*.

The reported policies are the most common ones, but variations exist which involve more complex mapping, like skewed-associative [78, 79, 11], column-associative [1], hash-rehash [2].

In systems where a MMU is present, the indexing and tagging can either happen using the virtual address or the physical address. This again has some impact on the access speed (the Translation Lookaside Buffer (TLB) has to be interrogated to get the physical address). In the context of this thesis, it will be assumed that all caches are physically indexed and physically tagged (PIPT), i.e. cache access happens after virtual-to-physical translation. This choice is made in part for simplicity, in part to cope with limitations of the simulation environment.

2.1.5 Cache coherency

In cases where different caches are present which may contain copies of the same data, there is the necessity to maintain data integrity and make sure that the most recent copy is always accessed. This is what happens for example in multicore systems, where multiple processors own private cache levels.

In most cache-based systems, it is possible to find a hardware implementation of a cache coherency protocol, to maintain data consistency across the memory subsystem. The most popular options involve the use of *snooping messages* or a *directory-based* approach. In the first one, when a piece of data is updated, if it is owned by multiple devices they are be notified as well. There are two main possibilities to deal with this situation:

- *Write-invalidate*: The other copies are deleted and a miss will be produced on successive accesses.
- *Write-update*: The other copies are updated with the new value.

In directory-based systems, instead, coherency is maintained by querying a structure that stores information about which caches possess copies of the data. This mechanism is generally slower for traditional systems, but better suited for distributed ones as it scales better.

Since the focus of this thesis is on memory optimizations for single-core systems, the coherency problem has not been treated. Nonetheless, this is an important element to take into account when designing systems with private cache levels belonging to multiple independent cores (or core groups).

2.1.6 Non-blocking caches

With the logic becoming increasingly faster, memory accesses constitute an important source of delay in modern systems. In classical caches, when a miss occurred, the CPU had to stall waiting for the access to be completed. To solve this issue, most modern caches possess additional structures to allow for parallel execution of requests [18, 30]:

- *Miss Status Handling Registers (MSHRs)* [53]: These registers are used to store outstanding (*in-flight*) misses. While waiting for data fetching from one miss, the cache is able to process other requests and potentially generate other misses, allowing the CPU not to stall if it does not immediately need the data. The MSHRs contain information on the

requested cache blocks, for which multiple subentries can exist that keep track of the specific target word and the source of the request. This way, successive misses requesting the same line (*secondary misses*) can be merged in a single request.

- *Write buffer*: This buffer serves a similar purpose, but for writes generated by the cache. Specifically, in write-back caches it stores pending line writes that are directed to downstream levels (due to eviction), while they are waiting to be executed. Also in this case, multiple writes can be coalesced into a single entry.

The size of these structures can have an impact on the execution, as when they are full they prevent the cache from accepting new requests. For this reason, this has been one key element of evaluation of the work presented in Chapter 4.

2.2 Scratchpad memories

Cache memories, while extremely useful to reduce the data bottleneck, present some inefficiencies due to the way they are built. First, there is the need for complex control logic to make the mechanism transparent. Every cache possesses a *tag array*, which is used to identify if a copy of the data is already present or not in that specific cache level. Other than producing an extra area occupation, there is an inherent energy cost associated to the tag array access, especially in the first level cache where multiple entries are read at once to make the access as fast as possible. The management of the copies in multiple cache levels and the maintenance of data coherency also make the implementation more complex and potentially less energy efficient. Finally, the fixed granularity of the data transfer - in chunks of several bytes - may lead to waste of bandwidth when fetched lines are underutilized.

While not being a common option in general-purpose processors, there is an alternative to caches, which is SPMs. These memories share the same purpose of decreasing the distance between data and execution units, and are on-chip solutions as well. The main difference with caches is that scratchpad

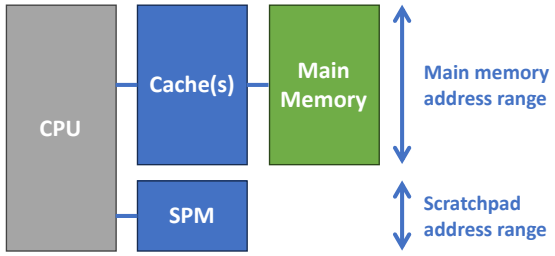


Figure 2.2: Example of hybrid cache-SPM system (single scratchpad)

memories are software-controlled: the programmer can directly read and write to the memory using load/store instructions. No cache controller or complex comparison logic is needed anymore, since there is no automatic data management. Of course, the drawback is that the complexity is shifted from the hardware to the software programmer, and manual data management can be complex to perform for big and dynamic applications. For this reason, it can be convenient to have a mixes solution with both memory types, using a scratchpad only when there is a potential benefit (as in Figure 2.2).

As it will be shown in the following chapters, scratchpads are smaller, faster and more energy efficient compared to caches of the same size, even when they are made with the same technology. Especially in relation to the first level cache, there is no extra energy consumption associated to the multiple parallel tag comparison and line pre-selection. Furthermore, assuming that it is possible to use a smaller granularity for data transfer and change the data layout, it is possible to reduce the bandwidth and energy consumption of applications that do not fully exploit the size of a cache line. These kind of optimizations require a detailed study of the application memory behavior, and modifications to the hardware and software to be enforced.

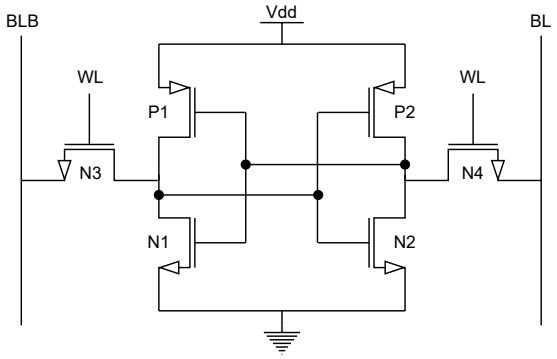


Figure 2.3: Standard 6-transistors SRAM cell

2.3 Memory technologies

2.3.1 The SRAM technology

To meet the speed and power requirements of the caches, a suitable memory technology is needed. Traditionally, the technology used for caches has always been the SRAM, at least for levels closer to the processor. This technology present several advantages over the one used in main memory designs, i.e. Dynamic Random-Access Memory (DRAM): a much smaller latency, high bit stability (i.e. no need for data refresh), easiness to integrate with CMOS logic. However, there are drawbacks as well: the cost per bit is higher, as well as the area occupation, which poses a limitation to the total size that can fit into a processor design. A typical SRAM cell, in fact, is made of at least 6 transistors [33], compared to the single transistor plus capacitor of a DRAM cell. A representation of the standard 6T SRAM cell is depicted in Figure 2.3.

The limited density of SRAM already constitutes a problem in modern high-performance processors, which are demanding for ever larger caches. To make a few examples, first Intel Atom processors had an L2 area occupation of 65%, and 90% of the die of Intel Itanium 2 processors was occupied by SRAM [97]. What makes the situation worse is that SRAM cells are beginning to scale way less than logic. A recent presentation from TSMC shows that, despite the reduction in transistor dimensions for their latest 3nm nodes, the area of the SRAM cells

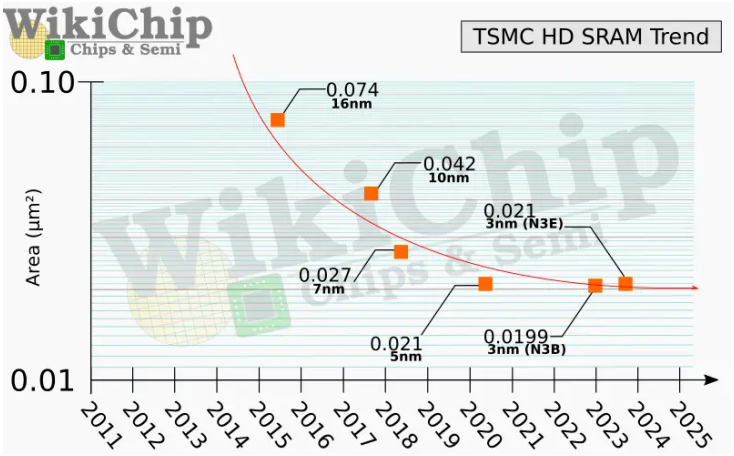


Figure 2.4: TSMC SRAM cell area per technology node [77]

is stagnating (Figure 2.4). This trend applies in a similar way to other foundries like Intel [77]. The reason for this is that reducing the transistor sizes affects the read and write margins, intensifies the effects of process variations and increases the static energy consumption.

The issue with cache density and area occupation can be somewhat circumvented with bleeding edge 3D-stacking technology. Some of the latest AMD consumer CPUs feature up to 128MB of L3 cache, which is present on a separate chiplet stacked on top of the core die through Through Silicon Vias (TSVs) [100]. Generally speaking, chiplet partitioning and 3D integration of different components, potentially based on different technology nodes, is one of the emerging strategies that is keeping Moore's law alive up to these days. However, adding more components to the vertical dimension does not address power density problems, actually making them worse.

2.3.2 Technology requirements for caches

Since the current status of SRAM is concerning for future designs, researchers have been exploring alternative memory technologies to be employed in caches for a long time. This is not an easy task, since the memories to be employed in

caches must be fast, power efficient and most of all reliable, due to the high frequency of accesses.

Specifically, the access time must be in the order of nanoseconds, from a few units to a few tens depending on the cache size. The energy per access should be more or less in line with SRAM as well, which is around a few tens to a few hundreds of picojoules. Furthermore, it is very important that the memory has a sufficient endurance to sustain frequent reads and writes: the generally accepted reference threshold is around 10^{15} operations, but this also depends on the cache level.

Among the alternative memory technologies that are available today (see Chapter 3.1), the only potential candidates for cache replacement are magnetoresistive memories, which will be introduced in the next section. All the other technologies break these requirements to some extent, and can have a better fit as a replacement/integration of other memory levels or for very specific applications.

2.3.3 Emerging memory technologies - MRAMs

Memories that are currently employed in our computer systems work by storing information (bits) in form of electrical charge. However, there are other ways of physically storing information, exploiting the properties of special materials. One of the most promising class of devices currently under development is Magnetoresistive RAMs (MRAMs). Basically, they work by polarizing a free ferromagnetic layer and measuring the resistance across a structure called Magnetic Tunnel Junction (MTJ) (Figure 2.5). In case the free layer is parallel to the fixed layer, the path has a lower resistance, which corresponds to a logic zero. In case the two layers are antiparallel, we have a logic one instead, since the resistance is higher. These memories are non-volatile, i.e. they retain the data even after cutting the supply voltage.

The way of inducing the polarization is different according to the specific device. STT-MRAM, for example, uses a spin-polarized write current running through the MTJ. In Spin-Orbit Torque MRAM (SOT-MRAM), instead, the

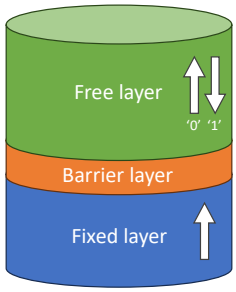


Figure 2.5: Magnetic Tunnel Junction (MTJ)

current is injected parallel to the stack, exploiting the transfer of the orbital angular momentum to polarize the free layer. Finally, other devices like Voltage-Controlled Magnetic Anisotropy MRAM (VCMA-MRAM) controls the orientation of the free layer with by generating an electric field instead of injecting a current. In essence, all these devices share the same fundamental principles, but their characteristics are slightly different due to the way they are programmed.

As already mentioned, MRAMs are currently the only emerging memories that possess all the properties to be used as caches, at least in some of their specific implementations. It is important to note though that a drop-in replacement for SRAM is yet to be found, and the introduction of such emerging technologies has to be properly evaluated in all aspects. This thesis work mostly revolves around STT-MRAM for the technology exploration part. It is the most mature among the spin-based magnetoresistive memories, and years have been spent on research and refining of the fabrication processes. This is one of the few technologies that is truly being considered as a replacement for SRAM caches in specific contexts. There are several reasons for this: its base cell is smaller (usually made of one/two MTJs and one/two control transistors), the leakage current is almost non-existent and read speed and dynamic energy are on par with current cache solutions. However, there are disadvantages as well, which are a limiting factor for the diffusion of these memories: the write latency is higher than SRAM, and the write energy only starts to be on par with SRAM at high memory sizes (Figure 2.6). This means that this memory technology, in its current status, is not suitable as first level cache (also due to endurance

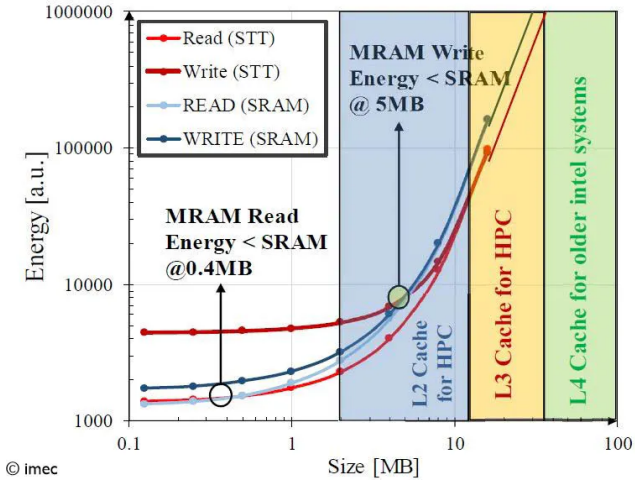


Figure 2.6: Energy comparison between SRAM and STT-MRAM for varying sizes [21, 75]

considerations), but can be potentially used for downstream caches if ways to mitigate the increased latency are found. For reference, recent literature stated that the write latency in STT-MRAM is estimated to be around 2 to 5 times larger than the corresponding SRAM one, in modern 5nm technology nodes [75].

Part of the work that will be illustrated in the following chapters consists in an extensive study of how a LLC memory technology replacement - from SRAM to STT-MRAM - would impact the execution of a variety of representative benchmarks. The conducted experiments prove that it is possible to reduce the overall energy consumption without severely impacting the performance in most situations. The study is performed on low-power ARM application processors, mostly found in the mobile or edge world.

According to several works in literature, SOT-MRAM is emerging as an interesting alternative to STT-MRAM, with reduced impact on write energy and latency due to the way this operation is performed. Oboril et al. [68] claimed that this memory technology closes the gap with SRAM with only a negligible impact on the cell area, and it is possible to use as a replacement for

L1/L2 cache. Saha et al. [74] reached similar conclusions in their study. On the other hand, sources closer to industry mention that the area overhead is still significant due to the extra access transistor and additional wiring, and that the fabrication of SOT-MRAM devices still presents some issues that are only recently being solved [22]. Since the technology exploration conducted in the first study of this thesis focuses on the LLC, it was more reasonable to stick with the denser STT-MRAM technology. The second contribution, instead, is mainly centered on data placement rather than technology. Potential future work involving technology exploration with smaller caches is certainly a good fit for SOT-MRAM. For example, it could be interesting to explore hybrid systems with SOT-MRAM as L1 cache and STT-MRAM as L2/LLC.

Chapter 3

Related Work

The topics discussed in the present manuscript cover different aspects related to computer systems design and modelling, trying to mitigate problems that also affect modern commercial solutions. Hence, one can expect that the amount of previous research is quite abundant. Here follows a description of some of the most relevant and interesting works, divided in categories for convenience, although some overlap between areas is naturally expected.

3.1 STT-MRAM and hybrid caches

Since the discovery of emerging memory technologies and their properties, researchers have been studying the feasibility of employing them as caches, focusing on their differences compared to traditional SRAMs. Several comparative studies that analyzed the characteristics of these memories are present in literature, such as the one from Yu and Chen [104] or a more recent work from Jangra and Duhan [43]. These works evaluated, other than magnetoresistive memories, solutions such as Resistive RAMs (ReRAMs) and Phase Change RAMs (PCRAMs). As mentioned before, these last options are not suitable for use in caches, mainly due to the poor endurance (ReRAMs, PCRAMs) and high write energy (PCRAMs). Hence, and for the other reasons

described in the previous chapter, the focus of this literature study will be on the STT-MRAM technology.

Few works have considered STT-MRAM as a potential replacement for a first level cache, like the exploration conducted by Komalan et al. [51]. According to the authors, this is possible thanks to a cross layer design methodology and modifications to the architecture, like a large victim buffer to hide increased latencies. Nonetheless, it still entails some performance penalty and, above all, endurance concerns. This is why most of other works, including the one here presented, focus on bigger caches with a lower number of accesses, e.g. last-level caches. One of these studies was conducted by Chang et al. [17]. Their comparative analysis included SRAM, STT-MRAM and eDRAM, optimized for cache use with power reduction techniques.

We have seen that one of the main limitations of STT-MRAM is the increased write times compared to traditional caches. Apparently, non-volatility characteristics of STT-MRAM can be lifted in favour of better write performance, when designing the device. The trade-off between write latency and retention time has been explored by several authors, including Chang et al. in the previously cited work. Comprehensive studies of this property were conducted, for example, by Smullen et al. [83], who explored hybrid caches with reduced-retention STT-MRAM L2/L3, to mitigate the performance issues of a fully STT-MRAM cache and still save energy compared to traditional systems. Other works exploited STT-MRAM pools with different retention times, storing data in the best matching pool according to the data read frequency and lifetime. This is the case of works from Khoshavi and Demara [47] and Kuan et al. [54].

Many other proposals in literature preserve the non-volatility properties of STT-MRAM, while mixing them with SRAM in hybrid systems. This can happen even within the same cache level, with banks of different technologies. For example, previous work from Cheng et al. [19] introduced a custom inclusion policy for STT-MRAM LLCs to reduce redundant writes, extending it to hybrid SRAM-STT LLCs to save energy while maintaining a low write overhead. Another work from Wang et al. [96] defined a placement policy for data in SRAM or STT-MRAM LLC banks, based on the access pattern of different write accesses, enforced with the help of a hardware pattern predictor.

Other works simply employ a single STT-MRAM technology for LLC, trying to find workaround to the increased latency. A previous paper from Wang et al. [95] proposed a way to identify "obstructing" accesses and occasionally bypass the LLC, going directly to L2 or main memory. The work presented in this thesis also focuses on cache contention issues, trying to understand their source. Some works from Hameed et al. [36, 35] propose STT-MRAM as a replacement for large DRAM caches, with several modifications to the row buffer and cache architecture to reduce conflicts and improve energy and performance.

Finally, a work from Evenblij et al. [28] identified in the banks contention the main reason of the performance degradation in systems with an STT-MRAM based LLC. Consequently, they analyzed the effect of varying the number of banks on performance and energy, providing a comparison with SRAM.

The first contribution of this thesis consists of a detailed feasibility study of a STT-MRAM based last-level cache in energy-constrained application processors. It can be considered a prosecution of the work from Evenblij et al. [28], with a more in-depth analysis on the causes of bank contention in the LLC. Additional elements not present in previous work are: slowdown evaluation of other cache microarchitectural parameters, energy comparison between a calibrated in-house memory model and a model produced with public modelling tools. More details about this work will be presented in Chapter 4.

3.2 Scratchpad memories and data placement

The second big topic which is covered in this thesis is scratchpad memories, in the context of heterogeneous solutions mixed with caches. SPMs have been explored for several decades, and they are widely employed for example in modern GPUs, where they can be used on demand by graphical applications. Here scratchpads are instead considered a solution for general-purpose CPUs, to allow for domain-specific optimizations with the goal of improving efficiency. Since SPMs are simple software-controlled memories, research is generally focused on how to use them, i.e. which data to store there and at which point in time.

Many works in literature treated the SPM data placement problem in systems without caches, or with a scratchpad in place of a specific cache type (*instruction* or *data*). Starting with instructions, a work from Egger et al. [26] proposed an automatic code mapping to SPM methodology for embedded devices without a Memory Management Unit (MMU). Still focusing on instructions, Janapsatya et al. [42] proposed a method to map frequently utilized code segments to an SPM, with the help of a hardware controller to perform copies at runtime. The cited works are interesting because, unlike many others, they do not involve access to the source code of the target applications, neither for profiling not for enforcing the policies. For this last task, binary modification techniques are used instead. One of the main goals of this thesis is indeed to show that optimizations are possible even without the possibility to access original sources.

Other literature focused on data instead of instructions. A following work from Egger et al. [20] proposed a data mapping methodology for SPM-based systems, based on the solution of an Integer Linear Programming (ILP) formulation using memory access and cache misses information. Udayakumaran et al. [91] presented a compiler-driven SPM dynamic allocation method for global and stack data, based on predicted frequency of utilization. Another class of works targets less predictable dynamically allocated data (which form part of the application *heap*). Dominguez et al. [24] claimed to be the first authors to propose a compile time method for allocation of heap data into an SPM. Their methodology consisted in identifying regions in the application (based on loop or procedure boundaries), then copying part of the heap content into a scratchpad, in form of fixed-size bins, when a new region is accessed. McIlroy et al. [65] also introduced an algorithm for heap data management in SPM, using a variety of techniques to balance effectiveness and memory footprint. The work does not give hints to the programmer about which data to map, nonetheless provides an optimized allocator that makes SPM management easier.

There are also works that cover both code and data, mapping them to a single scratchpad. This is the case, for example, of the work from Verma et al. [93], to map variables and code segments based on liveliness analysis. Other than focusing on the data mapping aspects, it is also interesting to make some more comprehensive architectural explorations. Banakar et al. [8] did a comparative

analysis between caches and scratchpads for different memory sizes, from an energy, performance and area point of view. Also in this case, they used an SPM placement policy targeting portions of both code and data. The study showed that unless the memory size is very small, scratchpad perform better in all the aspects.

Non-volatile memory technologies are also an option for scratchpads. Several feasibility studies exist in literature which evaluate the advantages of mixed SRAM-NVM scratchpad solutions, with special placement policies that account for the peculiarities of these emerging memory types [80, 9, 37].

The works that have been reported until now focus on the use of scratchpad in cacheless systems. In this thesis, one of the main goals is to analyze mixed cache-SPM solutions. Relevant related work has been produced, among others, by Chakraborty et al. [16] and Zhang and Wu [107], covering the data placement and optimal memory partitioning problem in this kind of heterogeneous systems. Another work from Komuravelli et al. [52] proposed a single memory organization which mixes aspects of the caches and aspects of the scratchpads, to get the best of both worlds.

The second contribution of this thesis is a series of architectural modifications to support semi-transparent data management in a hybrid cache-SPM system. This architecture is paired with a software methodology to analyze applications access patterns and take automatic decisions on the data partitioning. The main element of novelty in the architecture is support for data layout compaction, to transfer and store strided patterns without redundant elements. Additionally, the software methodology does not require access to the application source code, which is not common in previous work and has been only seen in simpler contexts.

3.3 Application profiling and characterization

Several works have been of help for the benchmark selection and memory profiling parts of this thesis.

Singh and Awasthi [82] performed a memory-centric characterization of the SPEC CPU2017 [12] benchmark suite, using different techniques to obtain a wide amount of insights on the applications behaviour. Song et al. [84] developed a methodology to detect similarities among benchmarks and select a representative subset, targeting the same SPEC CPU2017 suite.

The methodology proposed by Ji et al. [44] to analyze the access patterns of applications is similar to the one employed in the second part of this thesis, although there are differences in the context and reference metrics.

3.4 Chapter conclusions

In this chapter, an overview of the most relevant and novel literature covering the main topics of this thesis has been provided. While the previous research on emerging technologies and scratchpad memories is plentiful, there are several gaps that are filled by the contributions of this thesis. Comparative analyses about different cache technologies (including non-volatile ones) are present, but a deeper microarchitectural exploration based on different design knobs was missing. Also, it is hard to find a real comparison between calibrated energy models and ones generated with academic estimation tools, which is something that has been treated in this thesis. Regarding the use of scratchpads in the CPU memory subsystem, none of the available work in literature propose a way of automatically compacting non-contiguous data while storing it in an SPM. Furthermore, the works involving binary analysis and modification without access to the source code are limited and not applied to a heterogenous context. The contributions of this thesis add value to the existing literature and provide additional elements for improving on the current status of research.

Chapter 4

Hybrid Cache Subsystem Technology

This chapter contains an overview of the first experimental topic of this thesis, which is about exploring the impact of mixed technology types in the cache subsystem. The goal is to overcome some of the limitations of the SRAM technology (namely, high leakage currents and increasingly low density), aiming at an improved CPU energy consumption, which is critical for devices in the edge/IoT world. A relatively small performance penalization is considered to be acceptable in this context, if well compensated by an energy reduction. As a consequence, another important goal of this work is to see which applications are more impacted by the use of the new technology, and how it is possible to perform choices at the cache design time that can keep the slowdown as low as possible. The applications set has been selected to include a good mix of benchmarks from different domains, with a variable degree of memory intensity. Most of the content here described has been published in an original journal article [64].

4.1 Technology of choice

As already anticipated in the previous pages, in this work the technology of choice for the LLC replacement is STT-MRAM. The selection has been made for a series of reasons. First, its energy and latency characteristics are compatible with the ones of a cache, even if there are differences that will soon be highlighted. Secondly, the endurance is also better compared to other types of memory: the optimal value of 10^{16} cycles [6] is acceptable for a cache not too frequently accessed, like an LLC. Finally, the maturity of the device and consequently the availability of detailed models also had an impact in the decision. Other devices may have better characteristics on paper, but still a long way to go before being reliably fabricated. On the other side, it is already possible to find STT-MRAM memories in the market, albeit mostly as standalone chips.

The use of STT-MRAM in (last-level) caches entails a series of drawbacks that may impact the overall execution metrics of the system. The main critical points are two:

- A higher write energy compared to SRAM. This is usually more than compensated by the near-zero static leakage of the MTJ.
- An increased write latency. This is probably the most problematic point.

The performance impact strongly depends on some of the microarchitectural design choices, other than the specific application being executed. This will appear more clear in the following sections.

4.2 Microarchitectural parameters

Regardless of the technology, there are several microarchitectural elements that a memory architect gets to choose when designing a cache.

From previous work from Evenblij et al. [28], we know that the *number of cache banks* has an important effect on performance when using non-volatile memory technologies. A cache bank can be defined as an independent unit

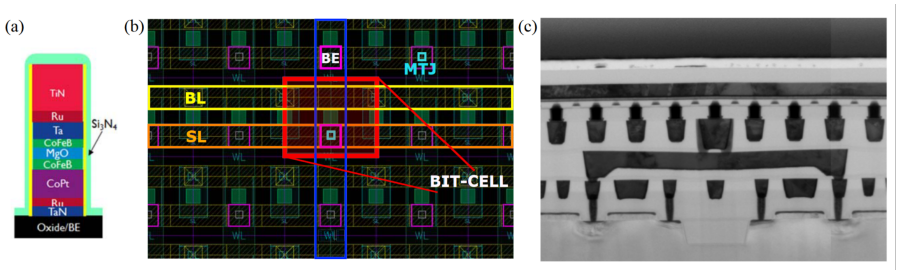
within the cache, and it possesses its own part of the data array and control logic. The purpose of having multiple banks is to allow for parallel access to the data. A high number of banks decreases the possibility of contention, i.e. the possibility that an access has to wait for a previous one to complete. This effect is, of course, more evident the higher the access latency is, which is particularly relevant for write operations in STT-MRAM. On the other end, increasing the number of banks involves a certain amount of duplication of the control logic and longer interconnect, which have a negative impact on consumed energy.

Other parameters that have been evaluated in this exploration - in terms of performance impact - are the *number of write buffer entries* and *number of MSHRs*. The write buffer is a support structure present in all modern caches, used to temporarily store write requests going to the downstream memory, to be processed in a second moment. This is because it is not advisable for the cache to block other operations while waiting to write to a slower memory. The write buffer asynchronously processes the pending write requests in background, and performs coalescing of accesses targeting the same line if needed. The MSHRs serve a similar purpose, but for misses, tracking outstanding memory accesses and preventing redundant requests from happening. Both structures are therefore used to hide latencies during cache operations, and a higher number of entries could be helpful in the context of the technology change.

Finally, the performance effect of enabling hardware prefetching has been also evaluated. As the name suggests, prefetching is a mechanism that allows data to be collected before being actively demanded by the application, when it is possible to establish a pattern in the data request sequence. Different implementations are possible, but for simplicity only the most common type has been considered, which is stride-based prefetching.

4.3 SRAM and STT-MRAM models

When conducting technology explorations - especially with high level simulations, where many details about the inner implementation of a device are hidden - it is of critical importance to employ accurate abstract models of the devices



(a) Perpendicular MTJ film structure

(b) Bit cell layout

(c) Cross-section TEM image

Figure 4.1: STT-MRAM 28nm cell design [50]

under evaluation. These abstract models are extracted from real or realistic implementation models, capturing the metrics that are needed to perform a correct evaluation and linking them to a set of parameters that can be changed prior to starting a new experiment.

In this study, industry calibrated models have been used for both SRAM and STT-MRAM. These models are not publicly available, but have been provided by technology experts in a context of collaboration for this work. The SRAM model has been generated using a commercial memory compiler, for a *Single Port, High Density Performance Leakage Control* device. The reference technology process is a 28nm CMOS *HPM High-K Metal Gate P-Optional Vt/Cell Std Vt*. Since this is a pretty standard model, the focus in the text will be mostly on the STT-MRAM one.

In the last part of this work, the evaluation has been repeated based on another energy model derived from a publicly and freely available tool (NVSim [25]). A comparison of the two models have been performed, showing that the results exhibit a lot of variation and that numbers from non-verified models cannot always be trusted.

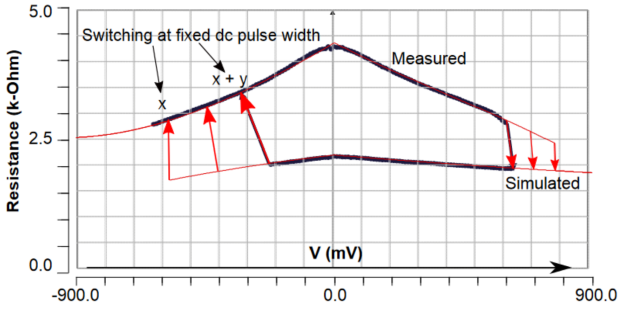


Figure 4.2: STT-MRAM MTJ R-V curve [50]

4.3.1 STT-MRAM in-house model

For the STT-MRAM, also created for a 28nm technology node, the reference design has been based on previous work by Komalan et al. [50], which in turn is based on a perpendicular MTJ implementation described in an article by Van Beek et al. [92]. The key characteristics are listed here:

- MTJ pillar diameter: 40nm
- Tunnel magnetoresistance (TMR): 150%
- Intrinsic resistance-area product (RA): 10.9 $\Omega\mu m^2$

The STT-MRAM cell is composed of a single MTJ and a single planar NMOS access transistor (1T1MTJ). The physical structure of the device and its layout are shown in Figure 4.1. The free layer and refence layer are based on CoFeB and separated by an MgO interface. The dimensions of this bit cell are 260nm x 220nm. The cell has an area occupation which is approximately equal to 1/3 of a corresponding SRAM cell. However, if we consider a full design including the periphery, the area advantage decreases (reaching a 40%-50% occupation). The device analytical model was generated based on real measurements, hence it is true to silicon data. The model takes into account the resistance, AC and DC behaviour, breakdown and temperature dependence of the MTJ. Comparing the simulated R-V curve with the measured data in Figure 4.2, it is easy to note that there is virtually no difference between the two.

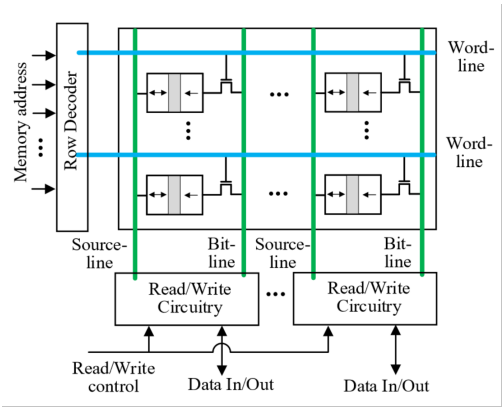


Figure 4.3: STT-MRAM memory array [67]

A representation of a typical STT-MRAM array is depicted in Figure 4.3. The access logic is not very different compared to a traditional memory: the address is decoded to activate a specific word line, then the current flowing through the corresponding MTJs is sensed to obtain the content of the line. Fetched data are usually stored in a row buffer, and single words can be accessed through the use of multiplexers. The decoder and the I/O logic, together with part of the control logic, form part of the array/macro.

In the specific model employed for the experiments, to form a memory macro, cell sub-arrays and logic are organized in a butterfly configuration: four sub-arrays divided horizontally by the I/O logic and vertically by the (pre/post) decoders. The aspect ratio can be either 1:1 or 1:2, depending on the macro size. The supply voltage is 0.9 V. The entire design was created with the help of the Cadence Virtuoso [38] tool. More details on the characteristics of this model are available in the published article [64].

The full STT-MRAM cache architecture model used for the experiments follows an organization described in previous work [67]. This organization is represented in Figure 4.4. The basic element is the *array* (or *macro*), which content has already been discussed above. A series of arrays interleaved with signal repeaters, plus a small part of control and I/O logic, form an *instance*. Multiple instances are grouped in a *bank*, which represents one of the multiple independent units

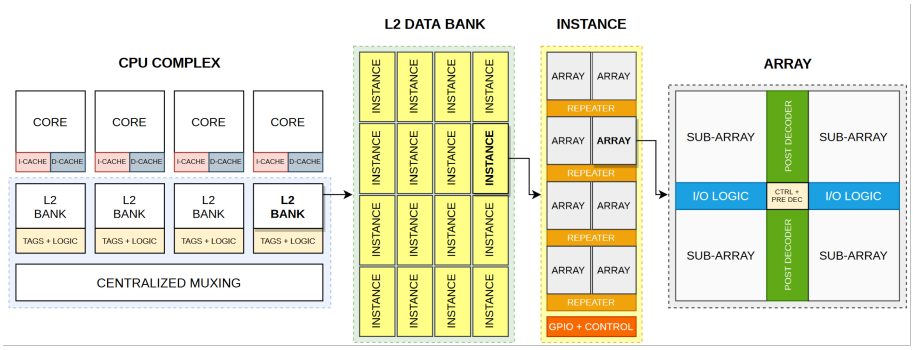


Figure 4.4: STT-MRAM LLC architectural breakdown [67]

that are present in a cache and can be accessed in parallel. In the specific case of the simulated models, since the considered sizes are reasonably small, each bank contains a single instance.

To access a line of data inside the cache, an internal address is generated which maps the different physical elements composing it (bank, array, sub-array, row, column). This address is derived from the physical address of the request and the result of the tag comparison. Contrarily to other designs, here data are not scattered to different sub-arrays accessed in parallel, but a single one is activated for each access.

The device read and write mechanisms are different if we compare a STT-MRAM cell with an SRAM one, so it is the relative device periphery (e.g. sense amplifiers, precharge and write circuits). However, much of the cache architectural periphery (cache controller logic circuits, buffers) does not need to change. Likewise, the tag array, due to its limited size and access frequency requirements, can remain SRAM-based even when the data array uses a different technology. This is what happens in our model as well: except for the device itself, most of the cache architecture is maintained equal after the technology change. This also ensures a fair comparison and helps obtaining insights on the pure role of technology in the different cache behavior.

To keep the analysis simple, we assume that the cache only operates in two power states: *normal* (active) and *light sleep* (standby). Thanks to non-volatility,

a lower consumption is possible in this last power state for STT-MRAM as opposed to SRAM, since the arrays can be completely shut down.

4.4 Simulated platform

The memory exploration conducted in this work has relied on gem5 [62] as the core element of the software framework. This is an event-driven, system-level simulator, which provides a fully software implementation of several components of a computer system, including parametric CPU models and user-defined cache hierarchies. Additionally, gem5 has complete support for binaries compiled for different Instruction Set Architectures (ISAs), which allows to run benchmarks and evaluate new systems based on simulation data.

The memory model of the simulator has been modified to properly evaluate the impact of NVM-based caches. The main additions to the gem5 classic memory system - specifically, to the cache part - are:

- Asymmetric read and write latency model
- Implementation of cache banks

These modifications are not to be considered an original contributions of this thesis, as they were derived from previous work. However, these features have been reworked and extended, including additional metrics to keep track of the origin of bank conflicts. The custom version of the simulator used for the experiments is available on a public repository [1].

4.4.1 CPU models

The simulated CPU models have been chosen to mimic common ARM Cortex-A application processors, in terms of pipeline, frequency and cache parameters. Specifically, two models have been employed: one in-order CPU, similar to a

[1] `https://github.com/artecs-group/gem5-artecs`

Cortex A53, and an out-of-order CPU, similar to a Cortex A15. Simpler in-order cores are a common choice for low-power IoT devices, while edge devices demand for an extra amount of computational power that an out-of-order architecture can provide, so they are both relevant in this context. The out-of-order model is the same that was used for previous work by Evenblij et al. [28], while for the in-order core the ARM *High Performance In-order (HPI)* [7] gem5 model has been adopted. A list of key parameters can be found in Table 4.1, focusing on operating frequencies and cache configuration.

The processor models derive from original gem5 classes, namely MinorCPU [88] and DerivO3CPU [89]. MinorCPU models a superscalar in-order CPU with four pipeline stages (Fetch1, Fetch2, Decode, Execute). DerivO3CPU was designed with the Alpha 21264 [46] as reference, having seven pipeline stages (Fetch, Decode, Rename, Issue/Execute/Writeback, Commit). Being an out-of-order CPU model, the latter introduces a Reorder Buffer (ROB) and logic to perform register renaming and track in-flight instructions and respective dependencies. The instruction retiring is performed in-order. Both models are highly configurable and can be tuned to behave like specific commercial CPUs with a reasonable accuracy [5, 13, 34].

Requests that need access to the memory are directed to a Load-Store Queue (LSQ), which implementation depends on the specific CPU model. For instance, in the MinorCPU this element consists of three independent FIFO queues: one for requests that are waiting for TLB lookup (*requests queue*), one for issued requests waiting for response (*transfers queue*), one for completed stores waiting for commit (*store buffer*). This last buffer is always checked first when new requests arrive, to satisfy them if data are already available. The DerivO3 implementation, instead, simply splits the LSQ in two queues (a *load queue* and a *store queue*). In the in-order CPU model used for this study, the requests queue size is 1, the transfers queue size is 2, the store buffer size is 5. In the out-of-order model, both the load queue and store queue have a number of entries equal to 16.

Both the in-order and out-of-order CPU models employed in this analysis make use of a branch predictor, respectively a tournament one (similar to the one present in the Alpha 21264 [46]) and a bi-mode one [58].

4.4.2 Cache configuration

The simulated cache subsystem consists of modern non-blocking caches, with the presence of MSHRs and a write buffer for each level. The caches are modeled with separate tag and data arrays, with a set-associative placement policy. The write policy is write-back, i.e. the main memory is only updated when data are evicted from the caches. The caches are write-allocate, which means that a new line is created on write misses. Regarding the replacement policy, a traditional Least Recently Used (LRU) has been chosen for cache data. It would have been interesting to analyze the impact of other policies as well, but this is left for future work.

All the caches in the gem5 classic memory system are non-inclusive. Specifically, in the experiments the first level caches have been chosen to be mostly inclusive of the LLC. This implies that both levels are filled after a miss from the latter, but inclusivity is not enforced when data are removed from it (there is no back invalidation).

The memory subsystem selected for the two CPU models is a 2-level cache hierarchy, with the last-level cache being either SRAM-based or STT-MRAM-based. LLC access latencies are reported in Table 4.2. These parameters have been extracted from the calibrated memory models described in the previous section. As it will be shown several other times, STT-MRAM parameters are actually divided based on two possible scenarios: a *typical case* and a *worst case*, depending on the effect of process variation, which can negatively affect the characteristics of the final device. While it constitutes an issue for the SRAM technology as well, the model that has been employed in this exploration does not include this pessimistic case, so only the typical case will be reported.

In the LLC architectural model, the last significant part of the cache line address is used for the bank indexing, with a direct mapping function based on the bits value. This means that contiguous lines will be distributed across different banks, to enable parallel accesses in case of subsequent data request. Lower level caches (instruction and data L1) do not include a bank model, i.e. they possess a virtual number of banks equal to the total number of cache lines. Given their low access latency, this simplification does not have any significant

Table 4.1: Simulated CPU models [64]

Model	Frequency	L1I		L1D		L2	
		Size	Assoc.	Size	Assoc.	Size	Assoc.
In-order	1.4GHz	16kB	2	16kB	4	256kB	16
Out-of-order	2.1GHz	32kB	2	32kB	2	1MB	16

Table 4.2: LLC latencies, in cycles [64]

Model	SRAM		STT-MRAM typical		STT-MRAM worst	
	Read	Write	Read	Write	Read	Write
In-order	8	8	8	14	14	28
Out-of-order	12	12	12	21	21	40

impact on accuracy.

4.5 Application set

The applications to test have been selected as a subset of the SPEC CPU2017 [12] benchmark suite. These benchmarks represent a common option to benchmark the performance of modern computers. To reduce the effort associated to simulating a large number of benchmarks, a only a subset of them has been selected. In order to retain the general characteristics of the set and not lose potential domain coverage, the methodology presented by Song et al. [84], has been followed, to discard redundant benchmarks with a high degree of similarity. This methodology has been integrated with information from the work of Singh and Awasthi [82] to make sure to have a balanced set of compute-intensive and memory-intensive applications. Among the input data sets made available by SPEC for each benchmark, the *train* ones have been chosen, which are less computation heavy than *ref* but still allow for reasonable analyses.

The simulation of an entire system through software tools, even with abstract models that are not entirely cycle-accurate, can be incredibly slow. This depends on the complexity of the simulated platform and on the intensity of the benchmark, nonetheless it is not uncommon to expect a slowdown of a factor

500-1000 compared to the native execution. This means a single simulation, with a specific CPU and cache configuration, can take weeks if not months. For this reason, common techniques to speed up simulations have been employed in this work. Specifically, the task has been accomplished with the help of the SimPoint [81] utility. SimPoint works by clustering Basic Block Vectors (BBVs), which can be seen as the "signature" of an application in different sampled execution windows (*intervals*). The BBVs keep track of the frequency of execution of each basic block, i.e. sequence of instructions with a single entry point and a single exit point. By analyzing the similarity of the BBVs, redundant phases can be discarded, as their behavior is already represented by a single window - the centroid of the cluster. Resulting simulation data for the various windows are then weighted and recombined to get the overall metrics, which should be close to the ones obtained by executing the full benchmark - at least in relative terms.

From a first execution of the benchmarks, the gem5 simulator is able to dump a checkpoint for each of the selected simulation points. A checkpoint is essentially a snapshot of the simulation (including CPU registers, execution context, main memory content), which can be loaded in successive experiments to restart the execution from a specific point in time instead of re-running the entire application. This greatly speeds up the entire simulation process when multiple configurations have to be tested. One limitation of this approach is that the caches content is not saved in the checkpoint (cache configuration can change from one simulation to another). Hence, the first memory accesses when restoring from a checkpoint will inevitably result in cold misses. The impact of such misses becomes less relevant if the interval size is large enough, as in this case.

The final selected benchmark composition, together with resulting information from the SimPoint methodology, is shown in Table 4.3. Here, one interval represents a window of 100 million instructions.

A few more data are shown that can be useful to characterize the benchmarks. Figure 4.5 represents a breakdown of the types of issued instructions. The benchmarks exhibit a heterogeneous behaviour, with *607.cactuBSSN* being the most memory intensive (more than 50% of operations are loads or stores) and

Table 4.3: Selected benchmarks (*train* set) [64]

Benchmark	Type	Domain	Intervals	Clusters
602.gcc (*200.c*)	INT	GNU C compiler	1012	19
605.mcf	INT	Route planning	1349	24
607.cactusBSSN	FP	Physics: relativity	1547	23
623.xalancbmk	INT	XML to HTML	2555	14
625.x264	INT	Video compression	3736	19
628.pop2	FP	Ocean modeling	6459	24
638.imagick	FP	Image manipulation	3027	11
641.leela	INT	AI: Monte Carlo	3958	18
649.fotonik3d	FP	Electromagnetics	1719	15
654.roms	FP	Ocean modeling	12017	22

638.imagick being the most CPU intensive.

Figure 4.6, instead, depicts the amount of Misses Per Kilo Instructions (MPKI), relative to the two executed configurations. This represents a good metric to evaluate the cache performance of one application in a specific system. Again, the variety of behaviors confirms the differentiation of the benchmark set. The application with most misses in both L1D and L2 appears to be *605.mcf*, which likely indicates a large working set size which does not fit in either level of cache. Benchmarks like *607.cactuBSSN*, instead, have a lot of L1D misses but fewer L2 misses, which means that most memory requests are served by the LLC. It is also possible to notice a strange phenomenon: the amount of data cache misses is generally higher in the out-of-order core, but the L2 misses are lower. This is probably a consequence of the MSHRs acting as a good filter for L2 accesses, while having to handle an additional number of misses (also secondary) due to the out-of-order capabilities.

4.6 Performance impact

The first experimental results that have been evaluated are the simulation times of the fully-SRAM cache configuration (*traditional*) and the SRAM L1

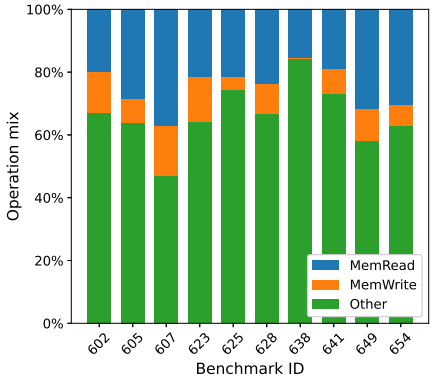


Figure 4.5: Instructions classification [64]

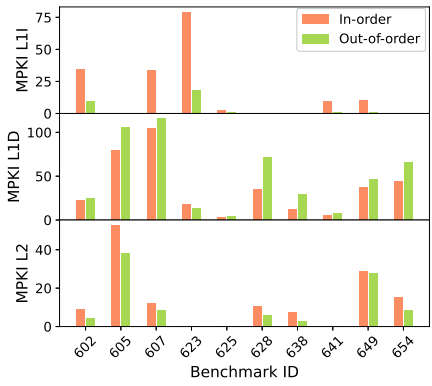
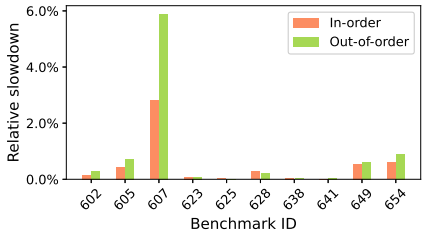
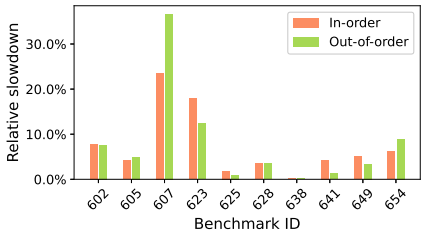


Figure 4.6: Cache MPKI [64]



(a) STT-MRAM typical case

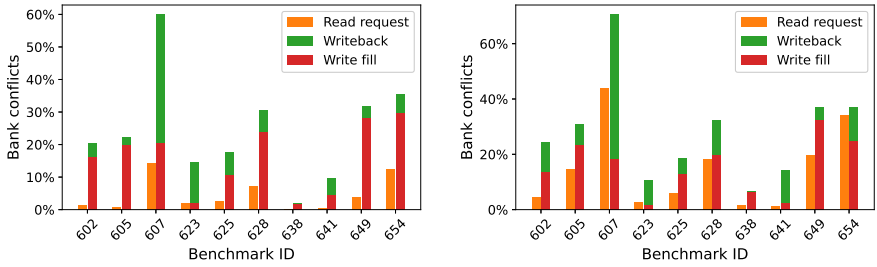


(b) STT-MRAM worst case

Figure 4.7: Relative slowdown per benchmark (8 banks) [64]

+ STT-MRAM LLC configuration (*hybrid*). Since the second configuration is always slower than the first one, the results are represented in form of relative slowdown, in Figure 4.7. The number of cache banks has been chosen to be equal to 8 for both configurations.

One of the immediate conclusions that can be drawn is that, if in the worst case the performance penalization can be as high as 40%, the typical case lies on more tolerable values, always under 6% and usually way smaller. In this case, the slowdown is entirely related to the increased write latency, since the read latency is estimated to be the same, as reported in Table 4.2.



(a) In-order (b) Out-of-order

Figure 4.8: Percentage and categorization of bank conflicts (STT-MRAM worst case, 8 banks) [64]

Another effect that is immediately visible is that the slowdown is dependent on the employed CPU model. It is generally higher in the out-of-order core, because the higher instruction-level parallelism creates more pressure on the memory system. The benchmark that is affected most by the technology change is without doubts *607.cactuBSSN*, which has a high number of memory operations (Figure 4.5) and the highest memory bandwidth [82]. Additionally, previous considerations on the cache efficiency for this specific benchmark still stand. On the other side of the spectrum, *638.imagick* seems to be completely unaffected by the change, which is natural due to the small amount of memory operations and almost absent writes. *623.xalancbmk* is penalized by the technology change, but only in the worst case, due to the increased read latencies.

4.6.1 Origin of cache conflicts

The slowdown results that have just been presented show a good correlation with the distribution of LLC bank conflicts. A conflict happens when an access is put on halt because the bank is already processing another operation. The bank could be busy because it is processing a read access, a writeback from a lower level or a write fill from downstream.

The LLC conflict distribution is plotted in Figure 4.8. The red and green bars show the percentage of conflict-generating write operations, while the

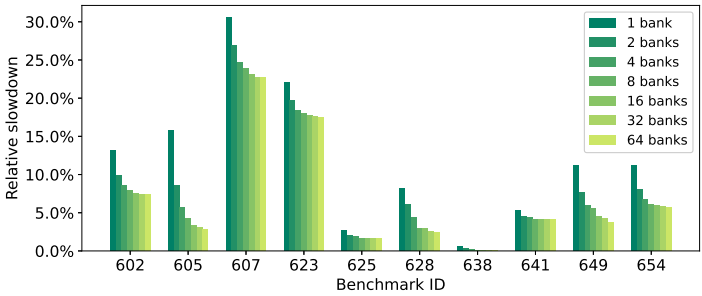


Figure 4.9: Relative slowdown varying the number of banks (In-order, STT-MRAM worst case) [64]

orange bar represents the percentage of conflicts arising from read operations. Unsurprisingly, the highest amount of conflicts happens in *607.cactuBSSN*. This is also the only case where writebacks generate more conflicts than write fills, due to the higher traffic between the L1 and LLC. *628.pop2* is also an interesting case, since despite the somewhat high number of bank conflicts its performance penalization is very contained. This benchmark resembles *649.fotonik3d* and *654.roms* in terms of data cache behaviour (Figure 4.6), but has less L2 misses, and a more stable bandwidth according to literature [82]. *638.imagick* has a more predictable behavior, since it has virtually no conflict and no slowdown.

Comparing the in-order core to the out-of-order core, there is a slight increase in the number of conflicts, which is reasonable due to the additional pressure on the memory system.

4.6.2 Effect of cache banks on performance

It has already been stated that STT-MRAM memories benefit from a higher number of banks, thanks to the ability to better hide the increased latencies. Figure 4.9 represents the relative slowdown of a sample configuration with a variable number of banks from 1 to 64. As imagined, there is a decreasing trend on the execution time for all the benchmarks. However, it is possible to appreciate strong differences on the speedup amount and slope. *605.mcf* is the

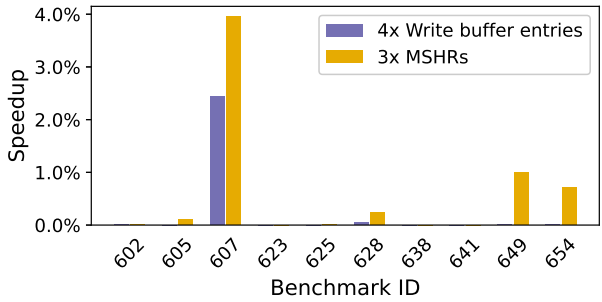


Figure 4.10: Effect of L1D write buffer and MSHRs increase (Out-of-order, STT-MRAM worst case, 8 banks) [64]

application that most benefits from the banks increase.

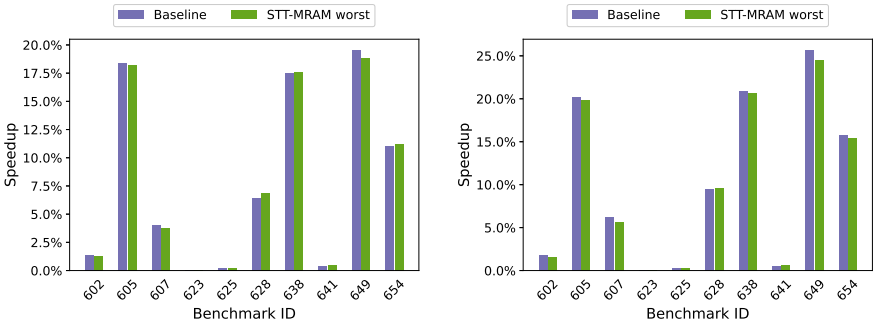
4.6.3 Effect of write buffer and MSHRs

With the goal of evaluating their impact on relative performance, support structures in the L1 data cache have been enlarged in successive experiments. The write buffer size has been increased from 4 to 16 entries in the in-order core, and from 16 to 64 entries in the out-of-order core. The MSHRs have also been increased by 3 times.

Results are shown in Figure 4.10. Except for *607.cactuBSSN*, where there is an improvement up to 4% on the execution time, the effect is very low on all the other benchmarks.

4.6.4 Effect of hardware prefetcher

Following the series of performance tests, the impact of enabling hardware prefetching has been evaluated. Two different configurations have been tried: one with a degree-4 stride prefetcher on the L1 data cache, the other with an additional degree-8 prefetcher on the last-level cache. Results are shown in Figure 4.11.



(a) Data cache prefetcher

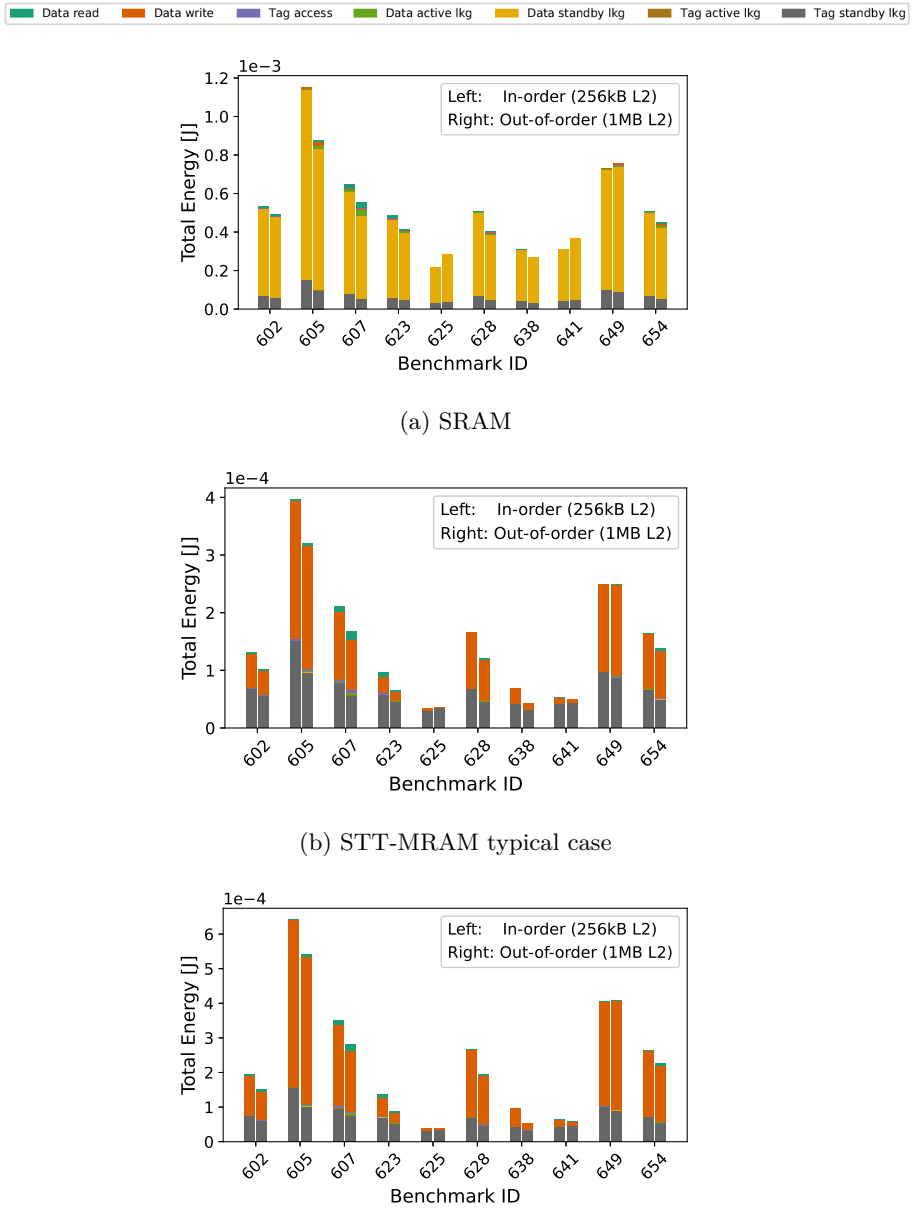
(b) Data cache + LLC prefetcher

Figure 4.11: Stride prefetcher effect on performance (In-order, 8 banks) [64]

While enabling hardware prefetching has a positive impact in most of the analyzed benchmarks, it is easy to see how it affects in a mostly identical way both technologies, i.e. it does not improve the STT-MRAM slowdown by a significant extent. Focusing on the presented data, there are situations where the prefetcher is very helpful in reducing the simulation time, like with *605.mcf*, *638.imagick*, and *649.fotonik3d*, while it turns to be ineffective with *623.xalancbmk*, *625.x264* and *641.leela*. The results with both L1D and LLC prefetchers enabled show a slightly higher speedup compared to the case when only the first one is active.

4.7 Energy consumption

After evaluating the performance of the mixed-technology system in relation to a series of parameters, an extensive energy study has been conducted, first with the calibrated in-house energy model and then with models produced by NVSim [25].

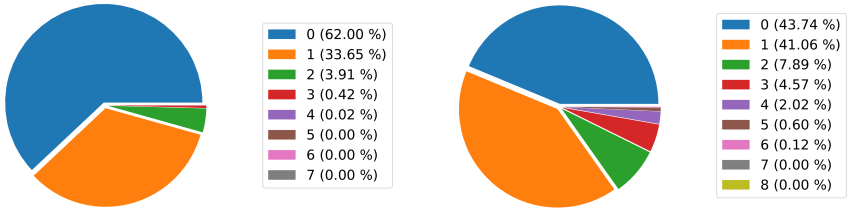


(a) SRAM

(b) STT-MRAM typical case

(c) STT-MRAM worst case

Figure 4.12: In-house accurate energy model - Average LLC energy consumption per $100 \cdot 10^6$ instructions (8 banks) [64]



(a) In-order (b) Out-of-order

Figure 4.13: LLC banks utilization of *607.cactuBSSN_s* (8 banks) [64]

4.7.1 In-house model

The first energy results (for the in-house model) are presented in Figure 4.12. It is immediately visible how the SRAM results are dominated by the static leakage. Analyzing the bank activity data, it turns out that the LLC bank utilization is on average very low, which explains this finding. Even the benchmark that best exploits the bank level parallelism, 607.cactuBSSN, does not use more than one bank for most of its execution time (Figure 4.13).

The situation radically changes when STT-MRAM is used instead of SRAM. the only leakage energy that is present - and becomes more pronounced - is the one associated to the tag array. The main contribution on the energy consumption, in this case, is given by the write operations.

Overall, it appears that the cache energy significantly benefits from the STT-MRAM technology, especially for benchmarks that are not write-intensive (e.g. *638.imagick*). This is true provided that a reasonable amount of banks is chosen. A summary of the LLC energy savings between STT-MRAM and SRAM in this configuration is reported in Table 4.4. The energy reduction is indicated in terms of ratio, using the following formulation:

$$\text{energy saving} = \left(1 - \frac{\text{STT-MRAM energy}}{\text{SRAM energy}}\right) * 100$$

To have a better view on the overall impact of the LLC technology change, energy numbers from the L1 cache have been calculated and considered as a

Table 4.4: STT-MRAM energy saving (8 banks - LLC only) [64]

Benchmark	STT typical		STT worst	
	In-order	Out-of-order	In-order	Out-of-order
602.gcc	75.37%	79.25%	63.26%	69.59%
605.mcf	65.63%	63.41%	44.33%	38.39%
607.cactusBSSN	67.29%	69.96%	45.88%	49.19%
623.xalancbmk	80.31%	83.99%	71.90%	78.38%
625.x264	84.44%	87.40%	82.16%	86.46%
628.pop2	67.07%	70.14%	47.24%	52.04%
638.imagick	77.58%	84.00%	68.58%	79.77%
641.leela	83.16%	86.26%	79.27%	84.18%
649.fotonik3d	65.84%	67.20%	44.69%	46.16%
654.roms	67.71%	69.27%	48.21%	49.77%

fixed part of the energy consumption for all the configurations. To obtain the data, the same SRAM in-house model has been utilized, configured to match the lower cache size. As it is logical to think, the total energy reduction of the cache subsystem is smaller as opposed to when only the LLC is taken into account. Even so, if we also include the L1 cache energy contribution, the overall estimated energy reduction is around 23% for the worst case and 28% for the typical case, which is still appreciable.

4.7.2 NVSim

The energy analysis has then been repeated using models generated with the help of NVSim [25], which is a memory modelling tool focused on emerging non-volatile memory technologies. The main idea behind this work was, in fact, to configure this tool to produce realistic models to be used in microarchitectural explorations, without having to rely on proprietary solutions.

Despite being well known and used in academia, NVSim presents some issues which are common to other freely available tools. First, it has not been updated within the last 10 years. Then, the provided STT-MRAM model is calibrated

against a 65nm industry prototype, with no guarantee that it can reflect the characteristics of more modern and scaled devices. In this analysis it has been evaluated whether a reasonable choice of parameters can make this tool obtain the same STT-MRAM vs SRAM energy trends as the calibrated in-house models previously used (and described in Chapter 4.3). This evaluation has been conducted without any modification to the NVSim codebase.

NVSim can be configured with a series of parameters relative to the device basic cell, process technology and cache design, that can be set in configuration files. The tool performs calculation based on given parameters to extract energy, latency and area information of a cache based on the selected technology. Being a specialized tool for non-volatile memories, it naturally includes a STT-RAM model that has been used as a starting point.

Internally, NVSim creates a model representing a typical hierarchy present in a memory design. The structure is not very dissimilar from the one reported in Chapter 4.3.1, albeit with different terminology and the absence of instances. The base element is the *subarray*, which contains the memory cells and periphery (row decorders, column muxes and output drivers). A group of subarrays operating together plus the corresponding predecoder block form a *mat*. Finally, the top structure is the *bank*, which connects multiple mats together with a bus or H-tree interconnect. One limitation is that the tool does not go beyond that and does not model a multi-bank device. For a fair comparison with the in-house model, the same multi-bank trends have been extracted from it and applied to the NVSim models.

As a starting point, the sample basic cell model provided by NVSim for STT-MRAM has been slightly tweaked for the experiments. This cell is present in two variants: *conservative* and *aggressive*. The exploration has been based on the latter, which is the most optimized design. The only modification applied to the cell has been reducing the read power from 23.4 uW to 5 uW: the original value is unreasonably high and would surely shorten the device lifespan.

After setting the memory parameters which are specific to the caches to model (size, associativity, line width, technology node), a sensitivity analysis has been conducted based on the *device type* and *optimization target*. The possibilities for

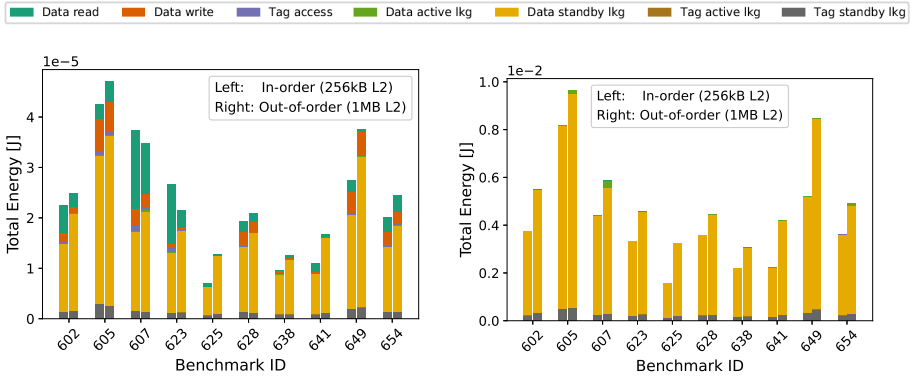
the device type are the ones defined by the International Technology Roadmap for Semiconductors (ITRS): High-Performance (HP), Low Operating Power (LOP), Low Standby Power (LSTP). As for the optimization target, the device can be optimized to obtain the best read/write Energy-Delay Product (EDP), latency, power, or area.

Tables 4.5 and 4.6 report the most convincing models (i.e. the ones that better represent to the trends of the in-house models) that have been extracted from the tool. The first table represents a comparison of the NVSim values with the in-house models data, for both SRAM and STT-MRAM. Since the exact details of the in-house memory models are confidential, the data are represented in terms of order of magnitude. The second table, instead, focuses on the ratios between the technologies, which are the real targets on which the calibration is based.

The first model has been chosen focusing on the STT-MRAM to SRAM consumption ratio. The criterion has been to minimize the Euclidean distance between the trends shown in Table 4.6 and the reference values. The winning model according to this logic, for a LLC size of 1 MB, is the LSTP variant optimized for read energy-delay product (*LSTP - ReadEDP*). Comparing the NVSim energy results depicted in Figure 4.14 with the in-house ones (Figure 4.12), it is possible to note that in this variant the SRAM leakage is way smaller, which also lowers the tags static consumption in the STT-MRAM configuration. In this case, the impact of the write energy is the STT-MRAM much higher in relative terms, and makes the entire energy consumption higher than SRAM, which is against the initial hypotheses and previous data.

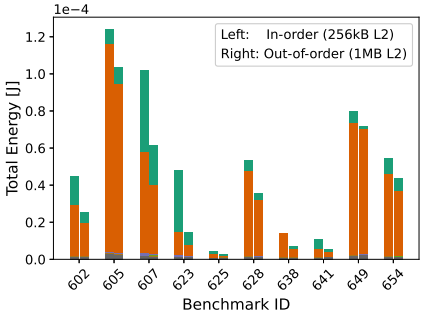
The second model has been selected with the goal of maintaining the energy advantage of STT-MRAM over SRAM. In this regard, the best choice is the LOP variant optimized for leakage power (*LOP - Leakage*). Looking at the plots, the energy breakdown appear to be more in line with the previous findings. However, the SRAM leakage power is one order of magnitude higher.

Unfortunately, we can conclude that not a single configuration closely matched the characteristics of the reference in-house model, not even in relative STT-MRAM vs SRAM terms. If one of the models has to be chosen, the *LOP* -

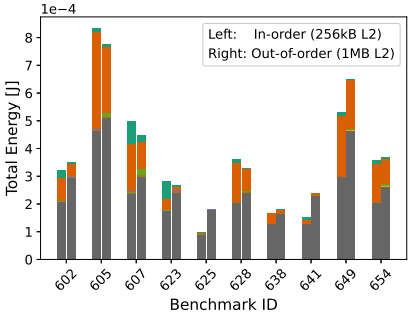


(a) SRAM (LSTP, ReadEDP opt.)

(b) SRAM (LOP, Leakage opt.)



(c) STT-MRAM (LSTP, ReadEDP opt., aggressive)



(d) STT-MRAM (LOP, Leakage opt.)

Figure 4.14: NVSim models - Average LLC energy consumption per $100 \cdot 10^6$ instructions (8 banks) [64]

Leakage variant with a manual reduction of the leakage can be considered as representative enough.

4.8 Pareto analysis

Lastly, a Pareto analysis has been conduced to analyze which combination of memory technology and number of banks gives the best set of energy and performance metrics.

Table 4.5: NVSim energy models comparison (1MB cache, single bank, $0.5 <$ mantissa ≤ 5) [64]

	SRAM	
	LSTP - ReadEDP	LOP - ReadEDP
Read Energy	E-10	E-11
Write Energy	E-10	E-11
Leakage Power	E-04	E-01
R/W Energy Ratio	1.20	1.20
	LSTP - Leakage	LOP - Leakage
Read Energy	E-10	E-10
Write Energy	E-11	E-11
Leakage Power	E-04	E-02
R/W Energy Ratio	15.31	15.01

	STT-MRAM aggressive	
	LSTP - ReadEDP	LOP - ReadEDP
Read Energy	E-10	E-10
Write Energy	E-10	E-10
Leakage Power	E-04	E-03
R/W Energy Ratio	0.44	0.53
	LSTP - Leakage	LOP - Leakage
Read Energy	E-10	E-10
Write Energy	E-10	E-10
Leakage Power	E-04	E-03
R/W Energy Ratio	0.67	0.66

	Reference values	
	SRAM	STT-MRAM
Read Energy	E-10	E-10
Write Energy	E-10	E-10
Leakage Power	E-03	E-04
R/W Energy Ratio	1.40	0.12

Table 4.6: NVSim STT-MRAM vs. SRAM consumption parameters ratios (1MB cache, single bank) [64]

		Read Energy	Write Energy	Leakage Power
		STT-MRAM aggressive / LSTP - ReadEDP		
SRAM	LSTP - ReadEDP	1.11E+00	3.01E+00	4.82E-01
SRAM	LOP - ReadEDP	2.45E+00	6.62E+00	1.66E-03
SRAM	LSTP - Leakage	3.23E-01	1.12E+01	5.25E-01
SRAM	LOP - Leakage	7.04E-01	2.39E+01	1.81E-03
		STT-MRAM aggressive / LSTP - Leakage		
SRAM	LSTP - ReadEDP	1.31E+00	2.34E+00	4.38E-01
SRAM	LOP - ReadEDP	2.88E+00	5.14E+00	1.51E-03
SRAM	LSTP - Leakage	3.80E-01	8.66E+00	4.77E-01
SRAM	LOP - Leakage	8.28E-01	1.85E+01	1.65E-03
		STT-MRAM aggressive / LOP - ReadEDP		
SRAM	LSTP - ReadEDP	1.07E+00	2.41E+00	1.95E+01
SRAM	LOP - ReadEDP	2.36E+00	5.29E+00	6.71E-02
SRAM	LSTP - Leakage	3.11E-01	8.92E+00	2.12E+01
SRAM	LOP - Leakage	6.78E-01	1.91E+01	7.32E-02
		STT-MRAM aggressive / LOP - Leakage		
SRAM	LSTP - ReadEDP	1.15E+00	2.09E+00	6.35E+00
SRAM	LOP - ReadEDP	2.53E+00	4.58E+00	2.19E-02
SRAM	LSTP - Leakage	3.33E-01	7.72E+00	6.92E+00
SRAM	LOP - Leakage	7.27E-01	1.65E+01	2.39E-02

Reference values

Read Energy	Write Energy	Leakage Power
2.30E-01	2.62E+00	1.20E-01

Results of this analysis are shown in Figure 4.15. The size of the dots represents the number of banks, from 1 (smallest dot) to 64 (largest dot). From these images it is possible to obtain a series of key messages:

- As expected, the STT-MRAM worst configuration is always sub-optimal.
- The energy initially decreases with the numer of banks (due to the smaller arrays), then starts increasing again (due to the additional logic and longer interconnect cost).
- The inversion point, where the energy starts to grow again, is different for SRAM and STT-MRAM.

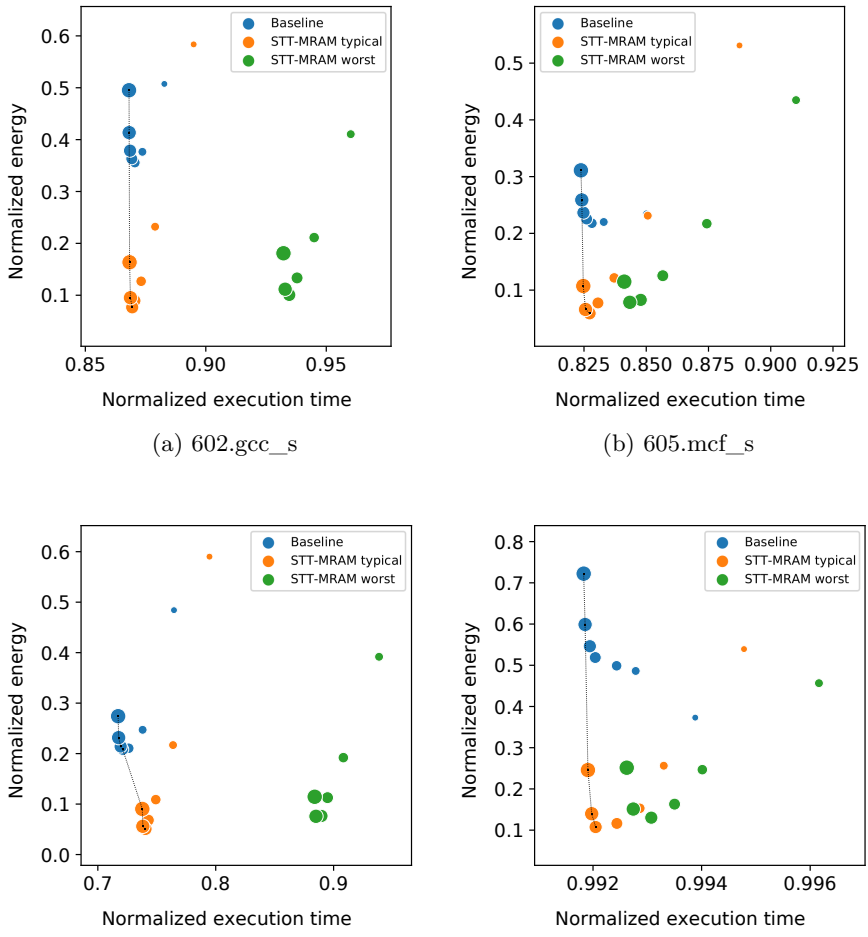
On average, from this analysis it emerges that the optimal amount of banks is 4 for SRAM and 16 for STT-MRAM.

As a final experiment, given the results just obtained, the performance of a 4-banks SRAM LLC is compared to a 16-banks STT-MRAM in Figure 4.16. The comparison is fair also from an area point of view, since the increased density allows to accommodate more banks. Except for *607.cactuBSSN* and *654.roms*, this STT-MRAM configuration performs even better than SRAM in the typical case. This is not true for the worst case, but the generally contained slowdown and the consistent LLC energy saving (73% - 77%) could still well justify its adoption.

4.9 Chapter conclusions

The goal of the work presented in this chapter was to show that the adoption of STT-MRAM for large-enough caches allows to reduce the overall power consumption of a system without having a significant impact on performance. The target of the analysis are embedded application processors, where the energy consumption is of critical importance.

The results clearly indicates that an LLC technology replacement is feasible and beneficial for energy, and that an informed selection of the number of



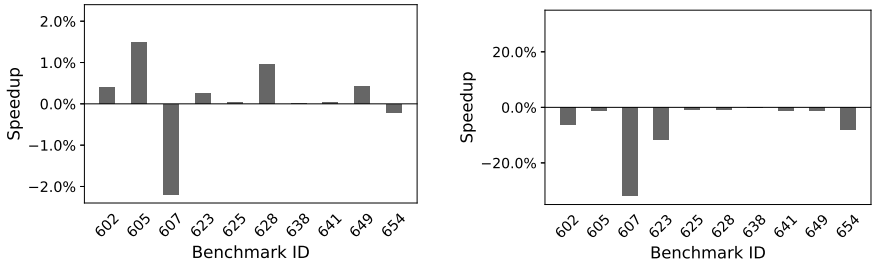
(a) 602.gcc_s

(b) 605.mcf_s

(c) 607.cactuBSSN_s

(d) 638.imagick_s

Figure 4.15: Energy vs. performance with Pareto front for different number of LLC banks (In-order) [64]



(a) Typical case

(b) Worst case

Figure 4.16: Performance using STT-MRAM w/ 16 banks instead of SRAM w/ 4 banks (Out-of-order) [64]

banks at design time can help reducing the latency bottleneck by increasing the request parallelism, closing the gap with SRAM. The Pareto energy-performance analysis also showed that STT-MRAM benefits from an additional number of banks compared to SRAM, which is easy to achieve thanks to its higher density.

The energy data, obtained with calibrated in-house energy models for SRAM and STT-MRAM, show that the elimination of the static leakage lowers the consumption of the LLC (and the entire cache subsystem) by a fair amount, despite the increase in write energy. A calibration of the NVSim modeling tool to obtain the same energy trends between STT-MRAM and SRAM was not successful, as none of the explored configurations reflect the characteristics of the in-house models, even in relative terms. Deeper modifications to the tool source code and mathematical models are likely to be needed for a good match with more modern technology processes.

Chapter 5

Application Memory Behaviour

In this chapter, a methodology to detect patterns in memory accesses of applications is introduced. The patterns information can be used for different purposes, but in this context it serves as an input for a data placement algorithm in hybrid cache-SPM architectures. The specific use case, with a detailed description of the architecture and some preliminary results, will be described in the next chapter. Most of the concepts here described have been published in an original article [63].

5.1 Goals and constraints

In order to be able to perform application-specific optimizations, both from a software and from a hardware perspective, a computer architect needs to have a deep understanding of how the application interacts with the system. One of the most important aspects is knowing what is the application memory access behaviour. With the growing complexity of modern systems, especially

in multi-core and many-core systems, the way the applications use memory resources can have an enormous impact on the execution metrics.

If we neglect optimizations automatically applied by the compiler, in most imperative languages it is possible to understand exactly how virtual memory resources are utilized from the original source code of one application. Specifically, it is easy to see in which procedure a certain data object is generated, what is the way it is accessed, and when its memory content is freed. Of course, there is a lot of variability given the dynamic behaviour of certain applications, in terms of control flow and dynamic resource allocation. Nonetheless, for simpler and more regular applications the behaviour is somewhat more deterministic, and such applications are used in a wide amount of domains: audio and video processing, filters, machine learning inference, etc.

Unfortunately, it is very possible that the system architect does not possess the source code of a target application, and cannot perform any memory consideration or optimization based on that. This is the case with closed-source or legacy software, which is not uncommon to see in industrial contexts. This is a strong limitation, but does not prevent the possibility to perform other types of analyses.

5.2 Dynamic binary instrumentation

Information on the memory behaviour can be also extracted from an application at execution time. This is possible thanks to Dynamic Binary Instrumentation (DBI) frameworks. The concept is simple: before being executed, application instructions are modified on-the-fly, to add custom analysis routines or additional pieces of code to execute. This process entails a certain amount of overhead, but it constitutes a quite flexible and powerful profiling technique. DBI frameworks usually allow developers to create user-defined tools using their APIs, which makes it easier to fully leverage their analysis capabilities. One example of DBI framework is Intel Pin [41], which is what has been used for the first steps of this work. Its syntax is simpler than alternative frameworks, but it only supports x86-64 architectures, which is an important limitation to keep in mind.

Nonetheless, memory behaviour is only partially ISA-dependent (if we exclude vectorization support, differences are mostly related to calling conventions), which makes the proposed approach generalizable in future work.

5.3 Memory regions of an application

All the applications occupy a certain amount of memory when executed. According to the type of information, this can be stored in different virtual regions which are mapped to a specific address range in the main memory. If we look at a compiled C/C++ application, there are several possible regions which form the virtual memory layout (Figure 5.1):

- *Text*, which contains the code of the application being executed.
- *Initialized data*, which stores all the initialized global, static, constant and external variables.
- *Uninitialized data*, which stores all the uninitialized global and static variables.
- *Heap*, which stores dynamically allocated data.
- *Stack*, which stores local variables and some function arguments.

As it can be seen in the figure, the biggest part is composed of the stack and heap. The two regions do not have strongly defined boundaries, but can dynamically grow in either direction.

This premise is important, since from an analysis and optimization point of view focusing on one region or another requires different approaches. The methodology here presented has dynamically allocated data as target, since most of the benchmarks that have been analyzed reserve space for their data objects at runtime.

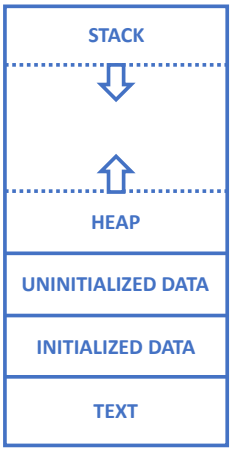


Figure 5.1: Memory layout of a C/C++ application

5.4 Profiling methodology

What is proposed in this chapter is an offline profiling methodology which allows to detect:

- The nature of the data objects allocated by an application.
- Patterns in the memory access of each data object.

The methodology requires a first execution step to collect the needed information, then data are post-processed to extract the patterns. As a sample use case for this methodology, the metrics that characterize the patters are then used to formulate a cache-SPM placement strategy to be applied to subsequent executions. It is assumed that the executed applications are not too dynamic, and the control flow or shape of the data structures does not change significantly. This is the case for the simple benchmarks used in the experiments [69]. This chapter is focused on the memory analysis and placement definition algorithm, while the details on the new architecture and experiments will be covered in Chapter 6.

Here follows a definition of the different steps of the methodology.

5.4.1 Memory objects extraction

The application is executed under the binary instrumentation framework to get information about all the data objects created at runtime. In the context of the present work, Intel Pin [41] has been employed in conjunction with a specific tool from previous work [15], but this approach is easily portable to other frameworks. A *data object* can be defined as the memory structure resulting from a dynamic memory allocation call. Examples of memory allocation functions are `malloc()`, `calloc()` and `posix_memalign()`. Every structure is characterized by a *start address* and a *size*. There are application that employ complex data structures which are dynamically expanded or shrinked, involving multiple allocation calls: the tool allows to group them together and consider them as a single object.

5.4.2 Detailed access trace

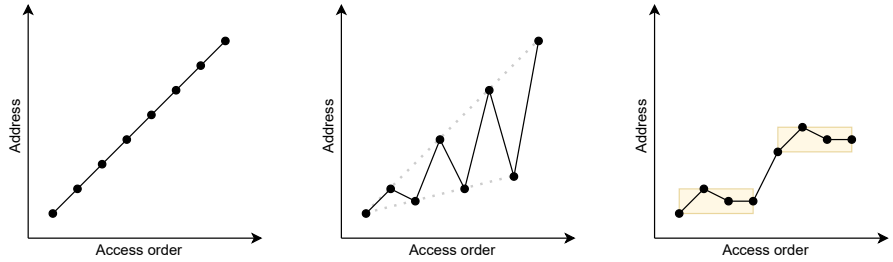
For each one of the data objects, or for a chosen target subset, a detailed access trace is extracted and dumped to a file. This operation can be done with DBI as well, concurrently with the previous step or in a second moment (provided that the application is predictable and the inputs are the same).

The traces have to contain, for each access, the *target address*, the *requested size* and the *type of operation* (read or write). In order to be able to recombine all the patterns and generate a global policy in a following step, a *global access index* must also be included.

5.4.3 Pattern analysis

This is arguably the most important step of the methodology. The previously generated access traces are analyzed with a custom tool, to detect series of accesses that can be identified as patterns. The tool that the candidate has developed for this work is able to identify the following types of patterns:

- *Sequential*: a strided series of subsequent accesses (Figure 5.2a)



(a) Sequential pattern (b) Interleaved pattern (degree 2) (c) Signature-based pattern

Figure 5.2: Example of identifiable access patterns [63]

- *Interleaved*: a series of sequential patterns mixed together in time (Figure 5.2b)
- *Signature-based*: a series of locally-irregular, globally regular accesses (Figure 5.2c)

Any sequence of addresses that fits one of the aforementioned types is called an *interval*. Subsequent intervals having the same characteristics form part of a bigger pattern, and can be considered as a single entity.

The developed tool is very simple by design, in view of a potential future hardware controller implementation. The accesses are analyzed with a sliding window, checking if the order of the addresses matches one of the categories defined above. To detect signature-based accesses, a buffer containing part of the past access history is also needed, to check for a matching irregular sequence. Both the sliding window and the history buffer are configurable in size.

5.4.4 Relevant metrics

The last step of the methodology consists in extracting a series of useful metrics from the obtained intervals.

In the context of this work, the goal is to find patterns that would benefit from being accessed from a scratchpad instead of a cache. The scratchpad access

cost, in terms of energy, is lower than a cache of similar size. Hence good candidates for the mapping are frequently accessed data. Furthermore, patterns that make an inefficient use of the memory spatial locality, using only a fraction of a fetched cache line, are potential candidates too, provided that there is the possibility to access such data in a more efficient way (more on this later).

Given the criteria that have been listed, the choice of metrics to be calculated for each pattern is:

- *Inter-interval reuse factor*: percentage of repeated accesses within the same interval.
- *Intra-interval reuse factor*: percentage of repeated access across subsequent intervals.
- *Data compaction factor*: percentage of space efficiency of the interval. This is calculated as the number of (potentially) accessed cache lines divided by the number of virtually required cache lines, in case of compacted data.

For more details about the mathematical formulation of each one of these metrics, please refer to the original paper [63].

5.5 SPM placement algorithm

The metrics obtained with the presented methodology can be helpful to decide how to manage data in a hybrid cache-SPM system. A simple but effective heuristic policy has been generated to be used in the context of this work, for the architecture and the experiments described in Chapter 6. This policy is reported in Algorithm 1. The basic idea is to use the SPM for intervals which can obtain the most advantage out of it, based on the reuse and compaction principles that have been previously discussed.

In order to decide which intervals to place in the SPM, a combination of the three metrics (inter-interval reuse, intra-interval reuse, data compaction) is used. In this first version of the algorithm, it is a simple weighted linear combination

of the factors. Since the scratchpad is limited in size, not all the intervals from multiple data objects can fit. For this reason, in case of overlap only the intervals with the best combination of factors are selected, while the others are discarded. The algorithm is, in essence, a formulation of the Knapsack problem.

Here follows a brief description of the algorithm. After the analysis phase, in which a list of intervals is produced for each of the data objects, this information is collected in a global view and used as input for the script. For each of the intervals, it is evaluated whether there is any other interval which overlaps in time with it (`getOverlapping()`). If this is the case, and if the amount of data used in the intervals exceeds the capacity of the SPM, a value is computed for each of the competing intervals and the one with the lowest value is discarded. The process is repeated until the resulting subset fits the scratchpad size. It is important to note that there is a boundary on the maximum interval size, which means that a single interval can always be stored into the SPM.

The `getValue()` function is used for the interval value computation, extracting the inter-interval reuse (`calInterReuse()`), intra-interval reuse (`calcIntraReuse()`) and compaction factor (`calcCompRatio()`) of its elements and returning a weighted sum of the metrics. In this first version of the algorithm, both reuse factors are considered as a single metric, selecting the highest value of the two. The reuse weight (`r_wgt`) and compaction weight (`c_wgt`), after an initial evaluation, have been set to 0.5 each, not giving priority to a metric compared to the other. Different possibilities have been explored, e.g. an unbalanced choice of 0.7 / 0.3, but the balanced choice is the one that provides better results on average.

The details of how the algorithm is effectively enforced are left for Chapter 6.

Algorithm 1 SPM Mapping Decision Algorithm [63]

```
1: function getValue(int)
2:     reuse ← max(
           calcInterReuse(int),
           calcIntraReuse(int)
       )
3:     compact ← calcCompRatio(int)
4:     return (reuse ∗ r_wgt + compact ∗ c_wgt)
5: end function
6:
7: discarded ← empty
8: for int ∈ intervals do
9:     if int ∉ discarded then
10:        candidates ← getOverlapping(int) + int
11:        while sum(candidates.size) > spm_size do
12:            wst = min(getValue(∀ int ∈ candidates))
13:            discarded ← discarded + wst
14:            candidates ← candidates − wst
15:        end while
16:    end if
17: end for
18: selected ← intervals − discarded
```

Chapter 6

Heterogeneous On-Chip Memory Organization

This chapter presents a heterogeneous cache-SPM architecture optimized for energy consumption, with semi-transparent control and data layout compaction in the SPM. This architecture could be employed in energy-constrained devices optimized for a specific set of tasks, such as edge or IoT nodes. It is less suitable for server contexts, where deterministic regular memory access patterns are less likely to be present. The explanation will be focused on the modifications that have been introduced compared to a traditional core. The architecture is evaluated by executing of a standard set of loop-dominated benchmarks, showing the impact on energy and performance. The policy described in Chapter 5 is used to determine the data placement between cache and SPM. Again, most of the concepts here described have been published in an original article [63].

6.1 Caches and scratchpads

The advantages and disadvantages of scratchpad memories over caches have already been briefly introduced in the previous chapters. Essentially, the lack of

complex management circuitry and the access predictability make these devices less costly to access, both in terms of latency and energy. Also, the SPM area occupation is inferior, for the same memory quantity.

On the other hand, scratchpads require manual data control. Applications have to be modified to introduce instructions that allow to allocate new data or move existing data to the SPM. This can be a hard task if the applications are complex and memory-centric.

Given the generally good behaviour of caches in most situations, it is usually convenient to have both devices in a single architecture. The normal path to data is through the cache, then only in certain specific circumstances the access is served by the SPM. To exploit the advantages of using a scratchpad memory, an optimal data placement algorithm is needed.

Other than modifications to the software, a series of hardware elements are needed in order to enable the use of scratchpads. In the following section, the custom design that has been developed in this thesis is described in details, with its peculiarities and original elements.

6.2 COMPAD architecture

COMPAD (as in *compaction scratchpad*) is the name of the architecture that has been developed in this thesis work. It differs from a normal CPU design only by a few additional components, other the scratchpad memory itself. The introduced modifications not only allow to move data copies to a scratchpad, but they do so only requiring a small amount of software overhead. Furthermore, they allow SPM data transfer with a reduced granularity with respect to a full cache line of 64KiB, dynamically changing the data layout when storing them. This allows to reduce access inefficiencies in some applications.

The elements that compose the COMPAD architecture are depicted in Figure 6.1. Their functionality will be now described.

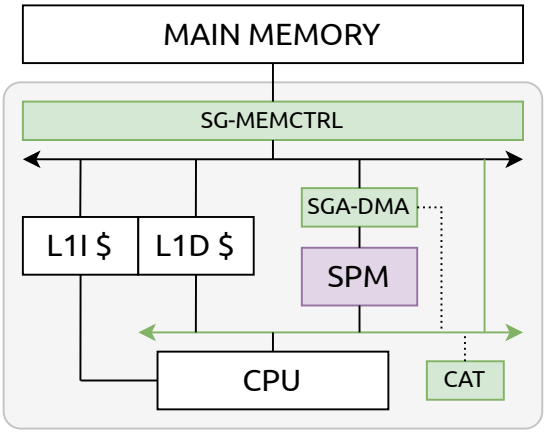


Figure 6.1: COMPAD architecture [63]

6.2.1 SPM

The first element of the COMPAD architecture is, of course, the scratchpad memory. In the specific configuration analyzed in this work, the architecture includes a single SPM with an access latency comparable with the first level cache. Anyway, nothing prevents a designer from using multiple parallel scratchpads, with different sizes or built with emerging memory technologies. Even in this case, the support logic will be the mostly recycled.

The scratchpad memory port is connected to a bus shared with the data cache and the SPM control devices. The other side of the bus is directly connected to the CPU data port. The control devices, that will be now described, can be accessed as Memory-Mapped I/O (MMIO).

6.2.2 SGA-DMA

Like in most architectures using scratchpads, it is advisable to have a programmable controller which can transfer data to the memory with a limited CPU intervention. This is the role of the Direct Memory Access (DMA) controller.

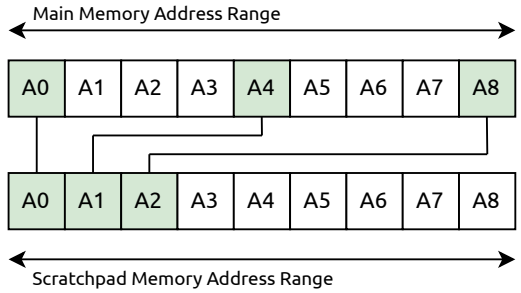


Figure 6.2: Dynamic data layout compaction [63]

The DMA implementation that is used in COMPAD is different than the ordinary. In order to support the data layout compaction in the SPM, the controller must be able to perform strided accesses to memory. Most commercial DMA IPs already possess scattering/gathering capabilities [101, 57, 39]. However, in this case the task is relegated to the memory controller: the SGA-DMA only encodes in the request the information about the pattern to transfer. This allows to maximize the bus width, since data are always sent in compacted form.

The concept of the SGA-DMA resembles other proposals that can be found in the state of the art [102, 106].

6.2.3 CAT

The Compacting Address Translator (CAT) is a component that strongly simplifies the data management from the software perspective. This programmable unit allows to transparently translate physical addresses, to point to the correct position in the scratchpad address space. The component is activated on demand with special instructions. It fully supports the data layout compaction feature, as it can be programmed with the strided pattern information (Figure 6.2).

An optimized design of this component is extremely important, as it stands in the critical path: it has to be interrogated to get the SPM address or fall back to the traditional path if no corresponding entry is found. Fortunately, this

only happens if the access belongs to one of the mapped ranges: this can be verified with a limited amount of range comparisons, which can happen directly at the bus level.

6.2.4 SG-MEMCTRL

The last key component of the COMPAD architecture is the scattering/gathering memory controller, that has already been introduced when talking about the SGA-DMA. When transferring data with the SPM, this memory controller has the task to access the main memory following the provided pattern information, either for properly distributing the data coming from the bust (*scattering*) or for packing non-continuous data to the bus (*gathering*).

Since typical memory interfaces only support burst accesses, this can be done by splitting the request in multiple subrequests according to the stride, and assembling the final block discarding redundant data. A better and more efficient option would be to implement a non-standard memory interface with efficient strided accesses, but it is not required for the component to be functional.

6.3 Simulated platform

For the experimental part of this work, the different elements of the COMPAD architecture have been fully implemented in a custom version of the gem5 simulator [62]. This version is publicly available on the research group repository [1]. The CAT and SGA-DMA have been implemented as MMIO devices, programmable by the user by writing to specific memory locations. A reasonable latency model has been implemented for all the components. Most of the logic is also functionally correct, although a few simplifications have been made which do not affect the final analysis. For example, the inter-reuse logic between intervals has not been functionally modeled, but the number of exchanged words takes data sharing into account.

[1] `https://github.com/artecs-group/gem5-artecs/tree/compad`

The characteristics of the cache and scratchpad have been extracted with the help of a widely used modeling tool, CACTI [98]. Table 6.1 provides a comparison of latency and energy data for different memory sizes, to prove that the initial claims on the benefits of scratchpads are correct.

The specific configuration used for the experiments, instead, is reported in Table 6.2. A single level cache configuration has been assumed for fair comparison with the scratchpad, which directly exchanges data with the main memory. To compensate for the additional SPM area, the data cache size has been reduced by half.

Regarding the enforcement of the SPM mapping policy through binary rewriting, this is left for future work, due to the complexity associated to the task and the limited time available. Instead, the policies have been tested using modified application traces, with injected SGA-DMA and CAT control instructions at the boundaries between selected intervals. In a real scenario (code/binary modification), this is equivalent to adding a few MMIO register writes in proximity of target loops, using loop indices to detect at which point in time the devices have to be (re)programmed. Compared to the modified traces, this should have virtually no impact on the execution, as the only difference is the presence of extra checks. The choice of the intervals to map has been performed according to the methodology presented in Chapter 5.

Most of the details of the CPU and cache behaviour are the same as described in Chapter 4.4.1 and 4.4.2. Due to specific requirements for the employed memory trace format, the DerivO3 CPU model has been used for both traces extraction and replay. However, traces have been modified to make it behave as an in-order CPU, enforcing order dependencies between accesses. This has been done to better represent a typical CPU in the context of low-power embedded SoCs.

The full profiling and simulation flow is depicted in Figure 6.3.

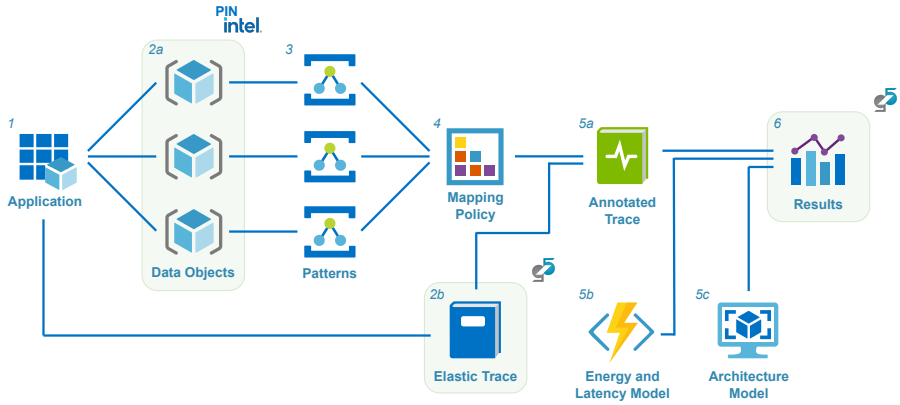


Figure 6.3: Complete experimental analysis and simulation flow [63]

Table 6.1: CACTI caches and SPMs parameters [63]

Device	*Acc. time*	*Read eng.*	*Write eng.*	*Leakage*
Cache 8 KiB	0.44 ns	0.030 nJ	0.028 nJ	4.51 mW
Cache 16 KiB	0.48 ns	0.031 nJ	0.029 nJ	7.86 mW
Cache 32 KiB	0.53 ns	0.033 nJ	0.031 nJ	14.57 mW
SPM 8 KiB	0.18 ns	0.008 nJ	0.010 nJ	3.70 mW
SPM 16 KiB	0.25 ns	0.009 nJ	0.012 nJ	7.15 mW
SPM 32 KiB	0.40 ns	0.010 nJ	0.018 nJ	14.04 mW

Table 6.2: Simulated system details (single core) [63]

CPU	*Architecture:* x86-64
	Ordering: In-Order
	Frequency: 1 GHz
Main Memory	*Size:* 512 MiB
	Standard: LPDDR3
L1I Cache	*Size:* 16 KiB
	4-way set associative
L1D Cache	*Size:* 16 KiB / 8 KiB
	4-way set associative
SPM (Hybrid)	*Size:* 8 KiB

Table 6.3: Selected benchmarks (PolyBench/C Suite) [63]

Benchmark	*Domain*	*Description [69]*
2mm	lin. alg. / kern.	2 Matrix Multiplications
3mm	lin. alg. / kern.	3 Matrix Multiplications
correlation	datamining	Correlation Computation
covariance	datamining	Covariance Computation
deriche	medley	Edge detection filter
doitgen	lin. alg. / kern.	3 Multiresolution analysis kernel
durbin	lin. alg. / solv.	3 Toeplitz system solver
gemm	lin. alg. / blas	General Matrix Multiply
gemver	lin. alg. / blas	Vector Mult. and Matrix Addition
gesummv	lin. alg. / blas	Scalar, Vector and Matrix Mult.
symm	lin. alg. / blas	Symmetric matrix-multiply
syr2k	lin. alg. / blas	Symmetric rank-2k operations
syrk	lin. alg. / blas	Symmetric rank-k operations
trmm	lin. alg. / blas	Triangular matrix-multiply

6.4 Application set

To conduct the experiments, a subset of benchmarks from the PolyBench/C suite [69] has been selected. This suite consists of a compilation of simple numerical computation applications spanning various domains. These applications are loop-dominated, so they are a good target for the proposed optimizations. The selected dataset is *medium*.

The complete list of simulated benchmarks and their description is reported in Table 6.3.

6.5 Simulation results

This section reports the results of the performed experiments.

Before showing the actual data, it is worth to have a look at the metrics resulting

from the profiling methodology. Figure 6.4a represents the relative amount of patterns selected by the algorithm, while Figures 6.4b, 6.4c, 6.4d represent the average values of - respectively - the compaction, intra-interval reuse and inter-interval reuse factors. It is possible to see that there is a high variability across the selected application set.

Figure 6.5a depicts the overall relative dynamic energy consumption of the hybrid cache-SPM system with respect to the cache-only one. All the benchmarks appear to benefit from the addition of the scratchpad and custom mapping policy, with some differences. Several applications exhibit substantial energy reductions, such as *2mm* (53%), *3mm* (51%), *correlation* (56%), covariance (56%), *gramschmidt* (52%), and *jacobi-1d* (54%). In contrast, *deriche* and *syr2k* experience a comparatively lower impact, with a reduction of 11% and 20%, respectively. These are also the benchmarks with the least amount of selected patterns, as per Figure 6.4a. The geometric mean of the dynamic energy savings is 43%.

Finally, the relative execution time is presented in Figure 6.5b. Here it is possible to see that the effect on performance is mixed, with benchmarks that perform much better than the baseline (like *correlation* and *covariance*, both with a 65% speedup) and others that perform worse (most notably *durbin* with a 40% slowdown, but also *jacobi-1d* and *syr2k*). The overall effect is still positive, with a mean speedup of 13%.

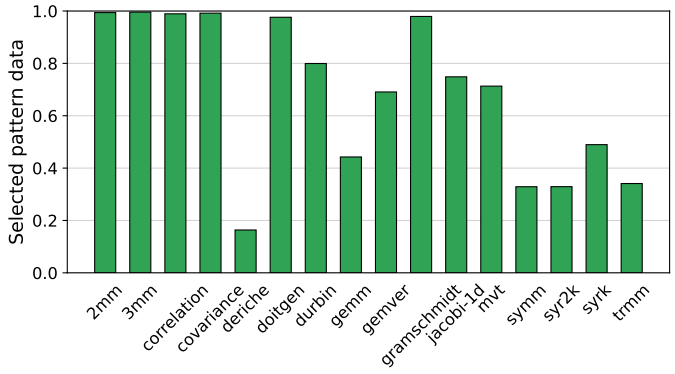
To try to explain the differences in performance between the benchmarks, it is possible to refer to the analysis metrics in Figure 6.4. Kernels like *durbin* and *jacobi-1d* show a very low compaction ratio, hence their spatial locality can be well exploited by the cache. Knowing from simulation statistics that their cache miss rate, which is low in the cache-only system, increases in the hybrid configuration, it is possible to conclude that the cache size reduction and additional SPM data exchange are responsible for their performance deterioration. In similar cases where there is no potential compaction and the miss rate is low, the performance can only get worse, assuming that the access latency is the same for the cache and SPM. It is also worth mentioning that the automatic pattern analysis is not optimal for these two benchmarks, which possess composite patterns that can't always be grouped, contributing to

the additional traffic. This issue can surely be addressed in future work. Other benchmarks like *symm*, *syr2k*, *syrk* and *trmm* show an almost non-existent inter-interval reuse, which again keeps the number of SPM data transfers high and can be the main cause of the slowdown. On the other end, *correlation*, *covariance* and *mvt* exhibit a much better execution time compared to the baseline. These benchmark have a high amount of mapped intervals and a generally balanced combination of metrics.

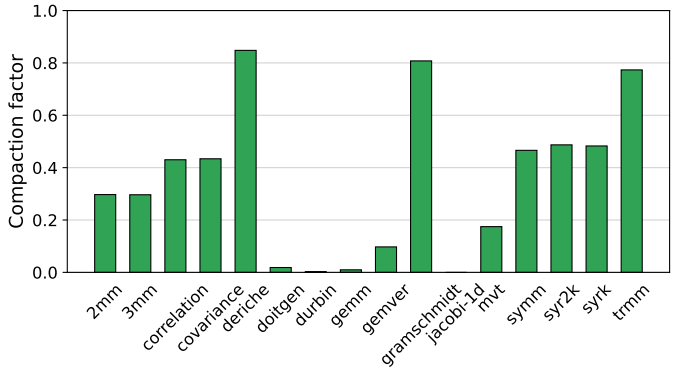
6.6 Chapter conclusions

This chapter illustrated the details of a custom cache-SPM architecture with dynamic data layout compaction. The architecture has been developed - once again - with the goal of reducing the energy consumption of embedded processors, exploiting the better characteristics of scratchpad memories and removing problematic patterns from the cache. Contrarily to the work presented in Chapter 4, this one is focused on hybrid control mechanisms rather then technology, and requires a smart data allocation policy to make use of the software-controlled SPM. The experimental work has been conducted with the help of the profiling methodology described in Chapter 5, on a set of loop-dominated mathematical kernels from the PolyBench/C suite.

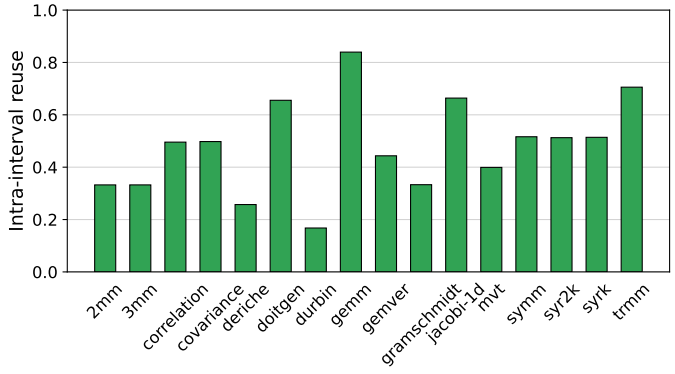
The simulations, targeting a system with a single level of cache and one added SPM, have shown that the energy reduction is not only always present, but also substantial in most of the benchmarks. The relative execution time, instead, is sometimes (but not always) higher than the baseline. This depends on the single application properties, some of which emerged during the analysis phase. Additionally, the target system may not behave optimally in cases where the spatial locality is already good, due to the additional data exchange with the main memory, which can be addressed in future work with a different memory organization or other architectural solutions. A refinement of the pattern identification and mapping algorithms can also help getting better results.



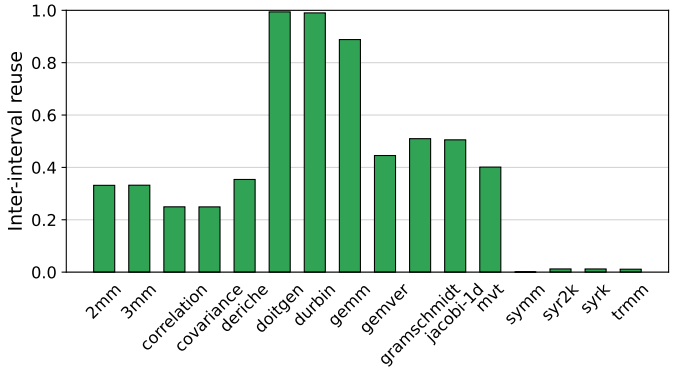
(a) Percentage of selected pattern data



(b) Cache line compaction factor

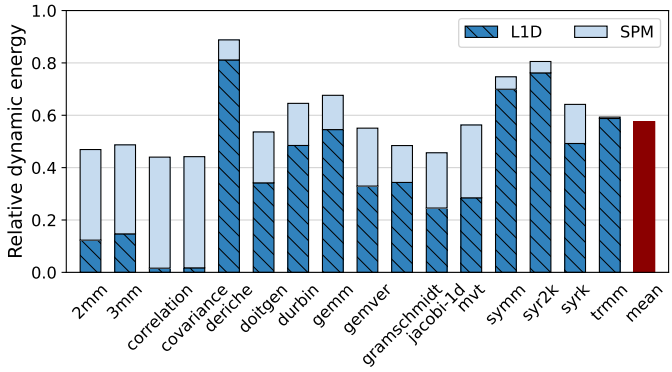


(c) Intra-interval reuse factor

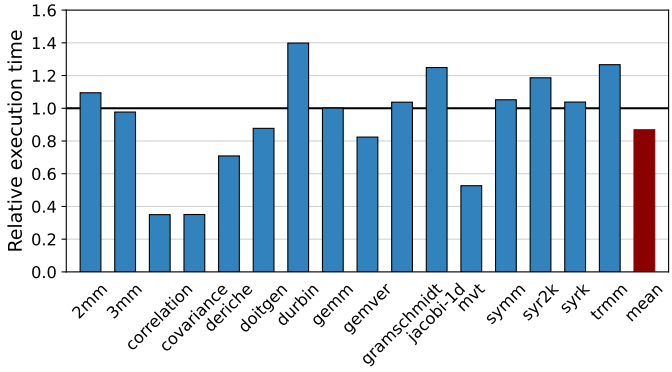


(d) Inter-interval reuse factor

Figure 6.4: Analysis metrics [63]



(a) Relative dynamic energy of hybrid system



(b) Relative execution time of hybrid system

Figure 6.5: Simulation results [63]

Chapter 7

Conclusion

7.1 Summary and final remarks

The thesis had the ambitious goal of finding effective techniques to reduce the power consumption of embedded processors, through a series of modifications to the CPU memory subsystem. It is well-known that this on-chip memory organization is the major contribution to the overall SoC energy consumption [105]. The problem has been faced from different perspectives. The point of connection between the approaches has been the exploration of heterogeneous or hybrid solutions, mixing conventional elements in general-purpose computers with other elements, either innovative or reused from other contexts (but with a clear novel extension).

The first experimental contribution of the thesis has been a comprehensive analysis of mixed-technology caches, where the last-level cache is based on STT-MRAM. That MRAM technology is the most suited for large on-chip caches, because of its combination of high density and high R/W access speed, compared to other emerging technologies like PCM, FeRAM, ReRAM. Specifically, the focus has been on studying the feasibility of a drop-in LLC technology replacement from a traditional SRAM-based solution, analyzing

the benefit it has in terms of energy and the potential negative impact on performance. Furthermore, a tuning of some microarchitectural parameters has been explored with the goal of mitigating the remaining (small) slowdown due to the inevitable higher write access delay. This detailed design-space exploration is novel compared to the state-of-the-art at the time of our research. Among the explored options, the design choice that has shown to have the most impact on the relative performance is the number of LLC cache banks. The study has been correlated with an analysis of the slowdown origin linked to the number of bank conflicts, and a detailed energy consumption breakdown based on two energy models: one in-house calibrated one and one publicly available. The main findings and messages of the study are:

- Using STT-MRAM instead of SRAM in this context brings a substantial energy reduction (60% to 76% of the LLC energy for the selected benchmark set and platforms). The relative slowdown due to the increased latency is usually tolerable (under 6% in the typical case).
- The number of banks can be increased for higher parallelism, hiding the additional latencies related to conflicts. The optimal number of banks for an STT-MRAM LLC, in terms of energy consumption, is 16, which is higher than the corresponding number for SRAM (4).
- The energy results obtained using NVSim do not match the numbers extracted with the calibrated in-house model in any of the tested configuration. This raises questions about the accuracy of public models in relation to newer technology nodes and fabrication processes. Unfortunately, the companies who fabricate these highly scaled memories do not disclose any detailed metrics to enable a calibration of these public-domain models. Extrapolating trends from older, less scaled tech nodes is known to be invalid, now that Dennard's linear scaling law is broken both for devices and wires.

All the contributions of this work have been published in a specialized journal [64].

The second experimental contribution of this thesis has consisted in a set of architectural modifications defining a custom cache-SPM architecture, and a methodology to profile applications to generate optimal mapping policies. This is based on the fact that accessing scratchpad memories has a lower energy cost compared to caches, and that certain workloads with strided accesses waste cache space and bandwidth by fetching elements that are not used in the short term. In this architecture, data can be dynamically stored in the scratchpad with a different layout compared to the main memory, allowing compaction of non-contiguous elements. One of the major elements of distinction with the state-of-the-art in this domain is that the entire workflow has been developed with the assumption that accessing applications source code is not possible. This requirement is crucial for industrial adoption, where source code is frequently unavailable for recompilation or easy design-time analysis. The profiling has targeted a series of representative mathematical computation benchmarks with a good amount of regular accesses to memory (*loop-dominated* kernels). The reference platform for the experiments is a basic embedded processor system with a single level of SRAM cache, to which an SRAM scratchpad memory and architectural support is added, without altering the total memory size. The major findings and messages of this analysis are:

- Placing in the scratchpad memory the patterns with high utilization or potential compaction have a beneficial effect on the cache energy (45% less on average for the tested applications). Performance is improved or worsened, depending on the specific characteristics of the application, but in general it remains quite close to the original.
- In benchmarks with good spatial locality, where the cache works well enough, the additional data transfer due to the direct exchange between the main memory and SPM can limit the effectiveness of the proposal, especially when the cache size is reduced to make space for the scratchpad.

Also in this case, the work has been described in detail in a recent publication [63].

The theme of energy reduction in modern systems is of critical importance. The obtained results, even if not groundbreaking, demonstrate that there is still

space for efficiency improvement in computer architectures design, especially if going for domain-specific solutions. Above all, the insights from the conducted work indicate that it is not necessary to radically modify existing systems, analyze/recompile applications source code, or to complicate the user experience to improve the execution metrics. The proposed system changes are either transparent to the final user, or make use of an automatic workflow that does not require manual intervention. All these elements are crucial for easy industrial adoption, and they are complementary and innovative compared to the state-of-the-art in the academic community.

7.2 Future work

Looking at the contributions presented in this thesis, a range of potential directions emerges for advancing beyond the established foundations.

Starting from the hybrid cache technology work (Chapter 4), several other possibilities can be explored to mitigate the additional write latency of the STT-MRAM or reduce the number of LLC writes. For example, a write-around policy could be applied to the cache in phases where such writes are more impactful. Similarly, assuming a mostly inclusive policy as in the experiments, some of the LLC fills can be dynamically skipped when penalizing write conflicts would arise because of them. An exploration of other relevant microarchitectural parameters not strictly related to the cache, such as the CPU store buffer size, can be helpful too. The use of different technologies does not necessarily have to be applied to different cache levels: a single cache with banks of multiple technologies, with online data placement strategies to minimize the performance impact, can be investigated as well, similarly to other related work [19, 96]. Finally, the work can be enriched with additional analyses involving newer memory models, like for example SOT-MRAM or VCMA-MRAM. These are especially relevant for smaller caches where density is not the most important metric.

In the proposed heterogeneous cache-SPM system (Chapter 6), solutions to reduce the data transfer bottleneck between the scratchpad and main memory

can be introduced. This could be achieved, for example, employing a second-level scratchpad to act as a filter. This approach can also be combined with a feasibility study of emerging technologies in the scratchpad hierarchy, as a link between the two works. Another solution could involve the introduction of shortcuts for direct cache and SPM communication, although it requires dealing with the change of layout. With reference to the SPM support additions and how they are implemented in the simulation environment, it is certainly possible to improve the functional model. For example, the implementation of a proper mechanism to exploit data inter-interval reuse is needed, dynamically remapping SPM addresses at intervals boundaries. Since the first experiments are based on simulation of modified access traces, more work can also be spent on the enforcement of the SPM placement policies in real scenarios, developing an automated binary rewriting toolchain based on profiling data. The COMPAD architectural modifications could also be implemented and tested on real hardware, like a modified open RISC-V core on FPGA, and a more thorough PPAC analysis based on actual technology data could be performed.

Focusing on the application profiling part (Chapter 5), the patterns detection heuristics and mapping decision algorithm can be reworked to improve the general performance-efficiency trade-off. The benefits of the methodology could also be tested on a wider variety of benchmarks, adapting the policies to specific contexts or classes of behaviour. Since the developed methodology already allows to reconstruct loop nests in simple cases, another potential direction can be to further improve this mechanism: obtaining a general view on the data access behaviour of target applications would make it easier to replace the heuristics with optimal placement policies using mathematical methods. Finally, it would be interesting to explore a hardware implementation of the pattern detector, to enable a fully automatic placement mechanism without the need of any software modification or offline profiling. This kind of solution would also have the advantage to work well with more dynamic applications, whose control flow is more dependent on input data. If going in this direction, an accurate estimation of the additional hardware overhead and the performance cost associated to the online pattern identification would need to be performed.

As a final remark, both presented works can be extended considering multi-core

systems with parallel or mixed workloads. This would increase the realism of the context, creating additional challenges but also a higher research value.

Bibliography

[1] A. Agarwal and S. Pudar. "Column-associative Caches: A Technique For Reducing The Miss Rate Of Direct-mapped Caches". In: *Proceedings of the 20th Annual International Symposium on Computer Architecture*. 1993, pp. 179–190. DOI: 10.1109/ISCA.1993.698559.

[2] A. Agarwal, J. Hennessy, and M. Horowitz. "Cache Performance of Operating System and Multiprogramming Workloads". In: *ACM Trans. Comput. Syst.* 6.4 (Nov. 1988), pp. 393–431. ISSN: 0734-2071. DOI: 10.1145/48012.48037. URL: https://doi.org/10.1145/48012.48037.

[3] B. Agarwalla, S. Das, and N. Sahu. "Efficient Cache Resizing policy for DRAM-based LLCs in ChipMultiprocessors". In: *Journal of Systems Architecture* 113 (2021), p. 101886. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2020.101886. URL: https://www.sciencedirect.com/science/article/pii/S1383762120301582.

[4] A. Akram and L. Sawalha. "A Survey of Computer Architecture Simulation Techniques and Tools". In: *IEEE Access* 7 (2019), pp. 78120–78145. DOI: 10.1109/ACCESS.2019.2917698.

[5] A. Akram and L. Sawalha. "Validation of the gem5 Simulator for x86 Architectures". In: *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 2019, pp. 53–58. DOI: 10.1109/PMBS49563.2019.00012.

[6] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill-Smith, and M. Krounbi. "Spin-Transfer Torque Magnetic Random Access Memory

(STT-MRAM)". In: *J. Emerg. Technol. Comput. Syst.* 9.2 (May 2013). ISSN: 1550-4832. DOI: 10.1145/2463585.2463589. URL: https://doi.org/10.1145/2463585.2463589.

[7] Arm Limited. *The Arm Research Starter Kit: System Modeling using gem5*. https://github.com/arm-university/arm-gem5-rsk/blob/master/gem5_rsk_gem5-23.0.pdf. 2023.

[8] R. M. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. *Comparison of Cache-and Scratch-Pad based Memory Systems with respect to Performance, Area and Energy Consumption*. Technical Report #762. Universität Dortmund, 2001. URL: https://api.semanticscholar.org/CorpusID:2889002.

[9] L. A. Bathen and N. Dutt. "HaVOC: A Hybrid Memory-Aware Virtualization Layer for on-Chip Distributed ScratchPad and Non-Volatile Memories". In: *Proceedings of the 49th Annual Design Automation Conference*. DAC '12. San Francisco, California: Association for Computing Machinery, 2012, pp. 447–452. ISBN: 9781450311991. DOI: 10.1145/2228360.2228438. URL: https://doi.org/10.1145/2228360.2228438.

[10] 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". In: *SIGARCH Comput. Archit. News* 39.2 (Aug. 2011), pp. 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718. URL: https://doi.org/10.1145/2024716.2024718.

[11] F. Bodin and A. Seznec. "Skewed Associativity Enhances Performance Predictability". In: *SIGARCH Comput. Archit. News* 23.2 (May 1995), pp. 265–274. ISSN: 0163-5964. DOI: 10.1145/225830.224437. URL: https://doi.org/10.1145/225830.224437.

[12] J. Bucek, K.-D. Lange, and J. v. Kistowski. "SPEC CPU2017: Next-Generation Compute Benchmark". In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE '18. Berlin, Germany: Association for Computing Machinery, 2018,

pp. 41–42. ISBN: 9781450356299. DOI: 10.1145/3185768.3185771. URL: https://doi.org/10.1145/3185768.3185771.

[13] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli. "Accuracy evaluation of GEM5 simulator system". In: *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. 2012, pp. 1–7. DOI: 10.1109/ReCoSoC.2012.6322869.

[14] T. E. Carlson, W. Heirman, and L. Eeckhout. "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation". In: *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 2011, pp. 1–12. DOI: 10.1145/2063384.2063454.

[15] P. Chakraborty and P. R. Panda. "SPM-Sieve: A framework for assisting data partitioning in scratch pad memory based systems". In: *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. 2013, pp. 1–10. DOI: 10.1109/CASES.2013.6662527.

[16] P. Chakraborty, P. R. Panda, and S. Sen. "Partitioning and Data Mapping in Reconfigurable Cache and Scratchpad Memory–Based Architectures". In: *ACM Trans. Des. Autom. Electron. Syst.* 22.1 (Sept. 2016). ISSN: 1084-4309. DOI: 10.1145/2934680. URL: https://doi.org/10.1145/2934680.

[17] M.-t. Chang, P. Rosenfeld, S.-l. Lu, and B. Jacob. "Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM". In: *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 143–154. ISBN: 9781467355872. DOI: 10.1109/HPCA.2013.6522314.

[18] T.-F. Chen and J.-L. Baer. "Reducing memory latency via non-blocking and prefetching caches". In: *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS V. Boston, Massachusetts, USA: Association for Computing Machinery, 1992, pp. 51–61. ISBN: 0897915348. DOI: 10.1145/143365.143486. URL: https://doi.org/10.1145/143365.143486.

[19] H.-Y. Cheng, J. Zhao, J. Sampson, M. J. Irwin, A. Jaleel, Y. Lu, and Y. Xie. "LAP: Loop-Block Aware Inclusion Properties for Energy-Efficient Asymmetric Last Level Caches". In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 2016, pp. 103–114. DOI: 10.1109/ISCA.2016.19.

[20] H. Cho, B. Egger, J. Lee, and H. Shin. "Dynamic Data Scratchpad Memory Management for a Memory Subsystem with an MMU". In: *Proceedings of the 2007 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES '07. San Diego, California, USA: Association for Computing Machinery, 2007, pp. 195–206. ISBN: 9781595936325. DOI: 10.1145/1254766.1254804. URL: https://doi.org/10.1145/1254766.1254804.

[21] S. Couet and G. Sankar Kar. *MRAM technologies: from space applications to unified cache memory?* https://www.imec-int.com/en/articles/mram-technologies-space-applications-unified-cache-memory. 2022.

[22] S. Couet and G. Sankar Kar. *Novel SOT-MRAM architecture opens doors to high-density last-level cache memory applications*. https://www.imec-int.com/en/articles/novel-sot-mram-architecture-opens-doors-high-density-last-level-cache-memory-applications. 2023.

[23] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc. "Design of ion-implanted MOSFET's with very small physical dimensions". In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268. DOI: 10.1109/JSSC.1974.1050511.

[24] A. Dominguez, S. Udayakumaran, and R. Barua. "Heap Data Allocation to Scratch-Pad Memory in Embedded Systems". In: *J. Embedded Comput.* 1.4 (Dec. 2005), pp. 521–540. ISSN: 1740-4460.

[25] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.7 (2012), pp. 994–1007. DOI: 10.1109/TCAD.2012.2185930.

[26] B. Egger, S. Kim, C. Jang, J. Lee, S. L. Min, and H. Shin. "Scratchpad Memory Management Techniques for Code in Embedded Systems without an MMU". In: *IEEE Transactions on Computers* 59.8 (2010), pp. 1047–1062. DOI: 10.1109/TC.2009.188.

[27] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. "Dark silicon and the end of multicore scaling". In: *2011 38th Annual International Symposium on Computer Architecture (ISCA)*. 2011, pp. 365–376.

[28] T. Evenblij, M. Perumkunnil, F. Catthoor, S. Sakhare, P. Debacker, G. Kar, A. Furnemont, N. Bueno, J. I. Gómez-Pérez, and C. Tenllado. "A Comparative Analysis on the Impact of Bank Contention in STT-MRAM and SRAM Based LLCs". In: *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, Nov. 2019, pp. 255–263. DOI: 10.1109/ICCD46524.2019.00039.

[29] T. Evenblij, M. P. Komalan, G. Sankar Kar, and A. Furnemont. *Getting the most out of your system*. https://www.imec-int.com/en/articles/getting-most-out-your-system. 2021.

[30] K. I. Farkas and N. P. Jouppi. "Complexity/performance tradeoffs with non-blocking loads". In: *Proceedings of the 21st Annual International Symposium on Computer Architecture*. ISCA '94. Chicago, Illinois, USA: IEEE Computer Society Press, 1994, pp. 211–222. ISBN: 0818655100. DOI: 10.1145/191995.192029. URL: https://doi.org/10.1145/191995.192029.

[31] D. Genbrugge, S. Eyerman, and L. Eeckhout. "Interval simulation: Raising the level of abstraction in architectural simulation". In: *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. 2010, pp. 1–12. DOI: 10.1109/HPCA.2010.5416636.

[32] P. Greenhalgh. *Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7*. https://www.eetimes.com/big-little-processing-with-arm-cortex-a15-cortex-a7/. 2011.

[33] W. Gul, M. Shams, and D. Al-Khalili. “SRAM Cell Design Challenges in Modern Deep Sub-Micron Technologies: An Overview”. In: *Micromachines* 13.8 (2022). ISSN: 2072-666X. DOI: 10.3390/mi13081332. URL: https://www.mdpi.com/2072-666X/13/8/1332.

[34] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi, C. D. Emmons, M. Hayenga, and N. Paver. “Sources of error in full-system simulation”. In: *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2014, pp. 13–22. DOI: 10.1109/ISPASS.2014.6844457.

[35] F. Hameed, A. A. Khan, and J. Castrillon. “Performance and Energy-Efficient Design of STT-RAM Last-Level Cache”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.6 (2018), pp. 1059–1072. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2018.2804938. URL: https://ieeexplore.ieee.org/document/8307465/.

[36] F. Hameed and M. B. Tahoori. “Architecting STT Last-Level-Cache for performance and energy improvement”. In: *2016 17th International Symposium on Quality Electronic Design (ISQED)*. Vol. 2016-May. IEEE, 2016, pp. 319–324. ISBN: 9781509012138. DOI: 10.1109/ISQED.2016.7479221.

[37] J. Hu, Q. Zhuge, C. J. Xue, W.-C. Tseng, and E. H.-M. Sha. “Management and Optimization for Nonvolatile Memory-Based Hybrid Scratchpad Memory on Multicore Embedded Processors”. In: *ACM Trans. Embed. Comput. Syst.* 13.4 (Mar. 2014). ISSN: 1539-9087. DOI: 10.1145/2560019. URL: https://doi.org/10.1145/2560019.

[38] C. D. S. Inc. *Virtuoso*. https://www.cadence.com.

[39] Intel Corporation. *Scatter-Gather DMA Controller Core*. https://www.intel.com/content/www/us/en/docs/programmable/683130/21-4/scatter-gather-dma-controller-core.html. 2018.

[40] Intel Corporation. *How 12th Gen’s Intel® Core™ Hybrid Technology Works*. https://www.intel.com/content/www/us/en/support/articles/000091896/processors.html. 2022.

[41] Intel Corporation. *Pin 3.27*. https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html. 2023.

[42] A. Janapsatya, S. Parameswaran, and A. Ignjatovic. "Hardware/software managed scratchpad memory for embedded system". In: *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.* 2004, pp. 370–377. DOI: 10.1109/ICCAD.2004.1382603.

[43] P. Jangra and M. Duhan. "Performance-based comparative study of existing and emerging non-volatile memories: a review". In: *Journal of Optics* 52 (2022), pp. 2395–2409. URL: https://link.springer.com/article/10.1007/s12596-022-01058-w.

[44] X. Ji, C. Wang, N. El-Sayed, X. Ma, Y. Kim, S. S. Vazhkudai, W. Xue, and D. Sánchez. "Understanding object-level memory access patterns across the spectrum". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017).

[45] A. Kanduri, A. M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen. "A Perspective on Dark Silicon". In: *The Dark Side of Silicon: Energy Efficient Computing in the Dark Silicon Era*. Ed. by A. M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen. Cham: Springer International Publishing, 2017, pp. 3–20. ISBN: 978-3-319-31596-6. DOI: 10.1007/978-3-319-31596-6_1. URL: https://doi.org/10.1007/978-3-319-31596-6_1.

[46] R. Kessler. "The Alpha 21264 microprocessor". In: *IEEE Micro* 19.2 (1999), pp. 24–36. DOI: 10.1109/40.755465.

[47] N. Khoshavi and R. F. Demara. "Read-Tuned STT-RAM and eDRAM Cache Hierarchies for Throughput and Energy Optimization". In: *IEEE Access* 6 (2018), pp. 14576–14590. DOI: 10.1109/ACCESS.2018.2813668.

[48] Y. Kim, W. Yang, and O. Mutlu. "Ramulator: A Fast and Extensible DRAM Simulator". In: *IEEE Computer Architecture Letters* 15.1 (2016), pp. 45–49. DOI: 10.1109/LCA.2015.2414456.

[49] M. Kimball. *AMD Delivers 128 Cores In Bergamo*. https://www.forbes.com/sites/moorinsights/2023/06/19/amd-delivers-128-cores-in-bergamo/. 2023.

[50] M. Komalan, S. Sakhare, T. H. Bao, S. Rao, W. Kim, C. Tenllado, J. I. Gómez, G. S. Kar, A. Furnemont, and F. Catthoor. "Cross-layer design and analysis of a low power, high density STT-MRAM for embedded systems". In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4. ISBN: 9781467368520. DOI: 10.1109/ISCAS.2017.8050923.

[51] M. P. Komalan, C. Tenllado, J. I. Gomez Perez, F. T. Fernández, and F. Catthoor. "System level exploration of a STT-MRAM based level 1 data-cache". In: *2015 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Vol. 2015-April. IEEE, 2015, pp. 1311–1316. ISBN: 9783981537055.

[52] R. Komuravelli, M. D. Sinclair, J. Alsop, M. Huzaifa, M. Kotsifakou, P. Srivastava, S. V. Adve, and V. S. Adve. "Stash: Have Your Scratchpad and Cache It Too". In: *SIGARCH Comput. Archit. News* 43.3S (June 2015), pp. 707–719. ISSN: 0163-5964. DOI: 10.1145/2872887.2750374. URL: https://doi.org/10.1145/2872887.2750374.

[53] D. Kroft. "Lockup-free instruction fetch/prefetch cache organization". In: *Proceedings of the 8th Annual Symposium on Computer Architecture*. ISCA '81. Minneapolis, Minnesota, USA: IEEE Computer Society Press, 1981, pp. 81–87.

[54] K. Kuan and T. Adegbija. "HALLS: An Energy-Efficient Highly Adaptable Last Level STT-RAM Cache for Multicore Systems". In: *IEEE Transactions on Computers* 68.11 (2019), pp. 1623–1634. ISSN: 0018-9340. DOI: 10.1109/TC.2019.2918153.

[55] K. Kuhn. "Chapter 1 - CMOS and Beyond CMOS: Scaling Challenges". In: *High Mobility Materials for CMOS Applications*. Ed. by N. Collaert. Woodhead Publishing Series in Electronic and Optical Materials. Woodhead Publishing, 2018, pp. 1–44. ISBN: 978-0-08-102061-6. DOI: https://doi.org/10.1016/B978-0-08-102061-6.00001-X. URL:

https://www.sciencedirect.com/science/article/pii/B9780081020616000001X.

[56] S. Kumar and P. K. Singh. "An overview of modern cache memory and performance analysis of replacement policies". In: *2016 IEEE International Conference on Engineering and Technology (ICETECH)*. 2016, pp. 210–214. DOI: 10.1109/ICETECH.2016.7569243.

[57] Lattice Semiconductor. *Scatter-Gather Direct Memory Access Controller IP Core User Guide*. https://www.latticesemi.com/-/media/LatticeSemi/Documents/UserManuals/RZ/ScatterGatherDirectMemoryAccessControllerIPCoreUsersGuide.ashx?document_id=24824. 2015.

[58] C.-C. Lee, I.-C. Chen, and T. Mudge. "The bi-mode branch predictor". In: *Proceedings of 30th Annual International Symposium on Microarchitecture*. 1997, pp. 4–13. DOI: 10.1109/MICRO.1997.645792.

[59] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob. "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator". In: *IEEE Computer Architecture Letters* 19.2 (2020), pp. 106–109. DOI: 10.1109/LCA.2020.2973991.

[60] L. Liebmann, J. Hibbeler, N. Hieter, L. Pileggi, T. Jhaveri, M. Moe, and V. Rovner. "Demonstrating the benefits of template-based design-technology co-optimization". In: *Design for Manufacturability through Design-Process Integration IV*. Ed. by M. L. Rieger and J. Thiele. Vol. 7641. International Society for Optics and Photonics. SPIE, 2010, 76410R. DOI: 10.1117/12.848244. URL: https://doi.org/10.1117/12.848244.

[61] N. Loubet, T. Hook, P. Montanini, C.-W. Yeung, S. Kanakasabapathy, M. Guillom, T. Yamashita, J. Zhang, X. Miao, J. Wang, A. Young, R. Chao, M. Kang, Z. Liu, S. Fan, B. Hamieh, S. Sieg, Y. Mignot, W. Xu, S.-C. Seo, J. Yoo, S. Mochizuki, M. Sankarapandian, O. Kwon, A. Carr, A. Greene, Y. Park, J. Frougier, R. Galatage, R. Bao, J. Shearer, R. Conti, H. Song, D. Lee, D. Kong, Y. Xu, A. Arceo, Z. Bi, P. Xu, R. Muthinti, J. Li, R. Wong, D. Brown, P. Oldiges, R. Robison, J. Arnold, N. Felix, S. Skordas, J. Gaudiello, T. Standaert, H. Jagannathan, D. Corliss, M.-H. Na, A. Knorr, T. Wu, D. Gupta, S. Lian, R. Divakaruni,

T. Gow, C. Labelle, S. Lee, V. Paruchuri, H. Bu, and M. Khare. “Stacked nanosheet gate-all-around transistor to enable scaling beyond FinFET”. In: *2017 Symposium on VLSI Technology*. 2017, T230–T231. DOI: 10.23919/VLSIT.2017.7998183.

[62] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillón, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, M. Fariborz, A. F. Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. S. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and É. F. Zulian. “The gem5 Simulator: Version 20.0+”. In: *CoRR* abs/2007.03152 (2020). arXiv: 2007.03152. URL: https://arxiv.org/abs/2007.03152.

[63] T. Marinelli, J. I. Gómez Pérez, C. Tenllado, and F. Catthoor. “COMPAD: A heterogeneous cache-scratchpad CPU architecture with data layout compaction for embedded loop-dominated applications”. In: *Journal of Systems Architecture* 145 (2023), p. 103022. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2023.103022. URL: https://www.sciencedirect.com/science/article/pii/S1383762123002011.

[64] T. Marinelli, J. I. G. Pérez, C. Tenllado, M. Komalan, M. Gupta, and F. Catthoor. “Microarchitectural Exploration of STT-MRAM Last-Level Cache Parameters for Energy-Efficient Devices”. In: *ACM Trans. Embed. Comput. Syst.* 21.1 (Jan. 2022). ISSN: 1539-9087. DOI: 10.1145/3490391. URL: https://doi.org/10.1145/3490391.

[65] R. McIlroy, P. Dickman, and J. Sventek. “Efficient Dynamic Heap Allocation of Scratch-Pad Memory”. In: *Proceedings of the 7th Inter-*

national Symposium on Memory Management. ISMM '08. Tucson, AZ, USA: Association for Computing Machinery, 2008, pp. 31–40. ISBN: 9781605581347. DOI: 10.1145/1375634.1375640. URL: https://doi.org/10.1145/1375634.1375640.

[66] G. Moore. *Excerpts from a Conversation with Gordon Moore: Moore's Law*. http://large.stanford.edu/courses/2012/ph250/lee1/docs/Excepts_A_Conversation_with_Gordon_Moore.pdf. 2016.

[67] S. M. Nair, M. Mayahinia, M. B. Tahoori, M. Perumkunnil, H. Zahedmanesh, K. Croes, K. Garello, T. Marinelli, T. Evenblij, G. S. Kar, and F. Catthoor. "Workload-Aware Electromigration Analysis in Emerging Spintronic Memory Arrays". In: *IEEE Transactions on Device and Materials Reliability* 21.2 (2021), pp. 258–266. DOI: 10.1109/TDMR.2021.3074251.

[68] F. Oboril, R. Bishnoi, M. Ebrahimi, and M. B. Tahoori. "Evaluation of Hybrid Memory Technologies Using SOT-MRAM for On-Chip Cache Hierarchy". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.3 (2015), pp. 367–380. DOI: 10.1109/TCAD.2015.2391254.

[69] Ohio State University. *PolyBench/C 4.2.1*. https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/. 2016.

[70] E. Park and T. Song. "Complementary FET (CFET) Standard Cell Design for Low Parasitics and Its Impact on VLSI Prediction at 3-nm Process". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31.2 (2023), pp. 177–187. DOI: 10.1109/TVLSI.2022.3220339.

[71] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. "DRAMSim2: A Cycle Accurate Memory System Simulator". In: *IEEE Computer Architecture Letters* 10.1 (2011), pp. 16–19. DOI: 10.1109/L-CA.2011.4.

[72] K. Rupp. *Microprocessor Trend Data*. https://github.com/karlrupp/microprocessor-trend-data. 2022.

[73] J. Ryckaert. *STCO: unlocking system-scaling bottlenecks*. https://www.imec-int.com/en/articles/unlocking-system-scaling-bottlenecks-system-technology-co-optimization. 2022.

[74] R. Saha, Y. P. Pundir, and P. Kumar Pal. “Comparative analysis of STT and SOT based MRAMs for last level caches”. In: *Journal of Magnetism and Magnetic Materials* 551 (2022), p. 169161. ISSN: 0304-8853. DOI: https://doi.org/10.1016/j.jmmm.2022.169161. URL: https://www.sciencedirect.com/science/article/pii/S0304885322001275.

[75] S. Sakhare, M. Perumkunnil, T. H. Bao, S. Rao, W. Kim, D. Crotti, F. Yasin, S. Couet, J. Swerts, S. Kundu, D. Yakimets, R. Baert, H. Oh, A. Spessot, A. Mocuta, G. S. Kar, and A. Furnemont. “Enablement of STT-MRAM as last level cache for the high performance computing domain at the 5nm node”. In: *2018 IEEE International Electron Devices Meeting (IEDM)*. 2018, pp. 18.3.1–18.3.4. DOI: 10.1109/IEDM.2018.8614637.

[76] D. Sanchez and C. Kozyrakis. “ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems”. In: *SIGARCH Comput. Archit. News* 41.3 (June 2013), pp. 475–486. ISSN: 0163-5964. DOI: 10.1145/2508148.2485963. URL: https://doi.org/10.1145/2508148.2485963.

[77] D. Schor. *IEDM 2022: Did We Just Witness The Death Of SRAM?* https://fuse.wikichip.org/news/7343/iedm-2022-did-we-just-witness-the-death-of-sram/. 2022.

[78] A. Seznec. “A Case for Two-Way Skewed-Associative Caches”. In: *SIGARCH Comput. Archit. News* 21.2 (May 1993), pp. 169–178. ISSN: 0163-5964. DOI: 10.1145/173682.165152. URL: https://doi.org/10.1145/173682.165152.

[79] A. Seznec and F. Bodin. “Skewed-associative Caches.” In: June 1993, pp. 304–316. ISBN: 978-3-540-56891-9. DOI: 10.1007/3-540-56891-3_24.

[80] M. Shekarisaz, A. Hoseinghorban, M. Bazzaz, M. Salehi, and A. Ejlali. “MASTER: Reclamation of Hybrid Scratchpad Memory to Maximize Energy Saving in Multi-core Edge Systems”. In: *IEEE Transactions on Sustainable Computing* (2021), pp. 1–1. DOI: 10.1109/TSUSC.2021.3049447.

[81] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. "Automatically Characterizing Large Scale Program Behavior". In: *SIGPLAN Not.* 37.10 (Oct. 2002), pp. 45–57. ISSN: 0362-1340. DOI: 10.1145/605432.605403. URL: https://doi.org/10.1145/605432.605403.

[82] S. Singh and M. Awasthi. "Memory Centric Characterization and Analysis of SPEC CPU2017 Suite". In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ICPE '19. Mumbai, India: Association for Computing Machinery, 2019, pp. 285–292. ISBN: 9781450362399. DOI: 10.1145/3297663.3310311. URL: https://doi.org/10.1145/3297663.3310311.

[83] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan. "Relaxing non-volatility for fast and energy-efficient STT-RAM caches". In: *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, Feb. 2011, pp. 50–61. ISBN: 9781424494323. DOI: 10.1109/HPCA.2011.5749716.

[84] S. Song, Q. Wu, S. Flolid, J. Dean, R. Panda, J. Deng, and L. K. John. *Experiments with spec cpu 2017: Similarity, balance, phase behavior and simpoints*. Tech. rep. Laboratory for Computer Architecture, Department of Electrical and Computer Engineering, The University of Texas at Austin, 2018.

[85] Steam. *Hardware & Software Survey: November 2023*. https://store.steampowered.com/hwsurvey/. 2023.

[86] Y. Taur and E. J. Nowak. "CMOS devices below 0.1 /spl mu/m: how high will performance go?" In: *International Electron Devices Meeting. IEDM Technical Digest* (1997), pp. 215–218. URL: https://api.semanticscholar.org/CorpusID:35663536.

[87] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu. "MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices". In: *FAST*. 2018.

[88] The gem5 community. *Minor CPU Model*. https://www.gem5.org/documentation/general_docs/cpu_models/minor_cpu. 2024.

[89] The gem5 community. *Out of Order CPU Model*. https://www.gem5.org/documentation/general_docs/cpu_models/O3CPU. 2024.

[90] TOP500. *Frontier - HPE Cray EX235a*. https://www.top500.org/system/180047/. 2021.

[91] S. Udayakumaran and R. Barua. "Compiler-Decided Dynamic Memory Allocation for Scratch-Pad Based Embedded Systems". In: *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. CASES '03. San Jose, California, USA: Association for Computing Machinery, 2003, pp. 276–286. ISBN: 1581136765. DOI: 10.1145/951710.951747. URL: https://doi.org/10.1145/951710.951747.

[92] S. Van Beek, K. Martens, P. Roussel, Y. Wu, W. Kim, S. Rao, J. Swerts, D. Crotti, D. Linten, G. Kar, and G. Groeseneken. "Thermal stability analysis and modelling of advanced perpendicular magnetic tunnel junctions". In: *AIP Advances* 8.5 (2018), p. 055909. DOI: 10.1063/1.5007690.

[93] M. Verma, L. Wehmeyer, and P. Marweclel. "Dynamic overlay of scratch-pad memory for energy minimization". In: *International Conference on Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004.* 2004, pp. 104–109.

[94] J. Von Neumann. *First Draft of a Report on the EDVAC*. https://web.mit.edu/STS.035/www/PDFs/edvac.pdf. 1945.

[95] J. Wang, X. Dong, and Y. Xie. "OAP: An obstruction-aware cache management policy for STT-RAM last-level caches". In: *2013 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, Mar. 2013, pp. 847–852. DOI: 10.7873/DATE.2013.179.

[96] Z. Wang, D. A. Jiménez, C. Xu, G. Sun, and Y. Xie. "Adaptive placement and migration policy for an STT-RAM-based hybrid cache". In: *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2014, pp. 13–24. ISBN: 9781479930975. DOI: 10.1109/HPCA.2014.6835933.

[97] WikiChip. *Static Random-Access Memory (SRAM)*. https://en.wikichip.org/wiki/static_random-access_memory. 2019.

[98] S. Wilton and N. Jouppi. "CACTI: an enhanced cache access and cycle time model". In: *IEEE Journal of Solid-State Circuits* 31.5 (1996), pp. 677–688. DOI: 10.1109/4.509850.

[99] M. Wolf. "Chapter 5 - Processors and Systems". In: *The Physics of Computing*. Ed. by M. Wolf. Boston: Morgan Kaufmann, 2017, pp. 149–203. ISBN: 978-0-12-809381-8. DOI: https://doi.org/10.1016/B978-0-12-809381-8.00005-5. URL: https://www.sciencedirect.com/science/article/pii/B9780128093818000055.

[100] J. Wuu, R. Agarwal, M. Ciraula, C. Dietz, B. Johnson, D. Johnson, R. Schreiber, R. Swaminathan, W. Walker, and S. Naffziger. "3D V-Cache: the Implementation of a Hybrid-Bonded 64MB Stacked Cache for a 7nm x86-64 CPU". In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. 2022, pp. 428–429. DOI: 10.1109/ISSCC42614.2022.9731565.

[101] Xilinx. *Channelized Direct Memory Access and Scatter Gather*. https://docs.xilinx.com/v/u/en-US/chan_dma_sg. 2010.

[102] A. Yousefzadeh, J. Stuijt, M. Hijdra, H.-H. Liu, A. Gebregiorgis, A. Singh, S. Hamdioui, and F. Catthoor. "Energy-Efficient In-Memory Address Calculation". In: *ACM Trans. Archit. Code Optim.* (May 2022). Just Accepted. ISSN: 1544-3566. DOI: 10.1145/3546071. URL: https://doi.org/10.1145/3546071.

[103] J. Yu, A. Dutta, T. Jaeger, D. Kohlbrenner, and C. W. Fletcher. "Synchronization Storage Channels (S2C): Timer-less Cache Side-Channel Attacks on the Apple M1 via Hardware Synchronization Instructions". In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 1973–1990. ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/yu-jiyong.

[104] S. Yu and P.-Y. Chen. "Emerging Memory Technologies: Recent Trends and Prospects". In: *IEEE Solid-State Circuits Magazine* 8.2 (2016), pp. 43–56. DOI: 10.1109/MSSC.2016.2546199.

[105] W. Zang and A. Gordon-Ross. "A survey on cache tuning from a power/energy perspective". In: *ACM Comput. Surv.* 45.3 (July 2013). ISSN: 0360-0300. DOI: 10.1145/2480741.2480749. URL: https://doi.org/10.1145/2480741.2480749.

[106] C. Zhang, P. Scheffler, T. Benz, M. Perotti, and L. Benini. "AXI-Pack: Near-Memory Bus Packing for Bandwidth-Efficient Irregular Workloads". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137243.

[107] W. Zhang and L. Wu. "Exploiting Hybrid SPM-Cache Architectures to Reduce Energy Consumption for Embedded Computing". In: *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*. 2014, pp. 340–347. DOI: 10.1109/HPCC.2014.59.

[108] V. Zimmer, S. Banik, and R. REGUPATHY. *Early Platform Hardening Technology For Slimmer And Faster Boot*. https://patents.google.com/patent/US20210081538A1/en. 2021.

[109] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic. "Performance Evaluation of Cache Replacement Policies for the SPEC CPU2000 Benchmark Suite". In: *Proceedings of the 42nd Annual Southeast Regional Conference*. ACM-SE 42. Huntsville, Alabama: Association for Computing Machinery, 2004, pp. 267–272. ISBN: 1581138709. DOI: 10.1145/986537.986601. URL: https://doi.org/10.1145/986537.986601.



FACULTAD DE INFORMÁTICA
DEPARTMENT OF COMPUTER ARCHITECTURE AND AUTOMATIC CONTROL
DAYCA-ARTECS
Plaza de las Ciencias 1
28040 Madrid
tommarin@ucm.es



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
ESAT-INSYS
Kasteelpark Arenberg 1 bus 2200
B-3001 Leuven
tommaso.marinelli@kuleuven.be