Exploration and conception of computing architectures of type computing in-memory based on emerging non volatile memories

Valentin Egloff

Aix*Marseille université
Socialement engagée

NNT/NL : 2022AIXM0446/038ED353

THÈSE DE DOCTORAT

Soutenue à Grenoble Minatec
le 08 décembre 2022 par

Valentin EGLOFF

Exploration and conception of computing architectures of type *computing in-memory* based on emerging non volatile memories

Discipline
Sciences pour l'Ingénieur

Spécialité
Micro et Nanoélectronique

École doctorale
ED 353 SCIENCES POUR L'INGENIEUR : MECANIQUE, PHYSIQUE, MICRO ET NANOELECTRONIQUE

Laboratoire/Partenaires de recherche
Commissariat à l'Énergie Atomique et aux Énergies Alternatives

list
cea tech

Composition du jury

Gilles SASSATELLI DR - LIRMM, Université Montpellier	Rapporteur
Lorena ANGHEL PR - Spintec, Grenoble	Rapporteure
Mathieu MOREAU MCF - IM2NP, Aix-Marseille Université	Examinateur
Alberto BOSIO PR - INL, Université Lyon I	Président du jury
Jean-Michel PORTAL PR - IM2NP, Aix-Marseille Université	Directeur de thèse
Jean-Philippe NOEL PhD - CEA, Grenoble	Encadrant CEA

Affidavit

I, undersigned, **Valentin Egloff**, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific direction of **Jean-Michel Portal**, in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with both the french national charter for Research Integrity and the Aix-Marseille University charter on the fight against plagiarism.

This work has not been submitted previously either in this country or in another country in the same or in a similar version to any other examination body.

Grenoble, the 08th september 2022,

List of publications and conference participations

Publications made during thesis

Publications

- **Poster :** “Shuffle operator for matrix multiplication in in-memory computing architecture”, **V. Egloff**, Poster in COMPAS, 2019-06, and also in ACACES HiPEAC, 2019-07
- **2nd author :** J.-P. Noel et al. “Computational SRAM Design Automation using Pushed-Rule Bitcells for Energy-Efficient Vector Processing”. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. ISSN: 1558-1101. Grenoble, France, 2020-03, pp. 1187–1192. DOI: 10.23919/DATE48585.2020.9116506
- **2nd author :** R. Gauchi et al. “Reconfigurable tiles of computing-in-memory SRAM architecture for scalable vectorization”. In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. ISLPED ’20. New York, NY, USA: Association for Computing Machinery, 2020-08, pp. 121–126. ISBN: 978-1-4503-7053-0. DOI: 10.1145/3370748.3406550
- **1st author :** Valentin Egloff et al. “Storage Class Memory with Computing Row Buffer: A Design Space Exploration”. In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2021-02, pp. 1–6. DOI: 10.23919/DATE51398.2021.9473992
- Maha Kooli et al. “Towards a Truly Integrated Vector Processing Unit for Memory-bound Applications Based on a Cost-competitive Computational SRAM Design Solution”. In: *ACM Journal on Emerging Technologies in Computing Systems* 18.2 (2022-04), 40:1–40:26. ISSN: 1550-4832. DOI: 10.1145/3485823. (Visited on 2022-10-04)

Patents

- **Co-inventeur :** Jean-Philippe Noel et al. “Method and device for designing a computational memory circuit”. 2021156420A1. 2021-08
- **Inventeur :** Valentin EGLOFF, Jean-Philippe Noel, and Jean-Michel PORTAL. “Device comprising a non-volatile memory circuit”. 4036916A1. 2022-08

Conference participations

- COMPAS, June 2019 in Biarritz, France
- ACACES HiPEAC summer school, July 2019 in Fiuggi, Italy
- Fête de la science, September 2019 in Grenoble, France
- DATE20, March 2020 in Grenoble, France, physical participation **cancelled** due to COVID
- DATE21, February 2021 in Grenoble, France [online]

Résumé

Les architectures d'aujourd'hui sont basées sur le modèle de von Neumann qui place au centre l'exécution des instructions. Ces architectures font face à de fortes limitations dans le contexte du *big data*. En effet, le mur mémoire est un phénomène lié à l'écart grandissant de performances entre les processeurs et les mémoires depuis les années 80. Pour atténuer cet écart, une hiérarchie de caches a été mise en place mais elle a en contrepartie largement augmentée la consommation énergétique sans être adaptée pour les grands jeux de données modernes. Non seulement ces architectures ont du mal avec une masse de données toujours croissantes à cause de leur haute consommation énergétique et leur faible débit, elles ne peuvent plus uniquement se baser sur les avancées technologiques pour s'améliorer. Ceci appelle à un changement de paradigme vers des architectures *data* centrées où le traitement de quantités de données massives en parallèle est le principe de base.

De nouvelles mémoires non volatiles promettent du stockage haute densité et peuvent intégrer du calcul en mémoire. L'intérêt de calculer en mémoire est d'opérer là où se trouve la donnée, ou tout du moins le plus proche possible, pour supprimer les allées et venues permanentes entre la mémoire et les cœurs de calcul. Les solutions existantes utilisent du calcul analogique très efficace mais prompt au bruit et avec une flexibilité limitée. Quand les données doivent être réécrites en mémoire, l'endurance de ces mémoires non volatiles n'est pas discutée. Nous concevons un emballage numérique qui étend les fonctionnalités mémoire avec du calcul vectoriel et développons une plateforme de simulation pour faire de l'exploration architecturale. Notre circuit, bien nommé C-SRAM, peut être intégré avec la plupart des technologies mémoire et est équipé de sa propre mémoire SRAM. Nous démontrons qu'effectuer le calcul au sommet de la hiérarchie mémoire, c'est à dire proche du stockage permanent, permet une réduction de la consommation énergétique d'un facteur 17.4 et une accélération du traitement en moyenne d'un facteur 12.9 comparé à un traitement avec un cœur SIMD. Grâce à la mémoire tampon intégrée, l'endurance de la mémoire non volatile n'est pas impactée et de fait, l'espérance de vie du système s'en trouve augmentée par rapport à d'autres solutions de calcul en mémoire.

Mots clés : calcul en mémoire, mémoire non volatile, architecture des systèmes, mémoire de classe de stockage, mur mémoire, mur énergétique, goulot d'étranglement de von Neumann

Abstract

Today computing centric von Neumann architectures face strong limitations in the data-intensive context of numerous applications. The key limitation is the memory wall due to increased performance gap between processors and memories. To mitigate this gap, cache hierarchy was introduced but it largely increased energy consumption while not being adapted for modern big datasets. Not only those architectures struggle with big datasets due to their high energy consumption and slow bandwidth, they can no longer be improved through technological advances such as node scaling. This calls for a paradigm shift to data centric architecture where treating massive amounts of data in a parallel fashion is the core principle.

New emerging Non-Volatile Memories (NVM) promise high density data storage and can easily integrate In-Memory Computing (IMC). IMC purposes is to compute where the data is or the closest to, to suppress back and forth data movements from the memory to the cores. Existing solutions use analog computing that has high efficiency but limited flexibility. When data needs to be written back after computation, endurance of NVM is often not discussed. We design a digital wrapper that extends memory functionality with vector computing capabilities and develop a simulation platform for architecture exploration. Our digital wrapper, aka C-SRAM, can be integrated with most memory technologies and comes with its own small SRAM buffer. We demonstrate that computing at the top of the memory hierarchy, i.e. close to the permanent storage, grants in average 17.4× energy reduction and 12.9× speed-up versus SIMD baseline. Thanks to SRAM buffer, NVM's endurance is not impaired and thereby extends system lifetime compared to other IMC solutions.

Keywords: memory wall, energy wall, von Neumann bottleneck, in-memory computing, non volatile memories, system architecture, storage class memory

Tout vient à point à qui sait attendre

Remerciements

Remarquez, y'a les voisins de mes vieux, ils ont quatre fils, y'en a un il est un peu attardé, et ben c'est leur préféré.

— Perceval IN *Kaamelott* BOOK II, EPISODE 97, « *Le tourment II* »

Not all who wander are lost.

— J. R. R. Tolkien

Although this thesis is written in english, all my work was done in a french environment, so I will thank people in my native language.

Et voilà, 12 ans après avoir passé mon bac, je finis enfin mes longues études. Études qui ont été un peu erratiques au début avec un passage raté par l'EPFL et une inscription en IUT sous l'impulsion de ma mère. Tout ça pour finir par ne pas si mal rebondir et intégrer l'ENS par la petite porte mais aussi parce que la lumière était allumée. Il en a coulé de l'eau sous les ponts entre mes projets de lycéen de devenir astrophysicien et maintenant en étant bientôt, si ~~Dieu~~ le jury le veut bien ou plutôt m'estime digne de l'être, docteur et rentrer dans la grande famille de la recherche publique française. Cette thèse aura été bien compliquée entre mes quelques problèmes de santé mentale mais aussi parfois physique, et le COVID (viva el COVID) et les confinements associés. J'ai appris à beaucoup relativiser (peut-être même trop) et à ne plus stresser sur des détails dans la vie de tous les jours qui n'ont finalement aucune importance. Avec cette thèse, j'ai beaucoup appris aussi bien humainement que techniquement.

Tout d'abord, je tiens à remercier mes encadrants : **Jean-Michel** et **Jean-Philippe** pour leur très grande patience et leurs conseils avisés. Bien évidemment, sans eux, je ne serais pas là pour écrire ces remerciements. Non pas parce que la thèse ne serait pas tout court, mais parce que j'aurais très certainement abandonné pendant le premier confinement (viva el COVID, *bis repetita*). Mais d'une certaine manière, je pense qu'il m'a aussi permis de finir ma thèse, mais je ne m'explique pas pourquoi j'ai cette impression. Pour **Jean-Michel**, merci pour les conseils scientifiques au sens large : qu'est-ce qui est important, qu'est-ce qui ne l'est pas. Pour **Jeanϕ**, merci pour ces échanges sur la culture française, et pas n'importe laquelle, la culture du général de Gaulle. Ensuite, je souhaite remercier les membres du jury pour avoir accepter d'être rapporteur pour ma thèse : **Alberto Bosio**, **Lorena Anghel** et **Gilles Sassateli** et à qui je souhaite évidemment une bonne lecture. Enfin, je remercie également **Mathieu Moreau** pour ses dernières corrections. J'ai l'impression que tu as essayé tous les liens dans ma thèse :).

Je ne peux pas oublier l'équipe avec qui j'ai travaillé pendant ces trois longues années, à moitié enfermé chez nous (viva el COVID, jamais 2 sans 3) : **Bastien**, **Maha** et **Henri-Pierre**. Bienvenue aux petits nouveaux dans l'équipe : **ValentinG**, **Maria** et **Hichem**, j'espère que vous vous y plaisez. Un grand merci spécial pour **Lorenzo** qui m'a beaucoup aidé personnellement et à qui j'ai pu aussi me confier (et réciproquement ;)) dans les durs derniers moments de ma thèse.

Merci à tous les gens du labo avec qui j'ai échangé, que ce soit techniquement ou juste pour la discussion autour de la machine à café : **Florent**, **Jean-Fred**, **Yves**, **Guillaume**, **Ivan**. Merci à **Mariam** pour son aide sur le chapitre 3 et le placement routage. Merci à **César** et **Éric** pour leur aide sur l'architecture et les caches pour les chapitres 4 et 5. Merci à **Manuel** qui m'a beaucoup appris sur quasiment tous les outils, pour son super niveau technique et ses discussions toujours intéressantes ;). Enfin, un merci spécial aux responsables scientifiques, **Yvain** (j'attends toujours le gateau au chocolat de ta fille) et **Pascal** qui malgré leurs agendas de ministre technique avaient toujours du temps à accorder à nous autres, pauvres thésards. Merci à **Simone** pour son sourire radieux qui illuminait le labo tout entier quand elle passait nous voir, bonne chance dans ta nouvelle carrière. Merci à **Marjorie** qui m'a accordé un peu de son temps pour être son cobaye quand j'avais besoin d'aide.

Une petite pensée pour tous les jeunes du labo, anciens thésards ou stagiaires que j'ai cotoyé : **Andrea** (merci pour le café), **Thomas** expert vétérinaire en Python, **François** pour les conversations un peu déjantées du midi (et désolé pour les autres autour de la table), **Nicolas** j'espère que ton dos, tes poumons et ta santé vont mieux, **Sota** qui nous a quitté pour la Suisse, **Kevin** et **Mona** avec qui j'ai partagé l'espace ouvert, **Antoine P.** qui vient de débuter sa thèse au CEA, courage tu vas en avoir besoin, **Antoine H.** qui a malheureusement quitté l'aventure doctorale un peu précipitamment (en plus, elle était glaciale) et les autres thésards : **Miguel**, **Manon**, **Paola**, **Housseim** du LGECA et tous ceux que j'ai oubliés (lisez quand même le paragrahe suivant :)).

Un grand merci aux potos avec qui j'ai passé tant de soirées chez eux ou au bar, ma vie à Grenoble ne serait pas grand chose sans vous : **Adrien**, psychologue à ses heures perdues et analyseur de ~~spectre~~ personnalité en ayant un côté suisse (neutre et externe) ; **Maxence**, marketeux qui a embrassé sa carrière scientifique plutôt que commercial mais qui pourrait vendre du sable à un bédouin ; **Stéphane** expert es barbecue et es bricolage ; **Roman** qui a essayé tous les bars de Grenoble et toutes les bières dans chaque bar, merci pour tous ces échanges techniques et humains et bon retour parmi nous.

Une dernière chose à remercier et sans quoi aucune recherche dans le monde ne serait possible dans ma conception toute personnelle : la machine à café qui nous a (presque) toujours bien servi.

One last thought for the online people who helped me relax after a hard day of work. Special thanks to **deen** who maintained DDNet tirelessly for almost 10 years now. Thanks to the *old* guys team for the games on fridays or saturdays evening: **|Ŗaþturę|** (you will survive), **Sir Skeleton** (where are you?), **Sir Kruksic** (who are you?), **Kasia** (Weedo is alive), **MyStery.Fox**. Hello to the dev team: **Learath2**, **heinrich5991**, **Jupstar ✪**, **Ryozuki**, **Robyt3** and the newcomer **Voxel**.

Coucou à toute l'équipe française de TeeWorlds : **Pipou** qui fait survivre la communauté française, **Cireme**, **iParano**, **snailx3**, **cris** aussi membre de l'équipe des *vieux*, la deuxième génération avec **Fluday** seul survivant, et la troisième génération avec **Véna**, **Neben**, **PlantKnight**.

Thanks to all people who contribute to the UNIX systems, Linux and the plethora of (sometimes useless) tools that makes life easier.

Table of Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

List of Acronyms

ADC	Analog Digital Converter. 31, 35, 39, 42, 44, 46
AI	Artificial Intelligence. 7, 23, 27, 32–34, 49, 64, 116, 118
ALU	Arithmetic & Logical Unit. 26, 47, 51, 53–55, 62, 118
BLAS	Basic Linear Algebra Subprograms. 65, 66, 98, 118
BRAM	Block Random Access Memory. 7
CAM	Content Addressable Memory. 10, 32
CBRAM	Conductive Bridge Random Access Memory. 18
CF	Conductive Filament. 18, 19
CIM	Computing In-Memory. 29, 33, 38
CNN	Convolutional Neural Network. 61, 64–67
CPI	Cycles Per Instruction. 49, 50
CPU	Central Processing Unit. xxi, 3, 5–7, 10–12, 14, 17, 22, 25, 27, 30–34, 38, 40–42, 52, 53, 65, 69–72, 76, 80, 82, 85, 87, 91, 93, 96, 97, 99–101, 105, 106, 109, 111, 115–121
C-RB	Computing Row Buffer. 92, 103
C-SRAM	Computational SRAM. 92
DBT	Dynamic Binary Translation. 71, 72
DLP	Data Level Parallelism. 5, 6
DNN	Deep Neural Network. 39, 61
DRAM	Dynamic Random Access Memory (see Section 1.2.1.2 for more details). xxi, 6, 7, 10–17, 19, 21, 22, 24, 26–28, 32–35, 38, 40, 41, 53, 67, 69, 70, 73, 74, 76–78, 82–85, 87–90, 92–99, 101, 103, 104, 107, 109–113, 115, 116, 119–121

DSP	Digital Signal Processor. 7
DVFS	Dynamic Voltage and Frequency Scaling. 5
EDA	Electronic Design Automation. 46, 48
FPGA	Field Programmable Gate Array. 7, 25, 27, 40, 41
FS	Full System. 71, 72
FSM	Finite State Machine. 47, 54, 55
GPU	Graphic Processing Unit. 6, 7, 25, 27, 33, 34, 41, 70, 83
HBM	High Bandwidth Memory. 6, 12, 27, 33, 116
HDD	Hard Disk Drive (see Section 1.2.1.3 for more details). 6, 11–15, 17, 22, 116
HMC	Hybrid Memory Cube. 33, 116
HPC	High Performance Computing. 8, 11, 42, 43, 89, 93, 120
HRS	High Resistive State. 16, 19–21
IMC	In-Memory Computing. 1, 17, 23, 25, 26, 28–30, 32–36, 39, 41–46, 48, 49, 59–61, 64, 65, 72, 73, 85, 90, 91, 116–118, 121
IoT	Internet of Things. 2, 23, 38, 43, 49
IP	Intellectual Property. 48
IPC	Instructions Per Cycle. 8
IR	Intermediate Representation. 72
ISA	Instruction Set Architecture. xiii, 26, 30, 39, 41, 42, 52–55, 80, 90, 117–119, 121
LFB	Line Fill Buffer. 69
LIM	Logic In Memory. 29, 37

LLC	Last Level Cache. 69
LOP	Low Operating Power. 74
LRS	Low Resistive State. 16, 19–21
LSQ	Load Store Queue. 69
MLC	Multi Level Cell. 76
MMU	Memory Management Unit. 80
MRAM	Magnetic Random Access Memory (see Section 1.2.2.3 for more details). xx, 18, 21, 22, 39, 74, 117
MTJ	Magnetic Tunnel Junction. 21
MVM	Matrix Vector Multiplication. 42, 64, 65, 116–118
NMC	Near-Memory Computing. 26, 29, 30, 32, 35, 40, 41, 48, 72, 73, 116
NMP	Near-Memory Processing. 29, 39, 40
NVM	Non Volatile Memory (see Section 1.2.2 for more details). 1, 15, 20, 22, 27, 35, 40, 41, 43, 44, 48, 49, 62, 72–74, 76, 78, 85, 91–95, 102–105, 107, 109, 111–113, 116, 117, 119, 120
OoO	Out of Order. xxi, 5, 72
OS	Operating System. 17, 70, 72, 80
OxRAM	Oxide Random Access Memory. 18
PCM	Phase Change Memory (see Section 1.2.2.2 for more details). 10, 18, 20–22, 39, 40, 48, 73–75, 90, 117, 119, 120
PCRAM	Phase Change Random Access Memory (see Section 1.2.2.2 for more details). 75
PIM	Processing In Memory. 29, 33, 34, 37, 40, 41, 116
PMU	Performance Monitoring Unit. 68, 69

RAPL	Running Average Power Limit. 70
RAW	Read After Write. xxi, 55, 118, *Glossary:* pipeline hazards
RRAM	Resistive Random Access Memory (see Section 1.2.2.1 for more details). 10, 18–22, 36–40, 45, 48, 74, 117
RTL	Register Transfer Level. 48
SA	Sense Amplifier. 10, 11, 16, 25, 31, 35, 38, 42, 116
SCM	Storage Class Memory. 1, 15, 21, 22, 28, 35, 40, 48, 65, 67, 82–85, 92–96, 99–104, 112, 119–121
SE	System Emulation. 71, 72
SIMD	Single Instruction Multiple Data. 3, 5, 6, 27, 30, 31, 34, 41, 72, 80, 92, 95, 96, 107, 111, 115, 120
SIMT	Single Instruction Multiple Threads. 6
SLC	Single Level Cell. 35, 36
SRAM	Static Random Access Memory (see Section 1.2.1.1 for more details). 1, 10–16, 18, 19, 21, 22, 25, 26, 28, 30, 32, 33, 35, 38–49, 54, 55, 58, 59, 61, 62, 74, 75, 116–118
SSD	Solid State Drive (see Section 1.2.1.4 for more details). 13–15, 17, 20, 35, 40
STT-MRAM	Spin Transfer Torque MRAM. 21, 39, 40
TCAM	Ternary Content Addressable Memory. 31, 39
TTM	Time To Market. 46, 48
WAR	Write After Read. xxi, *Glossary:* pipeline hazards
WAW	Write After Write. xxi, 55, 118, *Glossary:* pipeline hazards

Glossary

API	An Application Programming Interface is a particular set of rules and specifications that a software program has to follow to access and make use of the services and resources provided by another particular software program that implements that API. 26, 98
LRU	Least Recently Used is a cache replacement policy that replaces the oldest used block in the cache. It is simple yet quite effective. 82, 84
Pipeline Hazards	Occurs when a data is read or written simultaneously at different stages of the pipeline. This is prominent in OoO CPU due to instruction reordering but can also happen in normal pipelines. It includes Read After Write (RAW), Write After Read (WAR) and Write After Write (WAW). 50, 118
Swapping	Swapping is saving a dirty page from DRAM to disk when the former is full. When the page is needed again, it is loaded back from disk and if the DRAM is still full, another page is saved to disk (i.e. pages are swapped). 83–85, 87, 101, 103, 119, *see also* thrashing
Thrashing	Thrashing is a phenomenon where pages are constantly swapped from DRAM to disk such that the operating system becomes unresponsive. In this thesis, thrashing can also denote the same effect but between DRAM and C-SRAM. 98, 109, *see also* swapping

Preamble

Tiens, je ferais bien une partie de quelque chose. Un truc qui se joue vite. On pourrait faire un Sloubi. Je vous explique, y'en a pour 2 secondes.

— Perceval IN *Kaamelott* BOOK III, EPISODE 50, « *Perceval chante Sloubi* »

To err is human, but to really foul things up you need a computer.

— Paul R. Ehrlich

This thesis is divided in five chapters that will introduce you to the *why* am I doing this thesis: the global context raises issues about computing performances and efficiency that requires an innovating solution (Chapter 1). In-Memory Computing (IMC) is a promising solution compatible with new emerging Non Volatile Memories (NVMs) that also brings new technological improvements in the computer world. We study state of the art in Chapter 2 and show how it misses two key points about NVMs endurance and where to compute in the memory hierarchy. We propose our solution, a digital wrapper around a Static Random Access Memory (SRAM) that we call C-SRAM (Chapter 3). Our C-SRAM can then be tightly coupled to others NVMs or Storage Class Memories (SCMs). To perform an architectural evaluation, we develop a simulation platform fed with technological parameters from state of the art and our own works (Chapter 4). Putting it all together, we show that computing at the top of the memory hierarchy, i.e. close to mass and permanent storage, yields the most gains for both execution time and energy reduction (Chapter 5).

In this thesis, each chapter starts with a *funny* quote from KAAMELOTT, especially from Perceval who will teach you games from Wales country. A more serious quote is also present to give the reader a slight taste of the chapter. Each chapter begins with a small summary of the topics discussed within. You will find some outlines in light blue box (see below) to recapitulate the main points developed in sections and subsections.

I hope you will enjoy reading this thesis as much as I enjoyed writing this final sentence.

1. On the semiconductor industry

Normalement, ça se joue avec des bouts de bois. Il faut 50 bouts de bois de 2 pouces, 50 de 3 pouces, 50 de 4 pouces et ainsi de suite. Et à la fin, il faut 50 poutres de la longueur de la pièce. Vous avez ça ou pas ?

— Perceval IN *Kaamelott* BOOK III, EPISODE 50 « *Perceval chante Sloubi* »

Everything has its limit – iron ore cannot be educated into gold.

— Mark Twain

Hardware design comes to the end of its golden era where a simple wait of a few months could yield huge improvements for both performance and energy consumption. This was mainly driven by technology scaling and moving to smaller and more advanced nodes. However, as industry reaches the smallest possible node (3 nm), progress can no longer come from technology itself but must come from finer architecture and software designs to better utilize hardware. Famous von Neumann architecture where memories and computing are physically separate and logically distinct units must evolve to face new computing requirements posed by recent rise of big data applications and artificial intelligence. Internet of Things (IoT) devices are also presenting a challenge for energy efficient designs in the wake of societal changes in regard to global warming and energy sobriety.

Contents

This first chapter will give the reader a very wide introduction and contextualisation on semiconductor technology facing the end of a cycle with a halt to miniaturisation and other well-known obstacles to densification. The key to improve performance is to add more transistors into circuits. However this is defined by physical limits that have been or are being reached nowadays, including energy and memory wall problems (section 1.1). New technologies that may resolve partially these problems are being introduced, especially for emerging memories (section 1.2). These new memories enable a new computing paradigm to solve the admitted von Neumann bottleneck that is exacerbated by big data applications and the rise of artificial intelligence (section 1.3).

1.1. The end of technology advancement

1.1.1. Physical limits

For years, what has driven the semiconductor industry progress is the technology scaling, i.e. the miniaturisation of the transistor. Reducing the transistor, base unit of all the digital world, by a factor of $\sqrt{2}$ leads to a doubling in the total number of transistors in the same area. Gordon Moore predicted that this doubling would occur every 24 months, later revised down to 18 months. This is known as the Moore's law [1] which held true for almost 50 years (Figure 1.1). More transistors equals more functionalities or more complex ones; as such, we have seen parallel computing emerged during the 2000s with Single Instruction Multiple Data (SIMD) and multicore Central Processing Units (CPUs). However, miniaturisation has limits that cannot be exceeded. It is physically impossible to make a transistor that is smaller than a few atoms and we are already hitting this limit with 3 nm. This means that to answer the growing need for more computing power, semiconductor industry will have to rely on better architectural designs and smarter software models.

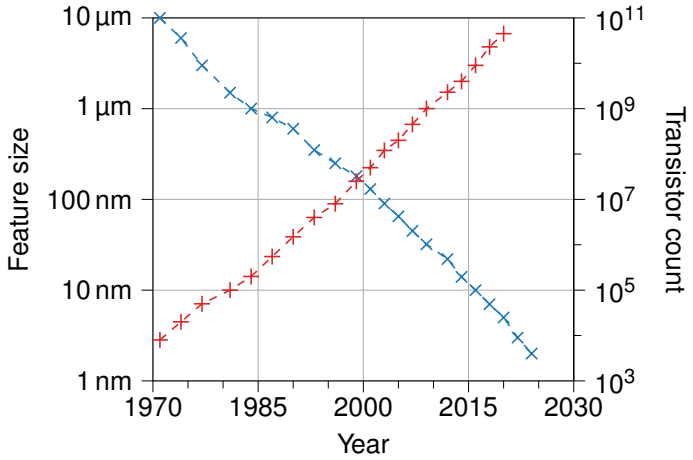


Figure 1.1.: ITRS roadmap as of 2020 (×, left axis) and transistor count per chip (+, right axis)

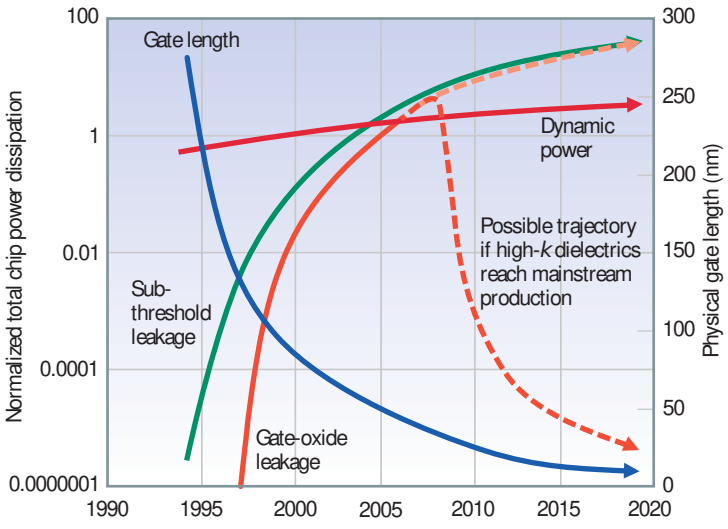


Figure 1.2.: Transistor leakage evolution. From [3]

Table 1. Technology and circuit projections for processor chip components.

	2010	2017	
Process technology	40 nm	10 nm, high frequency	10 nm, low voltage
V_{DD} (nominal)	0.9 V	0.75 V	0.65 V
Frequency target	1.6 GHz	2.5 GHz	2 GHz
Double-precision fused-multiply add (DFMA) energy	50 picojoules (pJ)	8.7 pJ	6.5 pJ
64-bit read from an 8-Kbyte static RAM (SRAM)	14 pJ	2.4 pJ	1.8 pJ
Wire energy (per transition)	240 femtojoules (fJ) per bit per mm	150 fJ/bit/mm	115 fJ/bit/mm
Wire energy (256 bits, 10 mm)	310 pJ	200 pJ	150 pJ

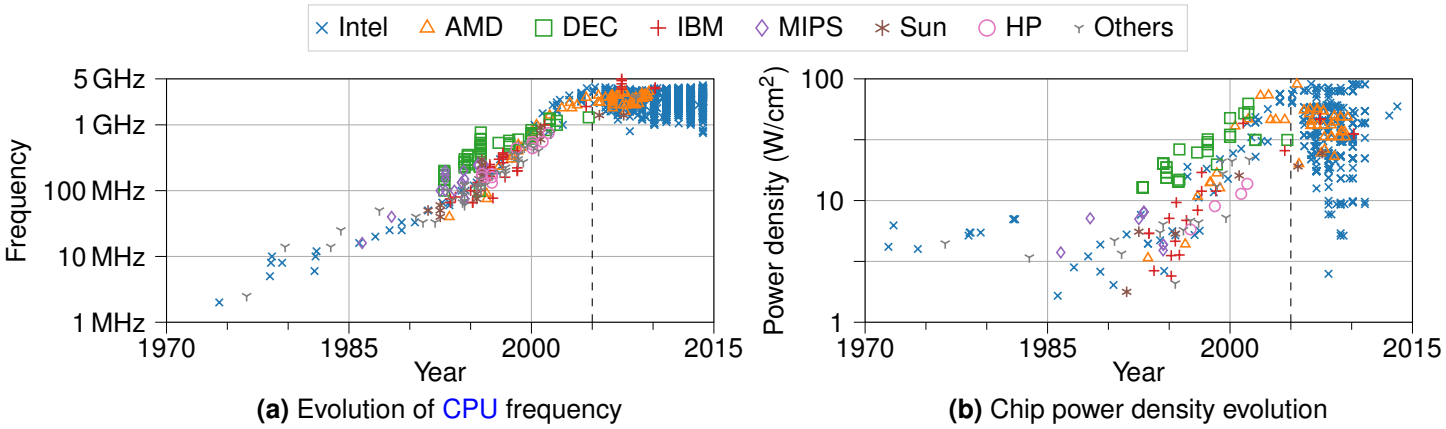
Figure 1.3.: Predicted scaling cost in 2010 (45 nm) for 2018 (10 nm). From [4]

Moving forward to more advanced nodes has also some technical limits that can be seen as side effects for laypeople readers. Smaller transistors are more leaky as the space between different voltage domains is also reduced. On the other hand, it allows to reduce voltage because the threshold voltage is lowered as well. All in one, this leads static power to be dominant in nodes smaller than 90 nm [2] and to increase for each smaller node (Figure 1.2). Moreover, although reducing transistor leads to gain in dynamic power, wire cost is not scaling down with the same tendency. Copper resistivity remains constant and data transit over long wires still stands as the main power sink in every design, especially when memory is on a chip of its own. This is shown in Figure 1.3 dating from 2011 that forecast this difference in power reduction from computing complex operations compared to transmission that will expand four times between 2010 and 2017.

Another problem linked to miniaturisation is Dennard's scaling [5]. It states that as transistors shrink, their power density remains constant. This was true from 1974 to approximately 1995. At that point, power density started to increase (Figure 1.4b) and it ultimately limited frequency increase. Indeed dynamic power is determined by two main factors which are the voltage and the frequency:

$$P_{\text{dyn}} = CV^2 f$$

Voltage is fixed by the technology and cannot go below the threshold voltage plus the line loss. C is the parasitic wire capacity swung at every clock cycle and is also a fixed parameter of the technology. So we can only play on the frequency but as we want the most performance, we tend to push it to the maximum acceptable limits by the design, i.e. the maximum power we can either deliver or dissipate. As power density increased, it soon started to be impossible to rise frequency without damaging the circuit hence a frequency saturation from 2005 as shown in Figure 1.4a. These technology problems are physical limits that cannot be broken without a new disrupting technology such as



(a) Evolution of CPU frequency

(b) Chip power density evolution

Figure 1.4.: CPUs evolution over years. 2005 Dennard's scaling break is visible in both graph. From [7]

optronic or spintronic that could leverage them. It also led to the famous expression by Herb Sutter: "*The free lunch is over*" [6].

1.1.2. Architecture improvements

Industry now faces a double challenge, the impossibility to increase working frequency and the increase in leakage current when moving on to more advanced nodes. To keep performance development in their chips, industries introduced multiple workarounds: Single Instruction Multiple Data (SIMD), multicore and Out of Order (OoO). First, SIMD CPUs were developed to treat multiple data in a single instruction using vector larger (128 bits or more) than the base register (32 or 64 bits at that time). SIMD exploits intrinsic Data Level Parallelism (DLP) in applications. The widest SIMD processor supports up to 512 bits vectors. Secondly, multicore designs permit two independent instruction flows to execute concurrently although they share some hardware, especially memories above L2 or L3 and buses. In some recent commercial chips, up to 64 cores can be used in parallel [8]. Third, OoO CPUs introduction improved compute unit use and reordering of instructions allowed CPUs to mitigate memory timings on independent data paths. With multiple compute units available, processors are said to be superscalar, i.e. capable of executing multiple instructions simultaneously. These three improvements however reached the limit to their computing performances due to power constraint, and insidiously led to the apparition of dark silicon [9]. This happens when complex circuits cannot be fully powered permanently or simply overheat and need to dynamically choose which part to power or to adjust either voltage or frequency using Dynamic Voltage and Frequency Scaling (DVFS). The latter was adopted by the industry because it allowed more flexibility and less *stuttering* in data streams. An example is given for Intel multicore chips and the use of SIMD extensions in Table 1.1. Dark silicon reveals the low energy efficiency of these designs.

Table 1.1.: Frequency scaling of an Intel Xeon Silver 4116, a 12 cores chip, function of active cores and active SIMD extension. From [10]

Mode	Base	Turbo Frequency/Active Cores											
		1	2	3	4	5	6	7	8	9	10	11	12
Normal	2.1 GHz	3.0 GHz	3.0 GHz	2.8 GHz	2.8 GHz	2.7 GHz	2.7 GHz	2.7 GHz	2.7 GHz	2.4 GHz	2.4 GHz	2.4 GHz	2.4 GHz
AVX2	1.7 GHz	2.9 GHz	2.9 GHz	2.7 GHz	2.7 GHz	2.4 GHz	2.4 GHz	2.4 GHz	2.4 GHz	2.1 GHz	2.1 GHz	2.1 GHz	2.1 GHz
AVX512	1.1 GHz	1.8 GHz	1.8 GHz	1.6 GHz	1.6 GHz	1.5 GHz	1.5 GHz	1.5 GHz	1.5 GHz	1.4 GHz	1.4 GHz	1.4 GHz	1.4 GHz

Not only those improvements are not sustainable in the long term, they also put pressure on other system components, typically on the memory system (Figure 1.11). For SIMD, memory now has to serve request up to 512 bits instead of scalar data of 32 or 64 bits. Caches are designed to respond swiftly to these requests but when they would have to serve only a single data, they now have to load large batch of data which increases their power consumption. As vector CPUs treat batch of data, which is now the size of a cache line, caches experience a high miss rate putting more pressure on the slower Dynamic Random Access Memory (DRAM) which becomes the von Neumann bottleneck. This is worsened by multicores because each core will request data to DRAM that the L3 cache cannot store due to its limited capacity. So now, DRAM has to deliver data to several cores simultaneously instead of just one. Each core having its own data set, data locality is reduced which also impacts caches and DRAM performance. This is illustrated in Figure 1.5 where performance of CPUs increase faster than which of memories leading to a performance gap between the computing and the memory systems. This is what is called the memory wall, because the memory cannot deliver data fast enough and the CPU just waits doing nothing. Note that above DRAM, Hard Disk Drive (HDD) and Flash disks have long been surpassed and cannot compete in terms of bandwidth with the need of modern CPUs nor of DRAMs.

To keep increasing throughput and energy efficiency, Graphic Processing Units (GPUs) were pushed in. They use Single Instruction Multiple Threads (SIMT) approach, i.e. different threads all executing same instruction on different data with predicates to allow branches and conditional execution to occur. It heavily simplifies the internal design of the processing elements making them more compact so that thousands can be put on a single chip. This benefits to application with heavy DLP such as filtering an image where the same operation is carried on all pixels with conditional code to handle edge cases. GPUs come with their own main memory, nowadays of type High Bandwidth Memory (HBM) with 256 bits IO and high bandwidth. They also have their own internal caches with 2 levels of cache. Nonetheless, initial data transfer from system main DRAM memory to GPU's memory must still take place before the algorithm runs and data must be sent back once it is done. This back and forth can end up representing more than 90 % of the total execution time depending mainly on the algorithm complexness and the data set size [11]. Overall, GPUs work pretty well on the same regular access patterns as CPUs. They also provide similar program flow with a wide range of complex instructions. Their massive parallelism is used to build some of the Top500 supercomputers [12, 13].

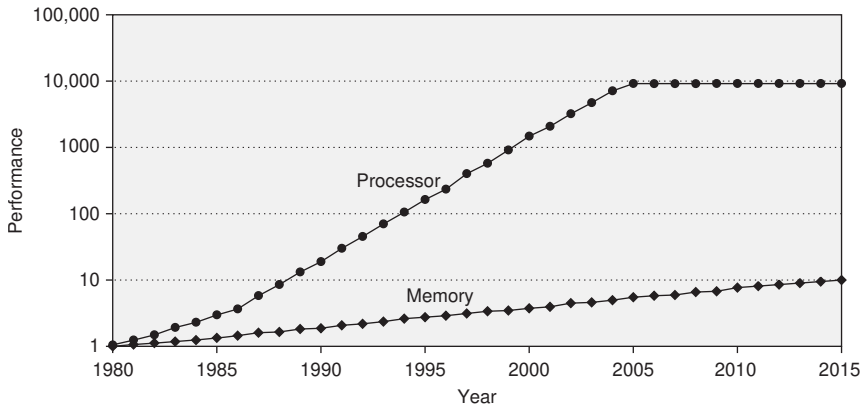


Figure 1.5.: CPU and memory performance trends. From [14]

However, both CPUs and GPUs are very generic and can be considered as swiss knives of computing. They do the job but not in a very efficient way except for regular linear access patterns. To improve energy efficiency and throughput, co-processors dedicated to specific tasks were designed, most common one being the Digital Signal Processor (DSP) for embedded systems with real time constraints. Unfortunately, the need for more, better and greener computing requires flexibility that these extra co-processors do not offer. Field Programmable Gate Arrays (FPGAs) are yet another possible mean to gain extra performance by allowing CPU to turn part of itself into a highly energy efficient application specific accelerator and bridge the gap between flexibility of use and efficient designs. Their programmability combined with their natural energy efficiency makes them suitable candidates for use as co-processors. They come with their own memory in the form of Block Random Access Memory (BRAM) with wide IO to feed their natural data level parallelism. Unfortunately, these BRAMs still need to be filled from another external memory which is often DRAM, but FPGAs do improve energy efficiency. So the main problem of memory wall is still there for initial and final data transfer, just like for GPUs. Another step further is using Application Specific Integrated Circuits (ASICs), which are fixed designs but with even better energy efficiency and throughput than FPGAs, but once again, the memory wall remains.

All these hardware solutions are to boost *classic* algorithms performance but there was also the breakthrough of new algorithms in the last decade, mainly Artificial Intelligence (AI) with neural networks. AI is a solution to treat massive amount of data and extract meaningful tendencies but it comes with its own *data* that are the neurons parameters which can also be counted in billions for some networks. Aforementioned hardware solutions can all improve neural networks performances but all end up hitting the memory wall.

The race for best performances, although a great source of hardware improvements such as branch predictors, prefetchers and so on, induced a rising complexity of CPUs that led to some security flaws [15]. But it also drives for more power and ironically reduces energy efficiency [16]. The industry focused on instruction centric paradigm

Integer		Floating Point		Memory	
Addition				Cache	64 bit access
8 bit	0.03pJ	16 bit	0.4pJ	8kB	10pJ
32 bit	0.1pJ	32 bit	0.9pJ	32kB	20pJ
Multiplication				1MB	100pJ
8 bit	0.2pJ	16 bit	1.1pJ	DRAM	1.3-2.6nJ
32 bit	3.1pJ	32 bit	3.7pJ		

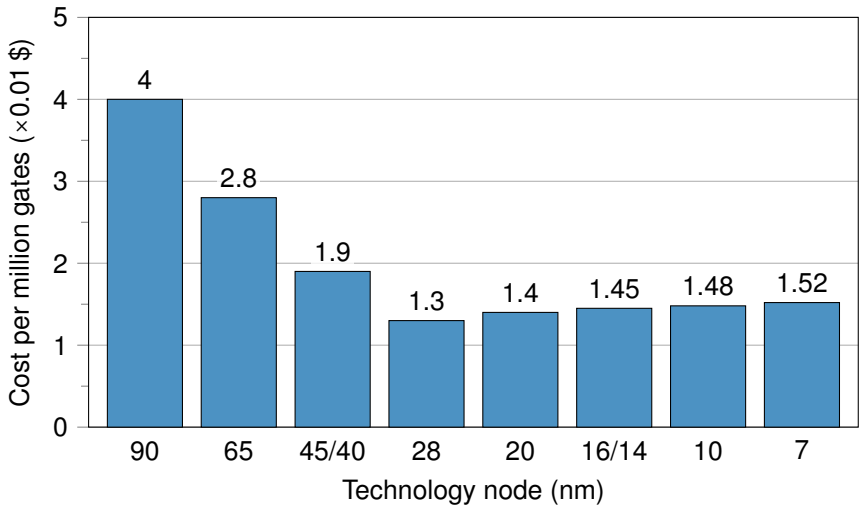
Figure 1.6.: Instruction energy breakdown along with some energy consumption of common instructions and memory accesses. From [17]

where everything was done to increase throughput of instructions, measured in Instructions Per Cycle (IPC). But when looking at the energy bill of simple instructions (Figure 1.6), we see that this is not very efficient as most of the energy comes from moving the data around. With the introduction of *big data* and artificial intelligence applications that uses huge batches of data, this calls for a shift to data centric architectures to solve all the two major challenges: the von Neumann bottleneck aka memory wall and the energy wall or dark silicon.

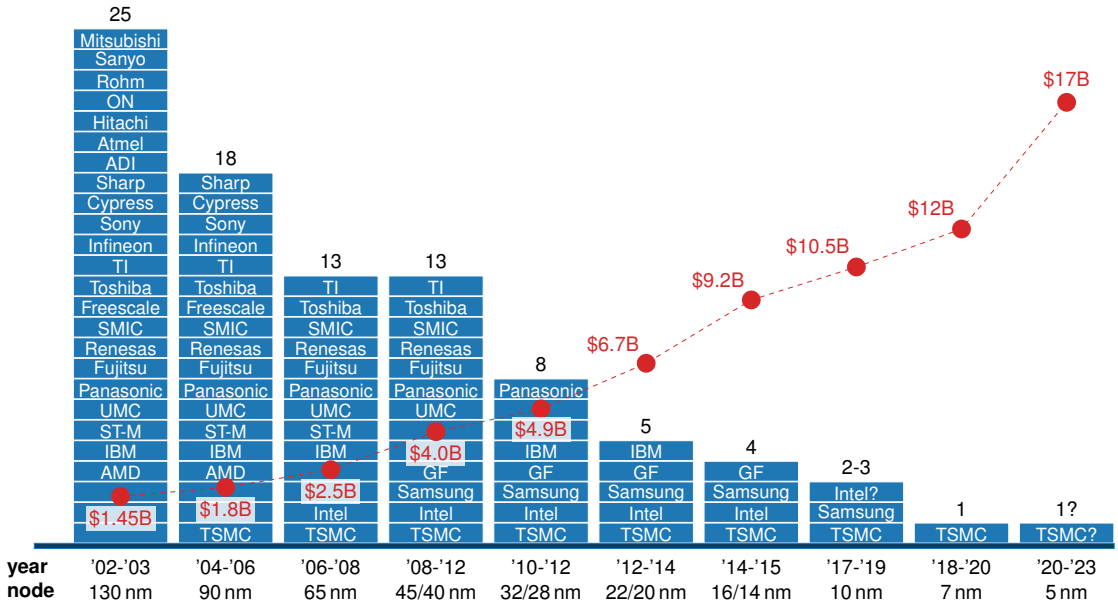
1.1.3. Socioeconomic impacts

This changing paradigm is in accordance with the evergrowing need for greener computing and better energy efficiency in data centres. High Performance Computing (HPC) centres are reaching tens of megawatt of power consumption which is the equivalent of a 20 000 inhabitants city [12]. Another important point is the economic cost of moving to more advanced nodes which grants no more benefits due to the rising cost of state of the art technologies presented in Figure 1.7. One more part is on water consumption of the semiconductor industry that requires large quantity of extremely pure water which is already a problem due to water shortage in Taiwan. Environmental rejects of different pollutants also need to be accounted for [18].

Up to this point, we have presented the global context and the challenges facing the semiconductor industry for the following years: no more possible scaling, no more power and a growing need for more energy efficient computing. These challenges call for either a shift to different technology or to rethink the architecture of systems to better use them. The best way to reduce energy consumption is to minimize data movement. In the following section, we present the standard memory technologies and the emerging memories that appeared in the last decade.



(a) Chip cost per million gates (in $). Cost stopped decreasing after 28 nm in 2012 and slightly increased for following nodes.



(b) Founders per node and the associated investment cost (●). High investment cost causes the number of founders to drastically decrease leading to potential monopoly and strategic dependency

Figure 1.7.: Cost of chips and investment needed for the founder. The decrease in cost per million gate could finance the investment for the next node before 2012 and the 28 nm node. From [19]

1.2. Memory technologies

Previous section dealt mainly with CPUs which is the core of computing systems. We have shown that instruction centric architectures faced a soon to come dead end due to energy and memory walls. This section introduces main memory technologies such as Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM) but also persistent storage to give the reader a broad range of possibilities and perspectives with their associated limitations which represent a major challenge in the data movement cost. Emerging memory technologies including Resistive Random Access Memory (RRAM) or Phase Change Memory (PCM) are presented along with their remaining challenges to make them viable economically and offer substantial benefits for system architects over conventional memories.

1.2.1. Main memory technologies

The main memory technologies are the most common ones that can be found in any consumer device. They are the most mature ones and present in the market for decades. However they have some intrinsic design flaws such as high leakage (whether dynamic or static power) or very high latency for non volatiles ones.

1.2.1.1. SRAM

Static Random Access Memory (SRAM) is a fast memory used in almost all existing CPUs dating back to 1964. It provides an extremely fast memory whose working clock frequency is above 1 GHz with virtually infinite endurance. The circuit diagram of a six transistors SRAM bitcell is shown in Figure 1.8. It is made up of two head to toe inverters and two access transistors. Read operation is performed by first precharging the bitlines to $\frac{V_{dd}}{2}$, then by activating the two access transistors and using a Sense Amplifier (SA) at the bottom of the bitlines to minimize the error margin. Write operation is done similarly by forcing the data on both bitlines which will switch the state of both inverters. However, the inverters are not perfect and leak, so the SRAM bitcell presents a high static power consumption. It is often arranged in large array, up to 8192 wordlines or bitlines which increases the dynamic consumption due to the large capacitance of the lines. To reduce dynamic switching power, bitlines are often split in local groups with access transistors to commute global bitlines.

Per se, the 6T bitcell is a 1 read-write (1RW) bitcell, which means that it can either be read or written once per cycle. SRAM bitcell have a large diversity as it also exists in 8T up to 16T. These extra transistors allow to add isolation between the bitcell and the bitlines so that read or write to several bitcells on the same bitline can occur concurrently. This is used to add more access port to the memory to make 1R1W, 1R1RW and even 2RW bitcells. Literature also shows that 6R6W bitcell is possible [20]. SRAM bitcell can also be used as Content Addressable Memory (CAM) memory that is commonly used in routers. Finally, as it is made up of six transistors, it has a very low density that does not allow to have large SRAM memory bigger than a few megabytes.

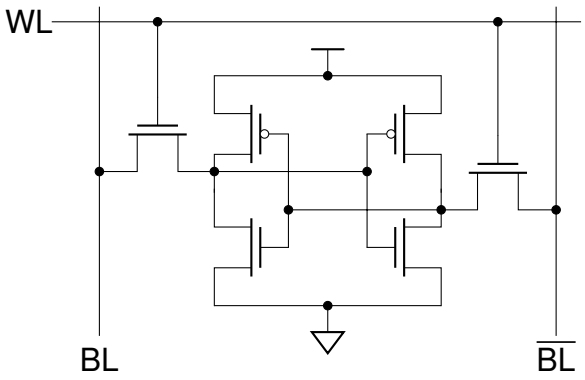


Figure 1.8.: SRAM bitcell circuit diagram

What makes SRAM so interesting is that it is a CMOS circuit that can be incorporated directly in chips design and scales down along with the technology. It is used as cache or scratchpad memory and is often tightly coupled to CPUs as it is the only memory to keep up the pace with high frequency. Other uses include small buffer memory in devices like HDD, Flash drives or anything that needs few amount of memory before transmitting over serial bus or medium that requires serialisation, e.g. radio transmission. Its flexibility allow designers to easily use custom SRAMs with wide IO or even asymmetrical IO (for serialisation for instance) as well as odd row number.

1.2.1.2. DRAM

Dynamic Random Access Memory (DRAM) is the main volatile memory in non embedded systems such as desktops, servers, HPC and even in some embedded systems like autonomous cars. It features an infinite endurance with medium speed (relative to SRAM) while having a very high density. Figure 1.9 shows the circuit diagram of a DRAM bitcell. It is composed of an access transistor and a capacitor to store the data. This capacitor is leaking so it needs to be refreshed periodically, hence the *dynamic* in the name. This leaking along with the refresh operation cause this memory to have a high dynamic power consumption even when the memory is idle. Read is performed by precharging the bitline to $\frac{V_{dd}}{2}$ and then by activating the access transistor. The capacitor then discharges or charges the bitline and a SA catches the difference. Read is thus destructive as the capacitor shares its charge with the bitline and the original data needs to be restored. Write is simply done by activating the access transistor and pulling the capacitor to the desired voltage (high for 1, low for 0).

To prevent the whole memory from being inaccessible during a refresh, DRAM is organized in ranks subdivided in chips and in banks. Banks are split across several chips for parallelism reason. Each bank is itself partitioned in subarrays which contain the wordlines and bitlines. Wordlines are referred to as logical rows that spans several chips while bitlines are logical columns. Columns are muxed in a similar fashion to SRAM. To read or write, a DRAM row must first be activated, i.e. selected, it is then loaded in the row buffer where read and write take place for faster operation. In

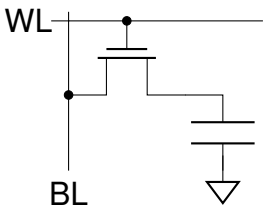


Figure 1.9.: DRAM bitcell circuit diagram

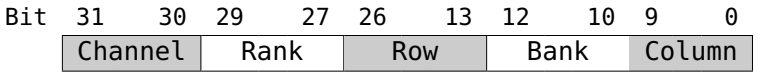


Figure 1.10.: Example of a DRAM addressing scheme

particular, burst mode allows several operations to contiguous addresses to happen with a single command and fully benefit from the row buffer. When all operations on the current row are done, either a different row within the same bank can be activated or a row in a different bank is selected. The former bank must first receive a *precharge* command to reset bitlines to $\frac{V_{dd}}{2}$ to minimize leakage before activating a row in another bank. When a row is closed, the row buffer is written back in place to restore data. Refresh affects a whole bank at a time and makes it unavailable until it is finished. The addressing scheme vary from chip to chip but it is mainly column first then bank then row as shown in Figure 1.10. Above ranks are channels which are physical buses and may be shared by several DRAM devices. Addressing schemes can also be interleaved or with some XOR between some bits to increase row-hit rate.

Banks are the physical output and have 8 bits IO. To have a 64 bits IO, 8 banks are disposed in parallel. The complicated rules and state machine to handle DRAM commands and its dynamic nature requires complex designs to ensure correctness. That is why CPUs have a portion of their area reserved for DRAM scheduling (see Figure 1.12b). However, DRAM's high density with its intermediate speed and high bandwidth makes it a suitable choice to fit between long term but slow storage and SRAM's high speed but low capacity. To answer the growing need for bandwidth, manufacturers have developed HBM that uses 3D stacking and have very wide IO (256 bits) compared to DRAM standards. Both are standardised by the JEDEC committee which makes DRAM a somewhat rigid memory format. Due to the complex state machine needed to respect timings and transitions, DRAM has a latency that can greatly vary between 20 ns to more than 400 ns. Finally, DRAM suffer from write disturb. Continuous write to the same row and bitcells by alternating activation, write and precharge leads neighbouring cells to be affected and even flipped due to parasitic capacitance between lines. The row hammer attack exploits this vulnerability [21]. Smaller nodes have more parasitic capacitance which augments this risks but also increases the leakage and reduces the stored charge which induces more refresh and more unavailability. As such, DRAM is limited in scaling and faces its own technological challenges.

1.2.1.3. Hard disk and tapes

HDDs and tapes are the most ancient forms of digital storages that are still in use today. They are also the only form of modern storage to use mechanical parts, i.e. an engine, incorporated for HDDs and external for tapes, to spin the disks or roll up and unroll

the tape. As such, they require a vibration free environment to be used safely. Shocks may damage data permanently, especially for HDDs and make the device completely unusable. Both HDD and tape have a high data density, not necessarily in surface but more in volume as disks can be easily stacked and tape film is really thin, around 10 µm. Largest commercial HDD is around 15 TB while tape goes up to almost 500 TB. The main cons of these technologies is obviously their very high latency around 10 ms for HDD while tape can go anywhere between one second to more than a minute depending on how far on the tape the data is. Main use of tape includes long term storage such as archiving or data backups for companies. One of unthought advantage of tape is being offline storage which protects data from online attacks. Endurance of both storages is not really a concern as mechanical parts wear out before it is reached.

1.2.1.4. NAND Flash

NAND Flash memory is the most common type of non volatile memory. It is more recent than SRAM and DRAM but the absence of mechanical parts allowed it to be used in numerous devices thanks to its non volatility. It is used in SD memory cards, USB sticks, smartphones and Solid State Drives (SSDs) for the most common devices. NAND Flash is made up of a single transistor with a floating gate which stores the information by retaining the charge after power down. Read is simply done by sensing the current flowing in the channel whereas write is more complex and requires several steps. First, due to how NAND Flash is built to maximize density, it is organised in blocks that cannot be written word per word but only as a whole. This means that even for changing a single bit, a full block must be written. Moreover, write operation requires the block to be erased before so data must first be read to keep non modified data intact. A block is typically around 512 B–4 kB.

The advantages of NAND Flash are its non volatility with high shock resistance thanks to no mechanical parts compared to HDDs or tapes. Besides, it has a very high density in comparison to SRAM and DRAM. NAND Flash indeed supports 3D stacking and most recent chips have up to 176 bitcells stacked [22]. This allows to have a virtual footprint of less than the theoretical minimum of 4 F^2. Moreover, each die is also vertically stacked with up to 16 other dies in a standard commercial SSD. This sums up to density superior to 100 Gbit/cm^2. On the other hand, NAND Flash is quite slow in regard to previous volatile memories. Its read speed is around 1 GB/s but its write speed is 5 times slower around 200 MB/s due to the erase operation. The main disadvantage of NAND Flash is its latency around 10 µs for reading which makes it around 100 times slower than DRAM. For writing, latency around 100 µs can be expected. These high latencies are due to the high voltage required to operate on the memory array, up to 15 V which takes some time to reach.

When people talk about memory, they often mention capacity, density and bandwidth but they rarely talk about endurance and persistence. Writing in NAND Flash requires high voltage which ends up damaging the cell after many programming cycles. This means that a NAND Flash has its lifetime determined by the write bandwidth and the capacity. To circumvent this problem, industrials added more memory to devices

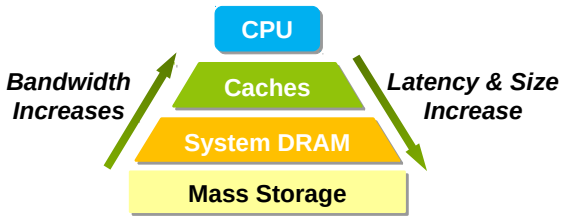


Figure 1.11.: Memory hierarchy in a conventional system. In server or cluster, DRAM and mass storage may be distributed or remote.

to be used when a block is failing. Commercial devices may have up to 20 % of extra memory. Another technique used is wear leveling. This allows to dynamically remap some blocks onto others to even the number of writes across the device. It also protects against write attacks aiming to destroy data by wearing out SSD prematurely. To manage wear leveling, NAND Flash devices embed a controller with their own SRAM memory that also allows to perform some operations on data. Finally, to speedup writes, SSDs may embed some DRAM to act as a write buffer but this is only effective if the amount of data is lower than the buffer size. Scalability is also a concern for NAND Flash as the high voltage needed to write the cell constrains the transistor size and limits the downscaling.

1.2.1.5. Current memory hierarchy

On one hand, CPUs need a working memory to store their temporary data. This memory can be a SRAM for small microcontrollers or DRAM for larger processors. On the other hand, a permanent storage is required to store programs and associated data. It is often made with NAND Flash or HDD. As explained in Section 1.1.2, processors have seen numerous architectural improvements to boost their performances. However, memories did not keep up the pace and, as a result, intermediate memories known as caches were introduced to mitigate timings. If we take a desktop or server CPU, its working frequency is around 3 GHz so it needs a memory to be the fastest possible to not waste clock cycles waiting for data. This is the goal of the L1 cache made in SRAM which is around 16–128 kB and usually have a latency of around 1–3 ns. To be able to serve an instruction and a data at once, there are often two L1 cache, one for instructions and one for data. To bridge the latency and capacity gap with the main DRAM memory, a L2 and a L3 caches were introduced. L2 has a capacity of 128–1024 kB and a latency between 5–10 ns while L3 can be up to more than 50 MB but with a higher latency around 20–50 ns. In rare cases, a L4 made from embedded DRAM can be present. We now have the complete modern memory hierarchy as shown in Figure 1.11. Three caches, one or more external DRAM chips and a, possibly remote, permanent mass storage. All these memories end up eating most of the available area. Figure 1.12a shows a 130 nm Intel Pentium M from 2005 where memory represents more than 60 % of the chip area, so this tendency is already decades old. A more recent processor (Figure 1.12b), a 2015 22 nm Intel Haswell shows a similar area distribution

Table 1.2.: Main memories key parameters. Data is from [23, 24].

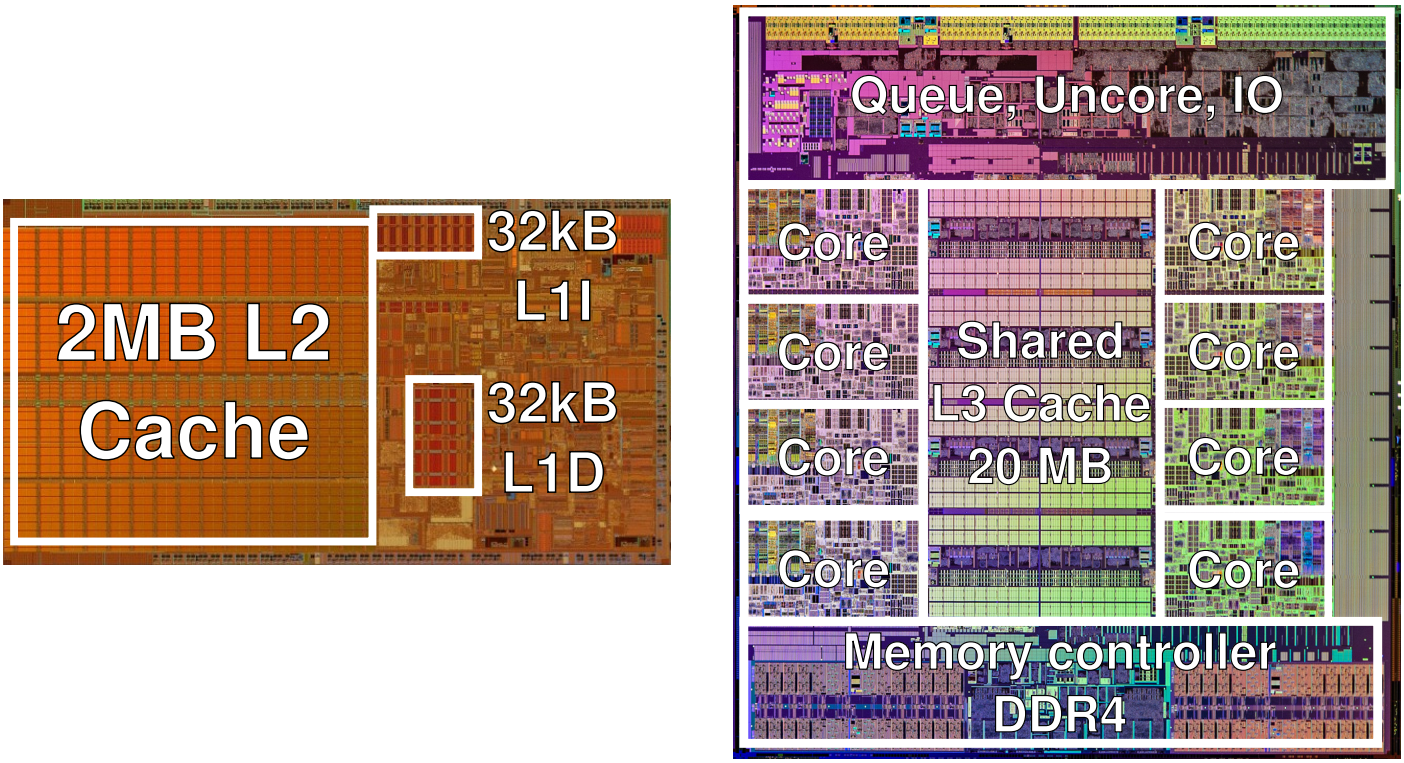
	Price ($/GiB)	Density	Latency	Bandwidth	Persistence	Largest size
SRAM	5000	120 F^2 or 2 Gbit/cm^2	1–50 ns	1 TiB/s	10 µs (Power off)	10–100 MiB
DRAM	20	8 F^2 or 25 Gbit/cm^2	20–400 ns	10–100 GiB/s	64 ms	100–1000 GiB
Flash	4	<1 F^2 or >100 Gbit/cm^2	1–10 µs	1 GiB/s	10–20 yr	10–100 TiB
HDD	0.1	100 Gbit/cm^2	5–20 ms	100 MiB/s	10–100 yr	10–100 TiB
Tape	0.01	49 Gbit/cm^2	1–100 s	300 MiB/s	100+ yr	100–1000 TiB

including complex DRAM controller taking the same surface as 2 or 3 cores. That is why new emerging non volatile memories with much better integration and higher density can be of great help. A summary of main memories parameters is presented in Table 1.2.

1.2.2. Emerging non volatile memories

Emerging Non Volatile Memories (NVMs) are a group of recent (namely 2010 and later) memory technologies that offer promising performances, density and scalability. From an electrical point of view, they all share the same characteristics. In previous memories such as SRAM, DRAM or NAND Flash, the physical property used to store data is the charge of the bitcell. These charges are maintained through power supply and are gone when the power is shut down (except for NAND Flash). In the case of resistive memories, the physical property used to retain data is the resistive state of the bitcell. This resistance changes depending on the current that flows through the bitcell, but the underlying phenomenon depends on the technology. These emerging NVMs provide a huge benefit compared to SRAM and DRAM especially, because it eliminates the need to have several of current levels of memory in the hierarchy. As such, it would make a big leap forward if it would allow to suppress the L3 cache, the DRAM and also the main storage (either spinning HDD or SSD). The introduction of NVMs would thus potentially replace 3 levels of the memory hierarchy into only one, leveraging huge gains in power consumption, timing (latency and bandwidth), density and silicon area. Another possible use is as Storage Class Memory (SCM), which is a class of intermediate memory between DRAM and NAND Flash, in terms of latency, bandwidth and energy.

The gains in power consumption must however be tempered. As of today, reading these NVMs may be cheaper than reading DRAM, but the write operation can be extremely costly depending on the considered technology. There is no static power compared to SRAM nor dynamic idle power compared to DRAM which make these memories more energy efficient. But if they are to replace SRAM, as of today, it would increase power consumption for this specific use with high bandwidth requirement. The gains in latency are also to be nuanced due to the write asymmetry where the write operation can take up to 10 times longer than the read operation which is problematic



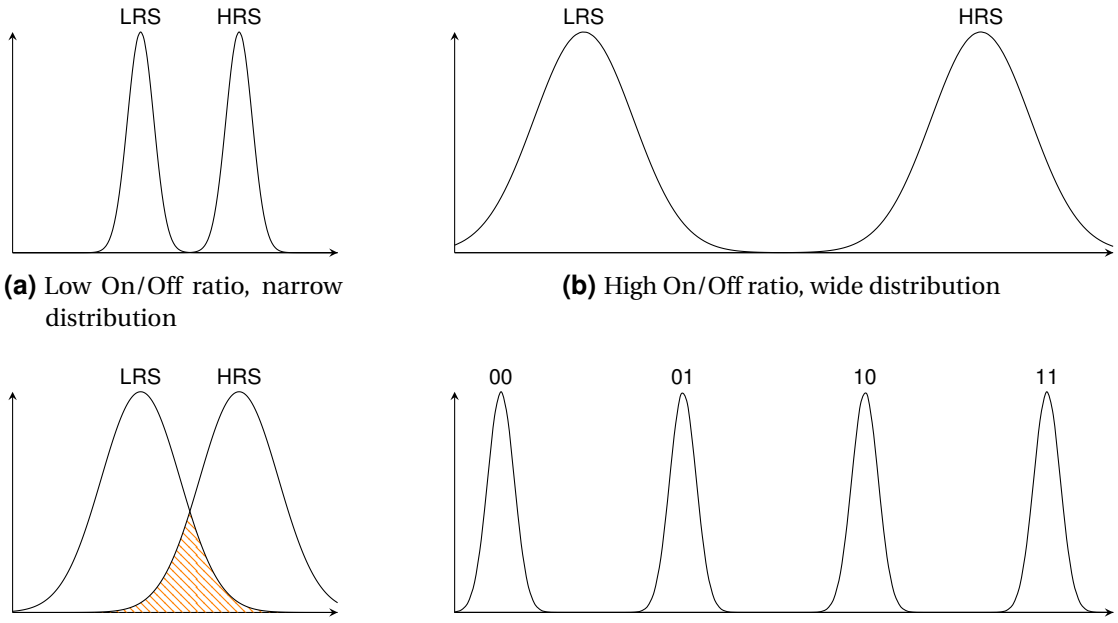
(a) A 130 nm Intel Pentium M die

(b) A 22 nm Intel Haswell die. Each core has a 256 kB L2 cache and 2×32 kB L1 caches

Figure 1.12.: Die photographs

in a system point of view. To ensure system responsiveness and guarantee performances in all use cases, write asymmetry still needs to find workarounds. However, as these are non volatiles, it suppresses the refresh operation that can hinder the access timing on DRAM. Another issue is that timing operations are usually better than at least DRAM, but not of SRAM.

As said earlier, the resistance is the physical property used to store data. We call Low Resistive State (LRS) the logical `0` and High Resistive State (HRS) the logical `1`. The ratio between HRS and LRS is called the On/Off ratio and determines the precision of the SA, the working frequency and if multilevel cells can be used. Unfortunately, contrary to electrical charge, resistance cannot be controlled accurately and follows a normal or log-normal distribution as shown in Figure 1.13. A narrow distribution with a high On/Off ratio is the best case as both state can clearly and easily be distinguished and may even allow multilevel cell (Figure 1.13d). The worst case is a low ratio with a wide distribution where some LRS cells might have a higher resistance than some HRS cells (Figure 1.13c). In this case, either error correcting code can be used but this requires more space and may fail if the distributions are really bad, or write verify loop to make sure the cells end up in a distinguishable state but this is non deterministic and write may take a long time. Low ratio with narrow distribution (Figure 1.13a) and high ratio



(a) Low On/Off ratio, narrow distribution

(b) High On/Off ratio, wide distribution

(c) Low On/Off ratio, wide distribution

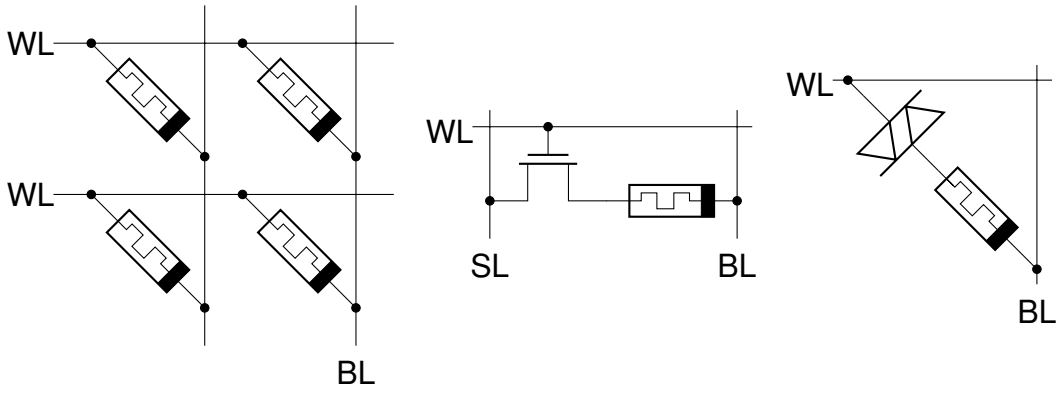
(d) High On/Off ratio, narrow distribution and multilevel cell

Figure 1.13.: Different RRAM resistance probability distribution. Orange hatched ( ) denotes state intersection and should be avoided at all cost.

with wide distribution (Figure 1.13b) are acceptable cases if the distributions do not overlap.

In terms of density, these can reach the theoretical maximum of $4\,F^2$, but it depends on the array structure and the access device to the bitcell: none (crossbar structure), transistor (1T1R bitcell) or back-end of line selector (1S1R) as shown in Figure 1.14. 3D technologies can enable even higher density like Flash already offers. As technology will mature, denser designs will ensue. For silicon area, as we can *theoretically* remove the DRAM and the mass storage (whether HDD or SSD), this removes 2 external chips from the system allowing more compact and efficient systems to be produced. With the advance of In-Memory Computing (IMC) and 3D stacking, we can even dream of a all in one chip where memory and CPU are tightly coupled [25].

There are still some work to be carried at hardware level including technology and architecture, but on software side as well. New data structures can benefit from the non volatility and Operating System (OS) needs to take it into account. Indeed, non volatility ensures that data remains even after power off, nonetheless this also cause some security threats as data will remain permanently which can include sensitive data such as passwords. OS must take care of erasing data after deallocation which was easier with DRAM. On the technology side, endurance for all these emerging memory technologies remains a serious concern that prevent any to be used for their purposed introduction. On the other hand, their integration and compatibility with the fabrication process, depending on the material used for some memories, greatly



(a) Standalone bitcell in a crossbar structure **(b)** 1T1R **(c)** 1S1R

Figure 1.14.: Circuit diagrams of 3 different bitcell types

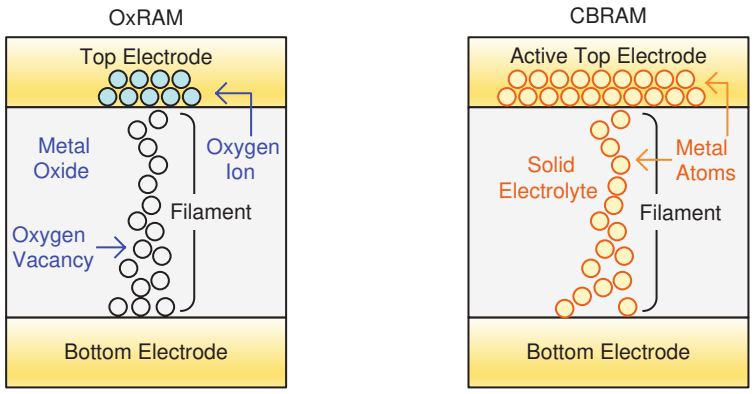
ease their adoption by industry and reduce the need for investment in new fabrication lines.

1.2.2.1. RRAM

Resistive Random Access Memory (RRAM)[1] is the first discovered and manufactured type of emerging non volatile memory dating back to the 1960s but it only attracted attention in the 2000s when it was made with back-end of line compatible materials. Although in its general form RRAM embraces all resistive memories including PCM and MRAM, we discuss in this section only about Oxide Random Access Memory (OxRAM) and Conductive Bridge Random Access Memory (CBRAM). In the literature, RRAM are sometimes referred as filamentous RRAM. Indeed, these technologies rely on a Conductive Filament (CF) inside an insulating material. OxRAM depends on oxygen vacancies as filament while CBRAM uses metal ions (Figure 1.15). *Set* operation is performed by applying a positive voltage between the top and bottom electrodes, whereas *reset* requires a negative voltage. This means that the selector cannot be a one way device such as a diode and also slightly complicates write drivers to be reversible. Access device can thus be a single transistor, an Ovonic Threshold Switch (OTS) or none at all in a crossbar array structure (Figure 1.14a).

It is often made from HfO_2 which is a high-k dielectric (highly insulating) used in transistor to make smaller grids and thus RRAM is easily integrated in current fabrication lines. With a crossbar array structure, it should be the most dense on-chip memory available, excluding 3D stacking technologies. It is aimed to replace potentially SRAM in higher level cache, typically L3 [27] while L2 and L1 are expected to remain with fast SRAM memory. Nonetheless, there are still challenges to reach these goals with serious reserves on endurance and variability within an array.

[1] ↑RRAM® is a registered trademark in Japan and EU until 20/02/2023 [26]. ReRAM is also encountered in the literature.



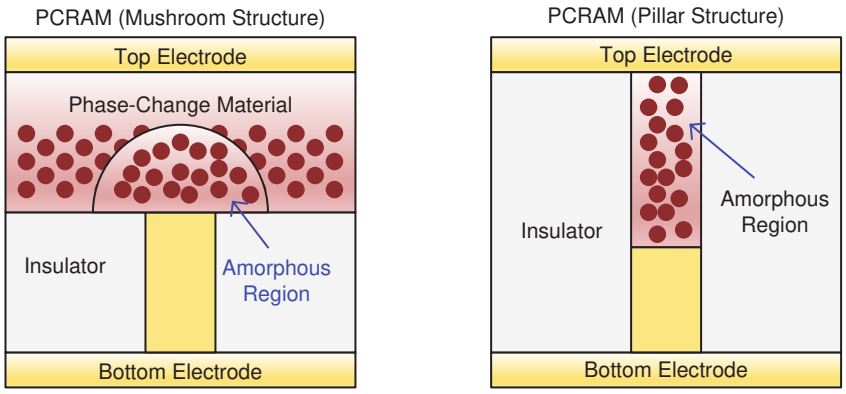
(a) OxRAM **(b)** CBRAM

Figure 1.15.: Different types of RRAM bitcell. From [30]

First of all, RRAM requires higher voltage and current than conventional SRAM to form and reset the CF. It requires bigger transistor to drive enough current (up to 100 µA). Higher current also means it is harder to shrink the pitch of the metal between lines due to IR-drop effect. Cycling between forming and resetting the CF ends up damaging the cell with micro cracks or migrating material (oxygen or metal) cemented up to the point where the cell is stuck in either LRS or HRS. Current technologies have an estimated endurance between 10 million to a billion cycles [28, 29] which is way too low for caches memory or even DRAM where the write bandwidth can be over a billion writes per second. Wear leveling techniques must be used to mitigate these bandwidth and equalize the wearing out on all the bitcells which slows down the working frequency of RRAM.

RRAM has a medium On/Off ratio often combined with wide distribution (intermediate between Figure 1.13b and Figure 1.13c) which makes reading slower to ensure the state of the bitcell. Another problem is the drift associated with the repeated read/write cycle. Both LRS and HRS distributions will shift independently for each cell meaning that some cells will have a worsened ratio while others will see it improves across the lifetime of the cell. Worsened ratio may overlap distribution rendering the cell useless which can be alleviated with wear leveling to move data to extra bitcells. Some array structure such as 2T2R [31] can be used to lessen low uniformity issues.

Overall, RRAM still has a promising future with write currents going down around 1 µA, reading and programming times lower than 10 ns and a retention time of at least 10 years. Endurance above 10^{12} cycles have been reported [32] although it is still a little too low for integrated cache memories. Power density due to higher write currents may also be problematic for some power constrained applications. Highest On/Off ratios are between 100 and 1000 which permits 4 level bitcells (2 bits) [33].



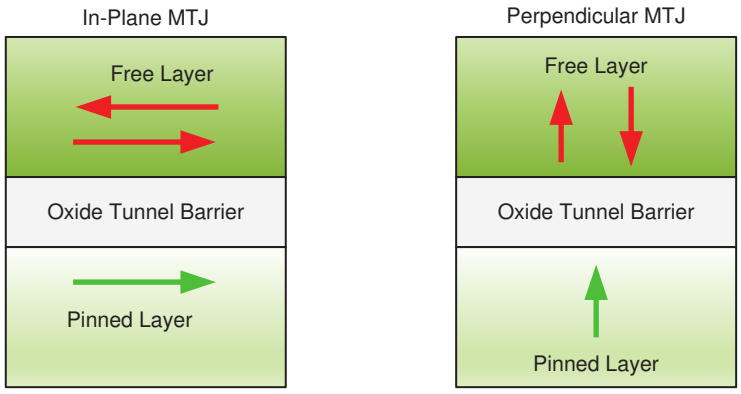
(a) Mushroom PCM bitcell **(b)** Pillar PCM bitcell

Figure 1.16.: Different types of PCM bitcell. From [30]

1.2.2.2. PCM

Phase Change Memory (PCM) is another type of resistive memory relying on the transition between amorphous and crystalline phase of a material, usually a chalcogenide. These two phases have greatly different electrical resistance which is used to store data. HRS corresponds to the amorphous phase, whereas crystalline is LRS. The *reset* operation consists of sending a burst current to melt the material and let it cool down to reach the amorphous phase. *Set* is done by sending a smaller current than *reset* and let the material slowly crystallize. *Set* operation is thus seemingly slower than *reset*. Contrary to RRAM, current is one way only as it is only used to heat the material so the selector can now be a diode which is slightly more compact than a transistor. Unfortunately, the high temperature needed to melt the material requires high current for a short amount of time which makes writing a high power operation. Current used to be over 1 mA and has now decreased to 250 µA with voltage around 3 V [34] similar to RRAM. Moreover, high temperature limits the density to prevent a write to disturb neighbouring cells. Multiple designs coexist such as mushroom or pillar type as shown in Figure 1.16, depending on the materials used.

Similarly to RRAM, PCM is subject to endurance issues that are even worse due to thermal expansion. It either creates voids in the cell until it gets stuck open or, due to melting repeatedly, have material migrating and forming a permanent conductive wire. Current technology has endurance between 1 million and a billion cycles [34–36], which is better than most recent SLC NAND Flash. This is enough to replace the former in fast permanent storage as SSD. Although heating and cooling down the material takes time, it is still faster than Flash with write timings of 100 ns and even less reported [34]. Given its better performance compared to Flash, PCM was the first NVM sold in consumer electronics by Micron and Intel under the Optane brand name with their 3D XPoint technology. Its On/Off ratio is also way better than RRAM up to 10^4 allowing multilevel cells to be used with 3 and even 4 bits [37]. Indeed, precise control of current and timing during pulse gives highly repetitive resistance output in



(a) Plane MRAM bitcell **(b)** Perpendicular MRAM bitcell

Figure 1.17.: Different types of MRAM bitcell. From [30]

contrast to RRAM which has very wide resistance distribution.

In perspective, PCM is planned to replace DRAM [35, 36, 38–40] in computing systems if its endurance is high enough. Otherwise, its use as SCM has already began with Intel and Micron 3D XPoint technology. Read timings are in the tens of nanosecond and write in the hundreds of nanoseconds. Write power is of concern due to high drive current which makes it the most energy consuming memory to write a bit with 10 pJ/bit whereas RRAM is around 100 fJ/bit and SRAM is even lower. Density is limited due to thermal constraint but this is partially circumvented with 3D stacking. Retention time is above 10 years thanks to the material stability in both phases. However, resistance drift due to thermal dilatation and cycling can be a problem in the long term for multilevel cells.

1.2.2.3. MRAM

Magnetic Random Access Memory (MRAM), and more specifically Spin Transfer Torque MRAM (STT-MRAM) is yet another kind of resistive memory using the magnetic orientation of a Magnetic Tunnel Junction (MTJ) to store data. A free layer that can take two different orientations and a fixed reference layer separated by an oxide barrier make up the bitcell (Figure 1.17). If the layers have the same direction, the cell is in a LRS and if they have opposite direction, then it is a HRS. *Set* is performed by sending a current pulse in the wanted orientation and *reset* is done by reverting this current pulse. Just like RRAM, write drivers must thus be reversible and this constrains the device selector as well. Contrary to PCM, it does not require a lot of power to switch state with current in the range of 100 µA and write timing inferior to 10 ns [41, 42]. Endurance is the best advantage of MRAMs as it does not suffer from any thermal dilatation or high current going through the cell. Estimated cell endurance are over 10^{12} cycles [30]. Voltage to operate the cell is also lower compared to previous memories and within 1.5 V, there also reducing constraints on transistor size.

Unfortunately, magnetic materials required for the fabrication process are not

Table 1.3.: NVMs parameters. Data collected from [28, 30, 34, 44–46]

	Cell Size	Multibit	Read Time	Write Time	Write Energy (/bit)	Endurance
RRAM	4–12 F^2	2	~10 ns	10–50 ns	0.1–10 pJ	10^6–10^{12}
PCM	4–30 F^2	4	10–60 ns	20–150 ns	10–500 pJ	10^7–10^{10}
MRAM	6–50 F^2	2	2–35 ns	3–50 ns	0.01–1 pJ	10^{12}–10^{15}

compatible with conventional CMOS technology which is still a problem to be solved. Magnetic nature of the bitcell requires the storage to not be in a magnetic environment that may disturb cells' data which can limit the use for some applications. Heat is also a limitation to the use of this memory in non controlled environment as it largely reduces the data retention time. Finally, MRAM yields a low On/Off ratio with a narrow distribution which makes multilevel cell harder to achieve, but not impossible [43]. 3D stacking is still a work in progress [41] and should help improve the relatively low density compared to RRAM or PCM [44]. Small nodes may also be problematic due to magnetic field interference between cells. As such, MRAM is planned to be a medium density memory compared to RRAM but its high speed and low power makes it a suitable candidate to fully replace SRAM in cache memories thanks to its high endurance [42, 44].

Emerging memory technologies have intermediate energy and timing between either SRAM or DRAM and Flash. Non volatility removes static and dynamic power consumption in the bitcell array which greatly improves energy efficiency. They can replace several memories in the system, mainly DRAM as it is the most power consuming one as well as the higher level cache. Non volatility also allow Flash replacement and better system integration. Another possibility is their integration as SCM in between DRAM and NAND Flash. However, they have limited endurance that is too low to consider a full replacement as of today. Having a permanent storage tightly coupled to CPU will cut down power loss over transmission line instead of having multiple chip connected on a system bus. A summary of their characteristic is given in Table 1.3.

1.3. A new computing paradigm

Now that we have introduced old and emerging memory technologies, we need to explain why we need to revisit the standard architecture model to fit the new needs of the industry. As we have seen, current memory technology are not really scalable with permanent storage being done with spinning HDDs and tapes which both have very high latency, low bandwidth and use mechanical parts that are more prone to failure. Although both have seen tremendous improvement for data density, their high latency and low throughput make them unsuitable for future high demanding uses. We first

introduce the rising demand for high throughput data treatment (*big data*) with the use of AI, then we discuss the challenges this trend faces and raises and finally, we introduce a proposed solution that is IMC.

1.3.1. Big Data

Since 1980, data storage has substantially increased and doubles every 40 months [47] which is an exponential growth as shown in Figure 1.18. This trend is still valid as of today but what changed is how this information is treated. What started with high density information, mostly sensor data, is now a sea of low density information which we need to extract the valuable droplets from:

- In finance, data now comes from stock variations but also analysis of political discourses, behavioral analysis, press text analysis to predict the most accurately how the stock will evolve;
- In social networks, text analysis, photo recognition, graph analysis and behavioral analysis all require huge amount of data (and tracking);
- In informatics, database search, insertion and deletion are recurring operation that are lengthy on terabytes dataset;
- In science, it includes: meteorology with the multiplication of sensor data and higher precision with models containing billions of nodes; biology with DNA analysis and pattern matching for protein modifications research; astrophysics where telescopes' data are harvested faster than they are treated leaving huge untreated databases even after telescopes retirement; subatomic physics such as particle accelerator that can generate terabytes of data in a second; medicine that has very wide input dataset to look for correlation between lifestyles and diseases; etc.

All of these are the consequences of the shift from industrial society to information society where data is the new colorless gold. With the IoT, it will be further exacerbated, but thankfully only 2 % of data is stored [48]. These new data as listed previously are of great volume, vary largely in quality and type, are generated quite rapidly and thus demand a fast treatment.

Not only data quantity increases as well as its diversity, algorithms also evolved with more complex access patterns. Graph processing with bunny hopping from node to node are not predictable in their patterns and prevent any form of caching relying on spatial locality. These irregular access patterns increase stress on memory systems. Improvements in image processing led to stride access patterns where only part of data is used in a regular way. But memory must still serve the full data to accommodate caches leading to underutilisation of the ideal bandwidth. Combining this with stencils application such as convolution used in filtering where data is accessed in subpart of the total also decreases temporal locality with nowadays high resolution pictures. All in all, we face an absurd amount of data whose algorithms

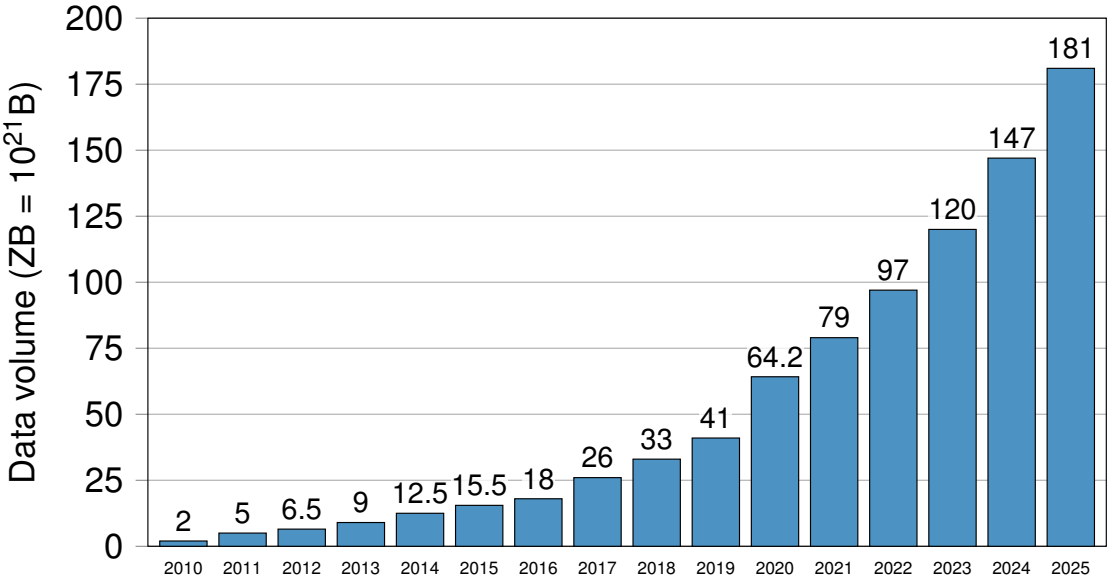


Figure 1.18.: Quantity of data created per year. Only 2 % is stored, rest is treated then thrown away. From [48]

needed to manage them have complex access patterns reducing the effectiveness of cache techniques. This forces data to be read and evicted from caches multiple times increasing the energy and timing cost.

1.3.2. Proposed solution: memory computing

In this data intensive world, we have to treat data in an energy efficient way and with high throughput. Standard computer architecture is compute centric rather than data centric; it is built in order to execute the maximum number of operations in the shortest amount of time but that does not equate to faster data treatment due to memory hierarchy and its intrinsic latencies and limited bandwidth. Data centric architecture is all about treating a massive amount of data in a parallel fashion while compute centric is about controlling the program path and needs the data to be brought through a complex memory hierarchy designed to hinder the slow memory timings.

The only foreseeable solution is to work at the bottleneck, i.e. the memory. As it is the bottleneck, we cannot treat data faster than its external bandwidth and thus this renders all cache levels and lower memories in the hierarchy obsolete. The data should be handled where it resides, at the topmost level of hierarchy that is the permanent storage or eventually the DRAM. A possible parallel is remote working rather than office work where humans are data to be managed. By doing so, we remove the morning and evening commute representing data transfer with its bottlenecks (roads, railways, etc.), its latencies and its energy cost (electrical, gas, etc.). Another solution is also to compute data where it is produced (local consumption) rather than to send it to an external compute unit, whether it is local to the system or a remote server. But

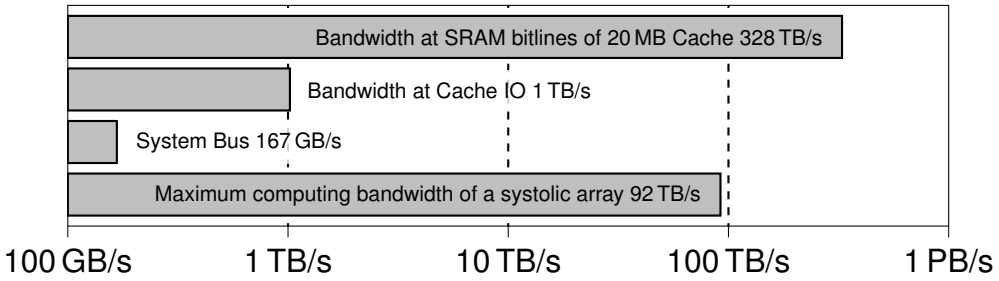


Figure 1.19.: Internal versus external memory bandwidth. From [49]

this is not always applicable.

Why is memory computing a viable and promising solution? First, it removes most of data transfers between the memory and the CPU or GPU. This yields timing and power improvements by removing costly intermediate memories such as cache and also slacken power constraint on the CPU (see dark silicon and heating problems in section 1.1). Second, it takes advantage of the much higher internal bandwidth of the memory, sometimes 100× faster than the external one (Figure 1.19). Not only this can incidentally increase throughput but also reduce the average time of algorithms execution. Third, it uses the full internal width of memory lines which is ranging from 100 up to 1000 larger data width, depending on the considered memory technology, allowing vector computing to occur. Fourth, it can alleviate the energy wall problem by reducing the performance and energy constraints put on the CPU.

How does memory computing work? Multiples classes of solutions have been proposed in the state of the art and we can split them in roughly two groups: analog computing in the bitcell array or digital computing after the SAs. The first group makes extended use of basic electrical rules to compute logical operation such as `NAND`, `OR` or `XOR`. It rely on the array interconnection between bitlines but is also heavily technology dependent. More complex operations are performed through multiple successive logical operations or with some additions to the periphery circuits. Second group adds digital computing units in the periphery but may also use some analog pre-computing performed in the array. Digital additions allow more complex operations such as arithmetic ones (`ADD`, `MUL`, etc.) to be computed in place in single cycle. For a more complete view on IMC techniques, refer to Chapter 2.

Note that memory computing is almost as old as digital computing itself. The very first paper that can be connected to IMC dates back to 1969 and proposes to interleave memory with logic units to compute basic logic functions [50]. It can be considered as a common ancestor to both IMC and FPGAs. A similar proposition is found in [51] from 1970 that introduces compute caches with *search*, *add* and *scale* operations. However, the first chip implementation leaps 20 years forward with an 8 kB SRAM prototype reaching 1.7 GOPS on a discrete cosine transform application used in video compression [52]. Those previous papers were not introducing solution to the memory wall as they do not mention it, nonetheless the memory wall problem has been known for more than 25 years [53]. Already in 1994, the memory was the limiting

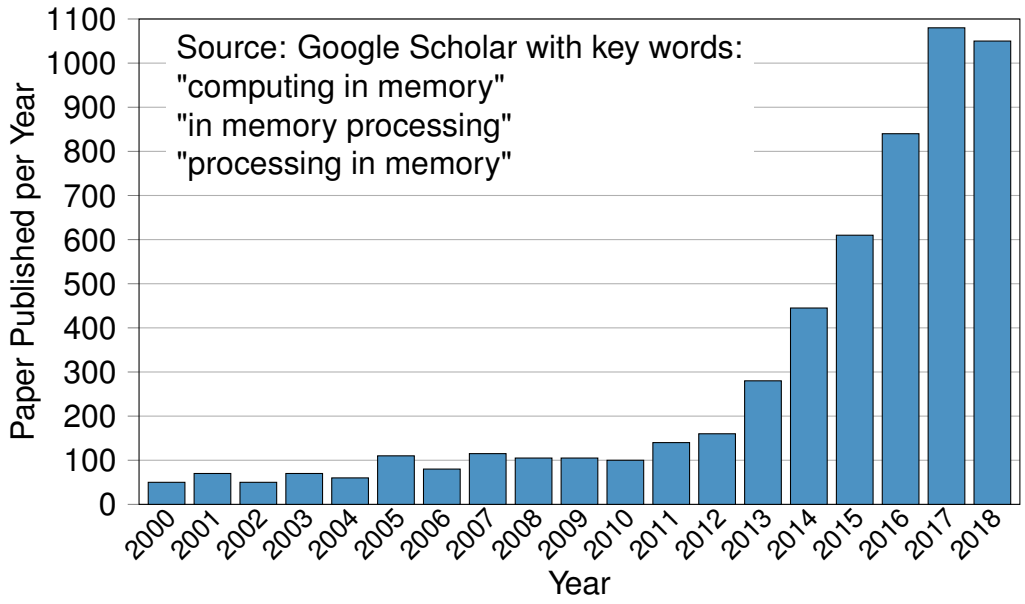


Figure 1.20.: Memory computing research interest in Google Scholar. Search term are not precise enough as they include some result from neurology (*background noise*). From [56]

factor, the von Neumann bottleneck, and the authors only state that scientists and engineers should think outside the box without providing any hint or direction. A first attempt was designed with the Terasys array [54] with 1-bit Arithmetic & Logical Unit (ALU) incorporated into SRAM memory chip for each column. It supports basic logic operations with distributed computing approach and their own high level language. It reached speedup between 5× to 50× versus single CRAY core. Another nail in the coffin is a paper by D. Patterson et al. in 1997 that reports many programs spend most of their time waiting for the memory [55]. It proposes an intelligent RAM that is vector computing tightly coupled to DRAM and foresees up to 4.5× better energy efficiency compared to conventional architecture.

Since 2010, there has been a new surge of papers on IMC or Near-Memory Computing (NMC) indicating a real interest in this solution in both academics and industry (Figure 1.20). Some questions are still open in the community, concerning mainly more software points than hardware. The programming of these new IMC devices and their associated Instruction Set Architecture (ISA) is for now not debated with everyone using homemade ISAs. RISC-V may be a starting point but it is a scalar ISA rather than a vector one (see Chapter 3 for our own). How the instructions are sent to the device or how it is synchronized with the rest of the system is also an unanswered question (see Chapter 4 for our approach). Coherency issues are almost nonexistent in the literature although they do exist as in every unified memory system. Overall it is how it should be integrated in a real system with a software Application Programming Interface (API) that also has hardware implications.

1.4. Conclusion

We have shown that the global context in the semiconductor industry is close to reach a dead end. Moore's law is coming to an end in the next few years after more than 50 years of continuous growth (Figure 1.1). Dennard's law has been broken since 2005 which led to dark silicon, i.e. part of circuit that cannot be powered due to power constraints and heat dissipation issues. Frequency scaling has also reached a plateau around 2005, there also putting a cap on performance for single cores (Figure 1.4a). Architectural improvements were the key to performance increase from 2005 to nowadays with the introduction of multicores CPU. Wider SIMD extension also provided some boosts to treat vectors of data which is particularly useful for the more demanding applications of today such as neural networks. To speed these applications even more and to improve their energy efficiency, general purpose GPUs were introduced with thousands of processing elements executing the same instructions simultaneously. FPGAs are now common in commercial datacenter but it is more a marketing strategy than a definitive solution. Application specific accelerators were also designed such as Google's Tensor Processing Unit (TPU). Finally, the rising costs of more advanced nodes make new chips more expensive per million gates. The technological investments required to fabricate the sub 10 nm nodes are astronomical and leave very few foundries choice (Figure 1.7) which also increases prices. All in all, semiconductor industry is at the end of an era after pushing everything it could to the extreme, both technologically (smaller nodes, better materials) and architecturally (SIMD, multicores, etc.).

All of the previous points were only tackling the problem at the compute or logic level but never at the memory level while the memory is the limiting factor (Figure 1.5) either in bandwidth (memory wall or von Neumann bottleneck) and in efficiency as data transfer is costly. Recent new memory technologies such as HBM and HBM2 partly reduced the gap between GPU and memory performance, but this gap still remains. In summary, we do not need to change how we treat the data but *where* we treat it. Thus, we need a shift from compute centric to data centric architectures where data is the focus point rather than the processing element, especially in the era of big data and AI. This shift should improve energy efficiency in the context of better energy efficient systems to meet the requirements of the Paris Agreements, while still increasing performance with a limited energy budget.

Memory computing is a promising solution to these problems. It satisfies the energy efficiency part by removing useless caches and memories in the system. It offers better performance in vector computing by profiting from the internal memory bandwidth. Emerging NVM technologies offer promising performances such as better read/write energies and faster timings. Their compatibility with CMOS process grants better integration and higher density than DRAM. Non volatility paves the way to a unified circuit containing memory for permanent storage and computing, all in a single chip using 3D integration technologies. This is already envisioned in the literature as the next leap forward [25].

2. State of the Art

Comme vous êtes second, vous avez plus que dix-neuf solutions possibles : soit vous passez, soit vous sciez les poutrelles en deux. Sinon, c'est les relances : doublette, jeu carré, jeu de piste, jeu gagnat, jeu boulin, jeu-jeu, joue-jeu, joue-jié, joue-ganou, gana, catakt, tacat, cacatac, cagat-cata et ratacac-mic.

— Perceval IN *Kaamelott* BOOK III, EPISODE 50, « *Perceval chante Sloubi* »

Claims that cannot be tested, assertions immune to disproof are veridically worthless, whatever value they may have in inspiring us or exciting our sense of wonder.

— Carl Sagan

Now that we have presented the global context in Chapter 1 along with the In-Memory Computing (IMC) concept, we focus here on the state of the art of various IMC solutions. IMC concept has been widely studied in the last years, since it is a promising solution to overcome the von Neumann bottleneck. Indeed, this concept is highly efficient for vectorizable kernels with a massive amount of data. It is also compliant with many memory technologies at every levels of the hierarchy (SRAM, DRAM, SCM, etc.). We present numerous implementations of IMC with varying approaches and technologies.

Contents

We first make a brief introduction on the multiple terms met in the state of the art and explain their differences in terms of hardware implementation (section 2.1). Then, we present the state of the art starting from the bottom of memory hierarchy (closest to the processing units) and going up to slower but higher capacity memories (section 2.2).

2.1. Taxonomy

State of the art is filled with different names regrouping several categories or class of what can be called In-Memory Computing (IMC). We can find the terms of IMC and Computing In-Memory (CIM) that were the first to appear and followed by other terms such as Near-Memory Computing (NMC), Near-Memory Processing (NMP) or Processing In Memory (PIM) which can sometimes denote the same or different technics. Other terms such as Logic In Memory (LIM) can also be encountered. The definition depends on the field of work of the authors that can be categorized in two groups: hardware point of view and system point of view. The hardware point of view focus more on the technological side and on how memory computing is performed at the circuit level. The latter system point of view is centered on the data movement at the global system level, i.e. does the data leave the memory chip or not. An excerpt of what is found in the literature is shown in Figure 2.1. It shows different level of hardware implementation for IMC and some limitations to operator choice due to complexity for some operators.

IMC relies on analog computing and multiple wordlines activation. It is limited in bitwise logic operations and analog multiplication using current summation. This requires some periphery modification to allow multiple activations in the row decoder. NMC is done by computing once the data is directly out of the bitcell array and is, contrary to IMC, completely done in the digital domain. Digital allows more complex operators and removes any disturbance from noise in the computation. The more complex operators include basic arithmetic operators such as addition or multiplication which requires passing information from different bit positions to compute the result. One step further is PIM. It gets data further from its original place and out of the memory chip or subcircuits but stays on the same chip or device. It

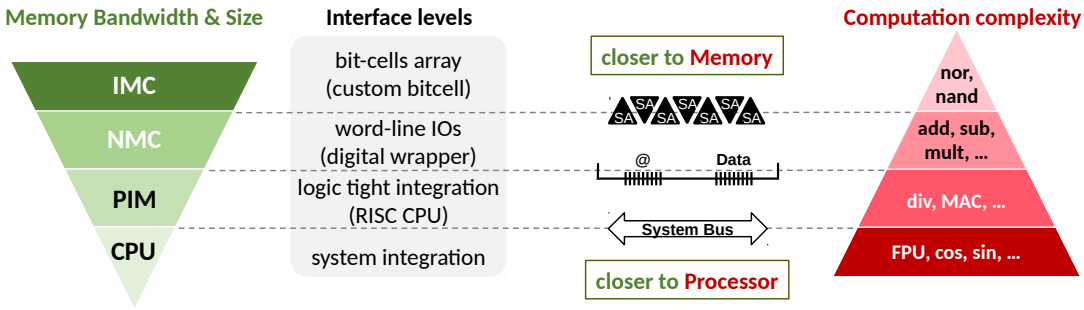


Figure 2.1.: Taxonomy. From [57]

can be viewed as an accelerator tightly coupled to memory or with its own memory depending on the implementation and the system conception. Note the contrast between *computing* and *processing* where the former is just performing operation while the latter may also *control* and advance the program path. The main metrics encountered in the state of the art include:

- **speedup**, i.e. how much faster is the execution of a program or application using the proposed IMC family based solution versus a baseline that is often a scalar or Single Instruction Multiple Data (SIMD) Central Processing Unit (CPU);
- **energy reduction** or how much energy was saved using the same comparison as for speedup;
- **energy efficiency**, that is how many operations can be executed or computed with a given amount of energy or power. It is expressed in operation per second per watt (OPS/W) or in joule per operation (J/operation).

Area can also be used as a metric for the circuit modifications or custom bitcells.

2.2. Memory computing

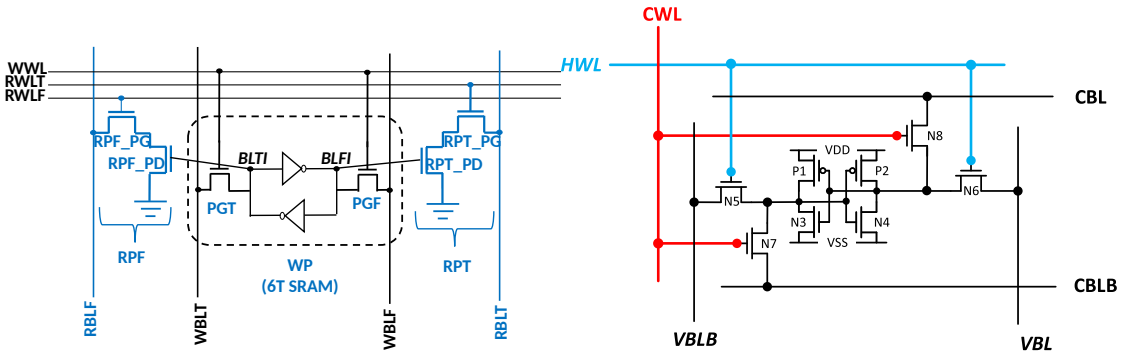
2.2.1. SRAM

Static Random Access Memories (SRAMs) are suitable for both IMC and NMC thanks to their versatility. They can be modified in multiple ways to extend memory functionality with computing. Nonetheless, bitcell modification can be costly in development and validation time as well as on silicon area.

2.2.1.1. Modified bitcell

To compute directly in place in SRAM, it is required to modify the standard 6T bitcell. Indeed, multiple row activations will have cells disturb each other as there is no isolation between them. Multiple designs were proposed in the literature as shown in Figure 2.2. Modified or non standard bitcells are all the cells that are not the conventional 6T bitcell represented in Figure 1.8. Modified bitcells provide more flexibility and prevent read disturb when accessing multiple rows. However, this comes at the cost of higher area footprint as well as design validation cost. We also make distinction between simulation based papers and silicon proven circuits.

Simulation Using 9T bitcell to solve read-write-disturb problem, [60] implements basic logic operators, AND, OR and their complements in the bitcell array. With the addition of multiplexers, they also implement addition and subtraction in respectively 1 and 2 cycles. Moreover, they develop their own Instruction Set Architecture (ISA) to execute instructions. They reach an energy improvement up to 3.0× versus a RISC core baseline with a single SRAM on general purpose applications which is



(a) 10T bitcell from [58] **(b)** 8T bitcell from [59]

Figure 2.2.: Non standard SRAM bitcells used to implement IMC

the main strength of their approach as it is not limited to basic logic operators, but still lack multiplication. Moving on to more complex operators such as multiplier, [61] performs matrix vector multiplication with 8T bitcell. The accumulation is done analogically and reading the result requires multibit Analog Digital Converter (ADC). They achieve a performance of 2.43 TOPS and an efficiency of 16.94 TOPS/W with 6-bit signed multiplication. Other works with 8T bitcells combine Ternary Content Addressable Memory (TCAM) and some shift operations within the array [62]. Some works add up to 3 read ports to perform more efficient search operation and boolean operations with an efficiency of 13.2 fJ/bit [63]. Finally, [64] proposes two different 8T bitcells so that even the use of Sense Amplifiers (SAs) can be removed when computing boolean operations. They achieve an energy efficiency of 11.2 fJ/bit with a 1 ns latency. Specialization for neural network applications using bit-serial computing is implemented in [65]. Although bit-serial is inherently slower, they apply it in a massively parallel fashion allowing timing improvement of up to 18.3× with a 2× energy reduction versus a multicore CPU.

Experimental In [66], a 10T bitcell is used to compute various cryptographic algorithms which all make an extensive use of the XOR operation. The remaining computations are performed by additional circuitry beyond the SAs. The chip fabricated in 40 nm CMOS Bulk with an ARM Cortex M0 CPU delivers a 6.8× speedup and 12.8× energy improvement against scalar baseline. However, their baseline should have been with a SIMD CPU as their current comparison does not account for the kernel vectorization itself. [67] proposes a 12T bitcell to compute analog XNOR and accumulate for binary and ternary neural network applications. The 65 nm chip reaches a 2.48 fJ per ternary operation with 98.3 % accuracy on MNIST. The main issue resides in the area taken by the 12T bitcell that is almost 40 times bigger than off-the-shelf 6T and the use of Flash ADCs also introduces area overhead. With some architectural concerns, [49, 68] uses bit-serial computation where all the elements of a vector are computed bit per bit. It is one of the few (if not the only one) that uses floating point operations which are more hardware expensive but offer way more algorithmic possi-

bilities. With a 8T transposable bitcell (Figure 2.2b) in 28 nm, they achieve 1.4 GFLOPS although some cycles numbers to compute floating point operations seem sketchy. Modified bitcell chips represents a good portion of the state of the art with often targeted applications. Neural network applications are the major part of it with face recognition with 5T bitcell [69], inference accelerator [70] even with 3D bitcells [71] or convolutional neural network accelerator [72]. Clustering algorithms that are part of Artificial Intelligence (AI) applications also have their own work [73].

2.2.1.2. Foundry pushed-rule bitcell

However, non standard bitcells have lower density versus conventional 6T SRAM bitcell. 8T or 10T bitcells are designed using logic rules which lower furthermore the density as these designs were not validated through fabrications and cannot be pushed too far without risking malfunctioning circuits. Standard 6T bitcell offers better integration and density. It is also present in founders design kit with pushed rule, i.e. maximum density and validated designs.

Simulation In [74], although they use 6T bitcells, they also bundle them in local groups with local bitlines connected to global bitlines through a transistor. This is made in order to remove disturbance when performing operation that includes boolean, copy, addition and shift operations. Addition and shift are performed outside the bitcell array as they require inter bitline data exchange. They achieve a 1.2 GHz on 64 bits operand with an energy efficiency of 2.61 fJ/bit which is better than previous state of the art. 3D technologies are studied in [75] with a standard 6T but with added connections to better benefit from the 3D monolithic integration. On boolean operations, they claim a 6.5× energy reduction. Finally, with architectural considerations, [59] proposes to incorporate IMC, in the form of *bitline computing*, within the caches of a multicore CPU. Multiple problems raised by this design are addressed including coherency, consistency, alignment issues and data placement. They adopt NMC for when data is misaligned or not in the same cache level which duplicates computing units. But by computing only in the caches, they obtain small benefits from it with only 1.9× better performance and 2.4× reduced energy as over half of the energy consumption is located in the Dynamic Random Access Memory (DRAM) system [76].

Experimental The use of Content Addressable Memories (CAMs) often requires complex 10T or 12T bitcells but [77] manages to realize it with 6T bitcells that enables logic in memory with AND or NOR operations. This approach yields a 50 % to 80 % area gain with a low energy search energy at 0.6 fJ/bit in 28 nm, 40 % lower than best state of the art solution. Almost standard 6T is used in [78] where a second word line is added to multiplex access mode between different neural layers in the same SRAM macro. They apply binary product-sum for fully connected layers and reach 55.8 TOPS/W in 65 nm chip. However, the chip is very limited in its capacity as it is heavily algorithm dependent. The same split wordline bitcell is used by [79] for XNOR operation in BNN but they focus more on batch normalization with process variations in mind.

They obtain a 3.33 fJ per operation but the same limitation as before applies. More specialized design is proposed in [80] for classifier applications. These applications usually require lot of computing to make a decision for each point but the proposed design attains a 113× energy reduction versus CPU approach with only 630 pJ per decision in 130 nm. Other specialized designs include a reconfigurable in-memory neural network implemented in 65 nm with 2.3 TOPS/W efficiency [81]. Another classifier with a different algorithm lower EDP by 6.8× for traffic sign recognition [82] while a CIM macro for AI edge chips with analog MAC achieves 9.45× better efficiency in 28 nm [83].

Despite showing impressive results at cache or accelerator level, these papers do not address the data movements through the memory hierarchy. The considered architectures are often a reduced memory hierarchy close to embedded systems or stand-alone accelerators. The only paper with architectural consideration is [59] and as expected by computing in the caches, their gains are quite low.

2.2.2. DRAM

Moving computing into the cache hierarchy or SRAM memories is the first in-memory computing step. The second step is to compute inside the DRAM main memory which is often called PIM for this technology. For DRAM, read being destructive, IMC requires row cloning [84] for more efficient computing.

Simulation A Graphic Processing Unit (GPU) comparison in [85] leverages the benefit from 3D die stacking in DRAM technologies. Based on the Hybrid Memory Cube (HMC) technology, they perform vector computation in DRAM intended for GPUs. At 22 nm node, they expect a 27 % performance loss but a 76 % EDP improvement, while granting a 7 % performance boost at 16 nm and 85 % EDP reduction. This is one of the early work in PIM with the advance of 3D memory technologies such as HMC and High Bandwidth Memory (HBM). A system study has been performed in [86] for bitwise operations in DRAM memory. Using triple row activation and a dual-contact cell (Figure 2.3), AND, OR & NOT operations are computed inside DRAM. By doing so directly in the bitcell array, they benefit from the full internal bandwidth. The reported performances reach 32× speedup and 35× energy savings on database applications but considering that data are already inside the DRAM and not coming from another external memory. Another extensive system wide study shows that data movement can represent up to 80 % of the overall system energy [87] on everyday tasks such as web browsing and video rendering. Their key observation is that moving data costs significantly more than computing but instead of tackling data movement at the source, they slightly reduce it by using PIM cores inside 3D stacked DRAMs, which remains the second biggest energy consumer in the system after the CPU [88]. For some very specialized tasks such as video encoding and decoding, they also introduce PIM accelerators. Nonetheless, they achieve a 55.4 % system energy reduction and 54.2 % speedup averaged on their consumer benchmarks. This study proves that

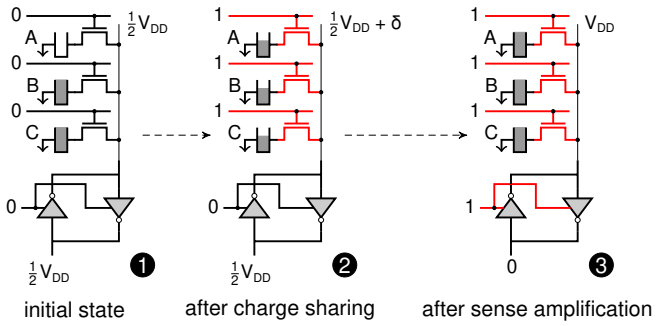


Figure 2.3.: DRAM charge sharing using triple row activation. From [86]

even for consumer devices such as laptops and mobile phones, PIM enhances their performances and battery lifetime. Other works include algorithm for this new kind of architecture [89, 90], AI accelerator using only MAC units enabling 54× speedup versus a GPU architecture [91], graph processing accelerator with dual row activation granting 83× speedup and 59× energy efficiency improvement over GPU [92]. Stochastic computing, a stream based numeric representation, is developed in [93] with 3.8× performance improvement over GPU baseline with binary arithmetic on neural network applications.

Experimental There are much less literature on DRAM PIM chips due to the complexity of a real DRAM system and a fabrication process that is different from CMOS chips. We have found only 3 papers with chip implementation of DRAM IMC or PIM. Commercial solutions combining DRAM with a scalar RISC CPU are already available such as [94, 95]. Available performance reports in [90] show a speedup of 25× compared to a 40 cores CPU with 64 GB of DRAM. Compared to GPU acceleration, this still yields a 5× speed up on DNA pattern matching application [90]. Note that this does not make use of any form of memory computing as the data is moved outside the memory array and circuitry to a small CPU with its own work and instruction memory. To circumvent technological issues with charge sharing that makes DRAM industrial reluctant to IMC, [96] employs a scheme of timing violations to activate simultaneously multiple rows on off-the-shelf DRAM modules. As such, it is the first work to demonstrate IMC with unmodified modules. However, they require the duplication of data as they cannot compute the NOT operation. Each operation also requires a full DRAM command which takes at least 18 cycles. Using bit-serial over 64K bits, they achieve 182 GOPS for COPY but only 19 GOPS for AND/OR and 2.46 GOPS for 8-bit addition as addition takes up more than 10 000 cycles. Nonetheless, they still reach a 9.3× better energy efficiency versus a SIMD CPU. Another point is that their first assumption about reluctant industrials is still there at the end because violating timings is also a constraint that may prevent devices from functioning normally. Finally, [97] uses analog IMC in embedded DRAM. They repurpose part of the DRAM into a compute memory while the rest of it is used to store data needed for computation. They apply it to neural network application with dot-product, averaging, pooling, and

rectified linear unit operation. Some bits are used to control multiplexers when a bit overflows into the next one and they add extra bitcells for averaging. A 16 kB chip in 65 nm is demonstrated and attains 4.71 GOPS and 4.76 TOPS/W which is on par with digital SRAM IMC implementation. Commercial solution has been announced by Samsung in 2021 but has not been released yet with claims of doubling system performance while reducing its energy consumption by 60 % to 70 % [98, 99].

Yet, DRAM is one of the biggest energy consumer in data centers and consumer devices, being around 20 % [88] of total energy consumption. When accounting for all data transfers between nodes, it can represent up to 80 % of the global system energy [87]. Computing inside this technology, although showing great performances which are in average from 10× to 50× speedup or energy efficiency improvement, only solve partially the von Neumann bottleneck as the data is often not originating from this memory.

2.2.3. NVM and SCM

Next and last step is to go on top of the memory hierarchy by computing directly in the Non Volatile Memories (NVMs), especially those that can be classified as Storage Class Memory (SCM), where massive data storage and large vectors are available.

2.2.3.1. Flash

3D NAND Flash is a particular memory as the voltage is higher than usual and bitcells are stacked on more than 100 planes [22]. As such, it is not convenient to use it for memory computing, but its large capacity prevents off chip data movement when considering large neural networks from hundreds of millions of weights to billions of them.

Simulation One of the first idea to perform NMC within Flash memories is to use the Solid State Drive (SSD) controller to execute part of the program remotely [100]. Indeed the controller is always present to handle wear leveling, garbage collection, eventually ECC and manage bus protocols. Each Flash package is extended with a processor or a coprocessor that improves system energy efficiency by at least 100× on facial recognition, as shown in Figure 2.4. Their evaluation process is however not well described. Using the classic current summation apparatus, [101] implements vector matrix multiplication within 3D NAND Flash and demonstrates that Flash variability is not a concern for IMC implementation. Unfortunately, no performance evaluation is presented. With the same approach of current summation in 3D NAND Flash at massive scale (more than 10 000 bitcells), [102] gets rid of any variability thanks to noise being cancelled at this scale. However, the precision of SAs is not discussed but they apply their work on 64 GB Single Level Cell (SLC) 3D NAND and estimate a performance efficiency of around 40 TOPS/W for 4-bit inputs. [103] develops a similar method with 4-bit ADCs and an efficiency between 2 TOPS/W and 20 TOPS/W

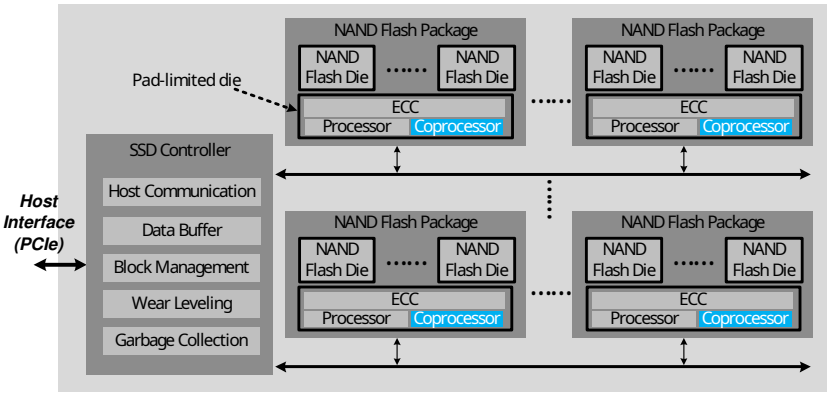


Figure 2.4.: Coprocessor integrated within NAND Flash SSD proposed in [100]

comparable to other works. Using SLC NAND that store complemented data, [104] accelerates vector matrix multiplication for BNN up to 2.56 TOPS/W. They also handle 8-bit input with a performance efficiency of 292 GOPS/W. Finally, [105] implements temporal correlation detection for event based system.

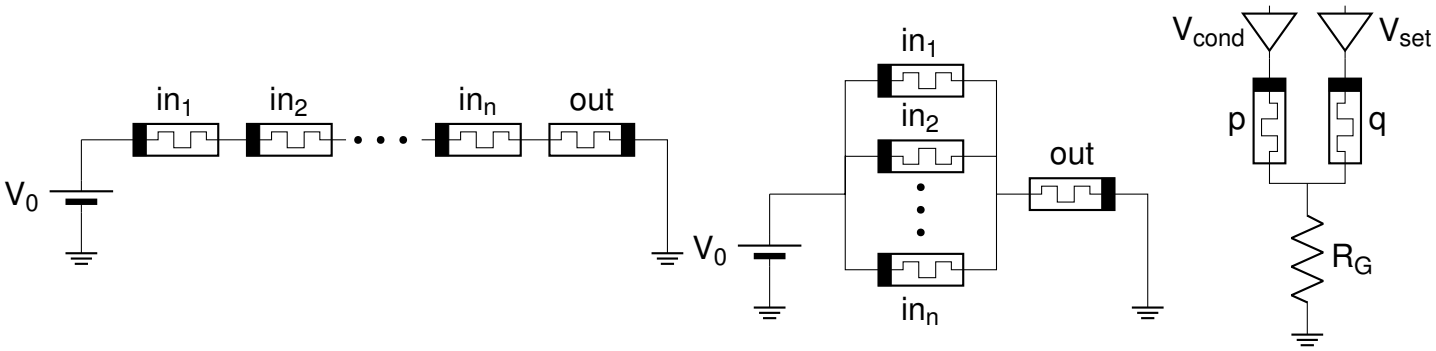
Experimental A prototype of IMC with NAND Flash memory is realized in 65 nm node in [106]. It uses bit serial computation and handles 8-bit weights and inputs. This paper is a prime example of the difficulty of memory computing within NAND Flash as most of it is about technology problems and their resolution. Nonetheless, it achieves an estimated but impressive 129 TOPS/W for a 5×5 convolution on 8-bit data using current summation method, despite a very low density of barely 16 $kbit^2/mm$. Commercial application have again been announced by Samsung in 2020 as SmartSSDs but as of today, no consumer devices have been made public [107].

Most proposed designs are dedicated to solely vector matrix multiplication and only uses current summation as analog mean of compute. Not only this greatly reduces the use for other algorithms, it is also less stable over a long period of time due to Flash variability and wear leveling. Nonetheless, NAND Flash IMC can be interesting for edge devices with pruned weights neural networks.

2.2.3.2. RRAM

Resistive Random Access Memory (RRAM) is probably the best suited memory for IMC. Indeed, it offers memory elements that are disposed in an appropriate manner to create logic functions directly in the bitcell array. For resistive memories, there is no need for non standard bitcell (if "standard" is meaningful for research prototypes) as resistance cannot disturb each other, although repeated read operation may end up disturbing neighbouring cells.

Simulation Demonstration of logic functions with RRAM based bitcells have been made with IMPLY [108] and NOR [109]. NOR logic being functionally complete, it



(a) N inputs NAND gate **(b)** N inputs NOR gate **(c)** IMPLY gate

Figure 2.5.: RRAM boolean gates. **(a)&(b)** When $\mathbf{V_0}$ is applied, **out** stores the result. **(c)** When $\mathbf{V_{cond}}$ and $\mathbf{V_{set}}$ are applied, result of $\mathbf{p} \rightarrow \mathbf{q}$ is stored in **q**. Adapted from [108, 109]

is possible to realize all others logic functions just from this basic operation. It uses parallel RRAM to perform it on N operands which makes it interesting and easy to implement. Both papers from the same authors develop the method to perform these computations and more advanced operations such as 8-bit addition. Unfortunately, those are proof of concept with no benchmarking. Moreover, the computation taking place only in the bitcell array, the result is written in place and wears memory at each operation, reducing the memory lifetime by as much (Figure 2.5). A memory processing unit is presented in [110] where the memory organization is kept standard with arrays inside banks and a chip containing multiple banks. This organization however generates data movements to perform computation between different data not in the same matrix and workarounds are discussed. Yet these data movements are expensive and the overall execution time on vector benchmarks is 1.5× slower than the ideal case. The same memory processing unit is used in [111] and compared against another PIM workflow and show speedup of 35× on image convolution with an energy efficiency improved by 3.4×. The worst case endurance using wear leveling is measured at 288 days, less than a year, which is not acceptable. [112] improves fixed point multiplication within RRAM to reduce area footprint and increase throughput. Logic synthesis to map logic functions within memory is explored in [113].

A programmable logic in memory (PLiM) is realized in [114] based on 3 operands majority function where one of the operand is set to either 0 for AND or 1 for OR. It intends to ease the transition from end-user application to LIM algorithm and demonstrates on PRESENT, a cryptographic application. Using a very optimistic RRAM technology, with 1 ns read/write latency and a write energy of 0.1 fJ, they achieve 5.88 pJ per encryption block (64 bits) and a throughput of 120 kbps. Endurance is not discussed. Implementation of XNOR and XOR functions is done in [115] but reduces the number of memristors and computing steps compared to previous IMPLY and MAGIC works, which in turn reduces the required energy by 56 %. Memory wearing out is not covered. Neural network applications are studied in [116] using XNOR

operation for BNN. They integrate it as accelerator within an ARM CPU and as such gain very little, about 11 % better performances and 7 % better energy efficiency. It is not clear why they choose to integrate their solution within a CPU rather than tackle memory transfers at a higher level. Fitting a whole neural network on a single RRAM array is not always possible and methods to minimize data movements are discussed in [117].

Complex accelerator based on RRAM, such as a Boltzmann machine, is designed in [118]. The accelerator sits aside the DRAM and can solve complex combinatorial optimizations and neural computations. Behind lies a classic matrix vector product engine with 1T1R bitcell and a current summation scheme to perform the computation. It provides fast state update for simulated annealing and neural training by mixing SRAM and RRAM in bitcell array. Simulation on a full scale system with multicore CPU, cache hierarchy and DRAM memory versus single threaded baseline demonstrate high performances gains up to 69× for deep neural training while energy is reduced by 63×. Circuit endurance is estimated to be at least 18 months for neural tasks with low (10^6) RRAM endurance. Overall, this accelerator presents highly interesting features but cannot perform exact (i.e. scientific) computations.

Experimental Brain inspired neuromorphic computing is realized using 64 conductance levels RRAM [119]. This allow to store multibit data in single cell to increase density but also eases neuromorphic computation between neural layers. The programming phase uses hybrid voltage and current pulse to increase network accuracy by 10 %. A non volatile processor targeting Internet of Things (IoT) edge devices is demonstrated in 150 nm CMOS technology [120], featuring non volatile flip flop backed up by 2 memristors, a fully connected neural network with embedded RRAM and non volatile SRAM for shared data between CPU and the neural network. 97 % of the overall latency comes from weights transfer between non volatile SRAM and the neural network while it stands for 62 % of the energy. Adaptive 3 bits SAs are used and null inputs are not computed to save 64 % of energy compared to the state of the art. It yields a 13× more energy efficient design running at 462 GOPS/J. Large scale RRAM CIM prototype with 16 Mbit 1T1R is prototyped in 150 nm node [121]. AND, OR and XOR operations are performed by setting a reference current value at adequate value to mitigate distribution effects in bitcells. Authors achieve a working frequency up to 70 MHz with 512 bits parallel computing. The same authors implement multiply and accumulate operation for BNN in 65 nm technology [122]. High performance neural accelerator is presented in [123]. It uses 2T2R bitcells to reduce IR drop over long bitlines and supports 8-bit inputs. It reaches 78 TOPS/W with 77 µs per inference on hand written digit recognition.

Presented state of the art shows that most works focus on performing only matrix vector multiplication in the RRAM array with analog computing. This prevents the chip from being used in other purposes and is more prone to noise. Basic logic functions are also demonstrated but require lengthy operation combinations for more complex operations such as addition. Moreover, most designs are very limited

in their precision with often single bit precision and never over 8-bit [124].

2.2.3.3. PCM & MRAM

Phase Change Memory (PCM) and Magnetic Random Access Memory (MRAM) have shown less interest for IMC as their foundry processes are not always compatible with conventional CMOS. However, they should have the same proprieties as RRAM to allow IMC implementation with roughly the same methods.

PCM Deep Neural Network (DNN) inference with 2 PCMs working in differential mode is proven in [125]. With taking resistance drift into account, authors maintain a 90 % accuracy. They also develop a method to train a DNN with PCM particularities which allows to keep the training accuracy steady and grants 10 % better accuracy. Finally, they raise several issues that are yet to be solved including thermal variation, noise, low precision and limited endurance that is enough for inference but not for training. Using 2T2R bitcell, high density 1 Mbit TCAM is achieved in 90 nm [126]. This new bitcell is 10 times smaller than usual 8T or more equivalent SRAM bitcell and allows high performance search and matching, up to 500 Msearch/s. [127] improves parallel programming of PCM bitcells using voltage to duration to finely tune conductances. This also allows different activation functions such as *ReLU*, *sigmoid* or *tanh* to be used for inference, reaching more than 97 % accuracy. 8T4R bitcell is used in [128] for matrix vector multiplication. Time-based current ADCs are used to fasten conversion and local digital function unit equalizes the different ADCs output after calibration process. Matrix vector multiplication is performed with 8-bit integers and PWM to take advantage of time based ADCs. The 14 nm chip achieves 10.5 TOPS/W at 1 GHz.

MRAM Basic logic functions using 1T1MTJ bitcell (Spin Transfer Torque MRAM (STT-MRAM)) is achieved by activating 2 wordlines simultaneously in [129]. No performance evaluation is proposed though. Using the same principle, [130] extends it with addition operation by concurrently computing AND and XOR operations in the array. Concerning process variations inherent to resistive memories, they propose to add ECC to mitigate those. A small architectural evaluation with flat memory model (embedded device like) including a developed ISA is proposed. They achieve in average 3.93× faster execution while reducing system energy by 3.83×. Computational RAM is introduced in [131] on the basis that data communication uses 5× more energy than actual computation. Similar to already presented [59], they introduce NMP and *true* IMC with multiple wordline activation. They extend periphery circuitry for more complex operations such as addition. They forecast miraculous 1500× faster execution time and 750× more energy efficient for 10 nm node. Transforming the multibit input layer of BNN into stochastic computing is performed in [132]. It uses XNOR operation in memory, but the popcount is performed with CMOS adder tree. A 2.1× energy reduction is achieved at the cost of an acceptable 1.4 % accuracy loss.

Chips design are quite recent. [133] presents a 22 nm 1 Mbit STT-MRAM circuit that also implements NMC for security applications which is limited to shift and rotate operations but up to 256 bits. They reach 342 MOPS with a very high bandwidth over 40 GB/s. A 128 kbit chip in the same node using 2T2R to store complemented data is reported in [134]. It handles 4 bits data and performs vector inner (scalar) products using current summation. It achieves 5.1 TOPS/W with 90 % accuracy on MNIST.

2.2.4. Other works

Some works are technology agnostics. This permits to take advantage of the similarities between RRAM, PCM and STT-MRAM. The proposed solution [135] only focus on bitwise operations thanks to large alteration of the internal memory organization and varying reference voltage to decide which operation to execute. Reported gains are up to 1.12× and 1.11× for speedup and energy saving respectively, but there again, endurance aspect is not discussed. Similarly, a generalized approach to select which part of an application should run on PIM rather than core is proposed [136]. Authors base their work on the non generic approach of PIM design, i.e. that PIM cannot run all operations of an application and it may not always be faster than conventional CMOS. This can accelerate transition from compute centric to data centric architecture while not having to modify the applications. Claimed gains are 10.9× better energy efficiency and 6.4× speedup compared to multicore CPU. This work is completely technology agnostic and as such, endurance is not discussed. However, the developed procedure seems to be easily extendable to support it for NVMs.

Combination of multiple memories is one way to get the best of both worlds, i.e. to merge advantages of both technologies and reduce some undesirable traits. To reduce data movements throughout the system, [137] adds SCM memories to NAND Flash SSD. Although they do not perform any kind of memory computing, it is also a part of the global solution for the memory wall as wear leveling and Flash memory functioning induces extra reads and writes to memory. With 10 % of Flash capacity, this solution yields a 10× speedup compared to Flash alone. The same idea was also pursued with NAND Flash only in [138] by targeting the best performing blocks rather than using extra memory. Similar idea is developed in [139] to extend NVM cache lifetime by using a small SRAM cache to merge writes. This prolongs NVM lifetime by a factor 100. A completely novel bitcell is presented in [140] which mixes DRAM and SRAM resulting in a 9T1C bitcell. The computing part is based on charge diffusion which is adequate for denoising and filtering images. It achieves 233 TOPS/W at 200 MHz. Another association between SRAM and RRAM allows to perform AND and NOR operations [141]. Additional circuitry is needed for arithmetic operations leading to 93 % improvement in energy efficiency and 6× speedup. NMP with NAND Flash and Field Programmable Gate Array (FPGA) embedding DRAM is performed in [142]. Special attention is paid to how the applications are parallelized as it can differ execution time by a factor 20. Weirdly enough, they do not provide evaluation against CPU baseline.

For architectural studies, several works focus on computing cache, such as [143],

where energy gain is up to 6× and raises up to 7.9× when cache is based on FeFET-RAM. However, when replacing SRAM with FeFET-RAM, the endurance aspect is not discussed. Similarly, [59] transforms the caches into compute units and show performance improvement of 1.9× while observing a 2.4× energy consumption reduction in average. Reconfigurable architecture using SRAM is considered in [144] to meet various requirements either from the application or from the data pattern used during different application phases. The programming model of this reconfigurable architecture is compatible with SIMD programming and shows a 60× EDP reduction compared to a SIMD 512 bits architecture. Nevertheless, the considered architecture is a reduced memory hierarchy close to embedded systems.

Adapting conventional programming models to PIM is one of the obstacles, along with cache coherence protocols and virtual memory. Overcoming those with PIM enabled instructions, a simple ISA extension, [145] tracks data locality to decide whether an operation should be performed in memory. The operation is carried near the DRAM rather than into it directly. Authors achieve 25 % reduced energy and 60 % speedup in the best case. Algorithm specific ISA for graph mining is proposed in [146]. This work intends to greatly enhance memory-bound graph algorithm by working on multiple vertices at once, which grants over 10× speedup versus manycore CPU. An access processor inserted in between main memory and cores serves as a bridge interface for near memory accelerators [147]. Doing so avoid modifying the memory circuits and is thus NMC. No evaluation is performed as it is a work in progress but the processor being in the middle with two buses interfaces, I do not see how this could help with memory wall problem. Streaming architecture is proposed in [148] where an application data path is split into several memory processors designed with NVM technology. Data go from one processor to the next as each processor computes a part of the application. Using convolutional neural network, they achieve up to 235 image/s/W and a speedup of 48× compared to a mobile GPU. However, this solution is close to a FPGA architecture and could face challenges to be introduced in a generic hierarchy.

2.3. Conclusion

All these papers show that IMC can be introduced at every level of the memory hierarchy and in a wide range of memory technology, including emerging NVMs. Moreover, all the proposed solutions exhibit gains in terms of performances and energy reduction compared to standard von Neumann architectures. Yet, IMC still has a long way to go through and face limitations and constraints.

First of all, it often relies on **analog computing** which has its pros and cons. Analog computing can better use the full available bandwidth which is often the full row width modulo muxes and sometime even the whole array. When the complete bitcell array is used, IMC can be viewed as a kind of systolic array [149]. Bandwidth is not reduced by buses interfaces but might be by analog to digital conversion. Maximizing bandwidth is likely to prevent non logic operations to be implemented. Latency is also minimal

as data stay within the array or may eventually need an analog to digital to analog conversion to be written back. It is only limited by the IR drop along bitlines and the required computing precision. Energy is greatly reduced as data are accessed only once and not serialized over buses. On the other hand, instant power is an issue as activating multiple wordlines and all bitlines significantly increases power consumption which is a concern for edge devices. Increased power requires increased power-grid density, leaving less space for other functions. In High Performance Computing (HPC) node, heat dissipation may be an issue in the long term. Analog computing is further limited in the signal to noise ratio as SAs need to differentiate small voltage margins (readout precision) [149, 150]. Moreover, when multiple wordlines are activated, SAs sense the current sum of all bitcells along the bitlines which requires high precision. High precision ADC needs more time to perform the conversion and uses more area and power. These problems are exacerbated when using multilevel bitcells. When considering devices non linearities, circuits non idealities, process variations and noise depending on technology used, it also reduces SNR. Yet, it can likely be a benefit for deep learning applications. In addition to SNR, bitcell stability must be accounted for. When computing takes place, non complementary data may be present on BL/BLb ending up corrupting bitcell data which is called read disturb. Transistor isolation is often used in SRAM with 8T bitcell, but this drastically reduces density and requires the use of non standard bitcell.

Secondly, eligible applications for *raw* IMC are limited. Indeed, bitwise logic operations are easily implemented, and matrix-vector multiplication is as well using analog computing. But this is not enough to satisfy either all applications or simply an application, i.e. its data intensive part, in its entirety. Sebastian et al. [151] gives a list of potential applications:

- **Scientific computing** especially linear algebra computational kernels, typically Matrix Vector Multiplication (MVM). However, IMC is often low precision which limits its use.
- **Signal processing**, optimization and machine learning where approximate solutions may be acceptable. Combinatorial optimization problems with IMC attracted some attention [118].
- **Stochastic computing** and security. As data stays within memory, IMC reduces risk of bus snooping or side channel attack within CPU. Side channel attacks within memory are possible but are of higher complexity due to bulk data treatment [66, 152]. Stochastic computing uses random numbers that can be generated by the stochasticity of switching behaviors in memristive devices.

In the previous applications, not all parts of said applications may use IMC. Complex operations such as division or square root which are *exceptional* in relation to other common operations do happen. Security applications needs special operators hence the use of specialized accelerator nowadays.

Thirdly, system integration with programmability, ISA and proper interface definition is still lacking. DIMM interfaces appear to be the most adapted. Coherence

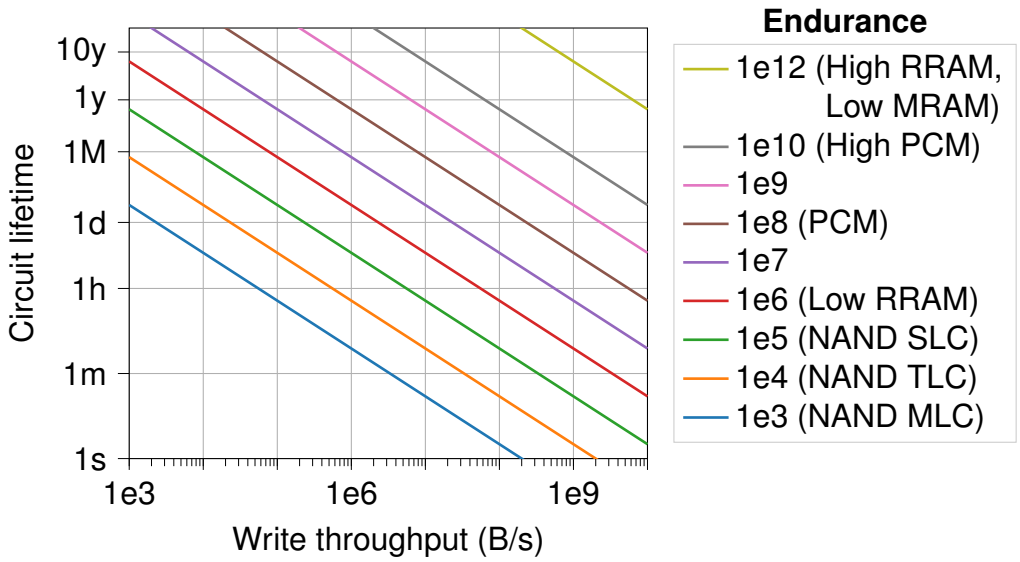


Figure 2.6.: NVM lifetime as a circuit for different endurances of elementary bitcell with a 1 MB technology agnostic NVM function of write throughput

and consistency problems arise from computing in different places. Operand locality and alignment issues are a limiting factor to IMC. In order to perform an operation, data should share the same bitlines and bits should be aligned (i.e. bit 0 of left hand operand should share the same bitline as bit 0 of right hand operand). That means that it is almost impossible to perform an operation on two consecutive words which will likely share the same row with different bitlines. As a consequence, **data movement eliminated by the introduction of IMC is just hidden within the same memory chip now**. Data should share the same array with the same bitlines and different wordlines, but it may be present in different arrays or banks. Furthermore, output address might be distant from input addresses [110].

Finally, another important point when performing IMC is the endurance of the NVM. Indeed, doing all operations directly in the memory array with the result written back there will considerably reduce the NVM lifetime. As shown in Figure 2.6, considering a 1 MB technology agnostic NVM, the lifetime is really limited. We considered a uniform wearing of all the bitcells through wear leveling technique and that the circuit is down when 20 % of the bitcells are dead. Computing directly in the NVM would thus **wear it down in less than a year**. Moreover, the slow speed and high write energy of different NVMs might be acceptable for IoT but not for HPC within data centers. Thus the throughput, energy efficiency and durability of this solution remain, to the best of our knowledge, an unsolved problem. Our main idea is to combine a SRAM enclosed in a digital wrapper with a NVM to get the best of both worlds. As such, we would benefit from the high speed of SRAM and its infinite endurance in addition to the non volatility and very high density from the NVM.

In conclusion, IMC solutions can yield promising improvements in terms of speedup and energy efficiency but at the cost of complex hardware implementation, non standard bitcells, non standard layout or high precision ADCs. Operator precision is at most limited to 8 bits in reported literature [124, 153], which is not enough for most applications. For emerging NVMs, endurance is rarely discussed or application targeted has limited write intensity (e.g. inference). Most works focus on application specific accelerator, mainly neural networks with few cryptographic applications. Nonetheless, there are some general purpose implementations [59, 60]. Few works have performed architectural evaluation [59, 143] and none consider the complete data movement from a permanent storage into the working memory.

That is why we believe a digital wrapper around any NVM will achieve the best performances and save endurance by using SRAM buffer to perform computation and serve as a write buffer. Digital wrapper grants more flexibility and uses conventional CMOS operators which are less restricted in terms of precision. It also offers the genericity required for computing systems and is not subject to analog noise.

3. CSRAM Design

Ou, chante Sloubi. Nous, on va faire que chante Sloubi.

— Perceval IN *Kaamelott* BOOK III, EPISODE 50, « *Perceval chante Sloubi* »

No man ever wetted clay and then left it, as if there would be bricks by chance and fortune.

— Plutarch

As shown in the state of the art, a good part of In-Memory Computing (IMC) designs rely on custom made bitcells for Static Random Access Memory (SRAM), or analog computing irrelevant of the considered technology (RRAM, SRAM, Flash, etc.). In this chapter, we explain the motivations to design a digital wrapper compatible with different types of memory technologies. We specify the constraints and capabilities of the said wrapper that we call C-SRAM. Finally we show that our design methodology provides fast, automated and cheap conception allowing efficient design space exploration. Our workflow provides swiftly testable designs reaching 2.05 TOPS/W on MAC operation or 425 GOPS/mm^2 while limiting area and power overhead to respectively 5 % and 20 % of the standalone SRAM. The obtained results are then used in the following parts of my work to compare IMC architectures against the usual von Neumann architecture.

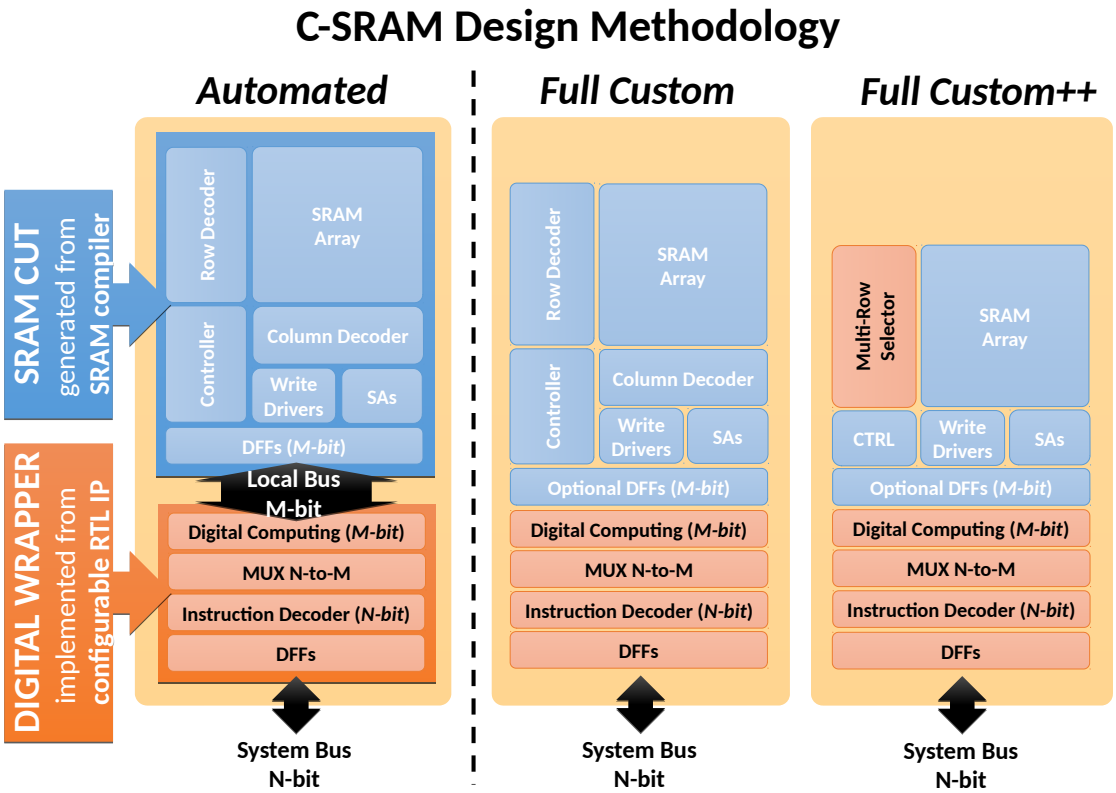
Contents

Chapter 1 introduced the global context and the need for a new computing paradigm that is data centric rather than compute centric: In-Memory Computing (IMC). In Chapter 2, we have presented different IMC implementations and determined that most of them rely on custom designs which breaks the conventional digital conception flow. We explain with more details in section 3.1 the motivations for a digital wrapper rather than an ad hoc solution. Section 3.2 defines the specifications and describes our C-SRAM implementation. Finally, section 3.3 compares our design against state of the art SRAM IMC implementations.

3.1. Motivations for a digital wrapper using standard design flow

Custom memory designs using non standard SRAM bitcell, such as 10T, must be qualified with a high-yield production: specific test circuits (e.g. BIST) need to be developed resulting in major changes to the well-established validation flow. Then, to expand the operating range, several memory sizes need to be qualified, which also requires a major development effort. Finally, to meet Electronic Design Automation (EDA) tool compatibility requirements in a time-limited design phase, automated generation of memory views (HDL, physics, timings and powers, etc.) need also to be developed. Subsequently, it is necessary to easily manage a large number of configurations (capacity, form factor, operations, etc.), while guaranteeing a high-yield production. This technology adjustment can take up to 2 years in advanced nodes such as 5 nm. A workaround is to design a glue logic, also known as a digital wrapper, connected to a memory. This drastically reduces the development time, down to a few weeks, hence reduce the Time To Market (TTM) that is of the utmost importance for industrial companies. Moreover, this digital wrapper can be more easily parameterized, allowing on/off features at design time rather than designing a monolithic circuit not adapted for all targets. This makes the digital wrapper a more modular approach. Some drawbacks of a digital wrapper are that it offers less performance than full-custom solutions in terms of TOPS/W, and uses slightly more area to achieve the same functionalities.

If we wanted to use true analog IMC, i.e. using analog electrical rules to make operations directly inside the bitcell array, we would have been very limited in terms of available operations. Indeed, only NAND and OR (or AND and NOR) are feasible with SRAMs for logic operations; MAC can be done using Kirchoff's law and an Analog Digital Converter (ADC) but the precision is limited to a few bits although the area overhead of the ADC is limited to less than 10 % [67, 72]. Modifying both the array and the periphery to activate multiple wordlines at once would not only have a significant design cost, but would also not be cost-effective in terms of both area and energy. With only two logic operations available, only few transistors are spared for ADD operations, but it is useless for SUB or even MUL operations. So the gains obtained would be annihilated by the periphery overhead, and by the use of several boolean instructions to compute more complex operations, unless the design is constrained for bitwise



Design Methodology	Automated	Full Custom	Full Custom ++
Computing classification	*NMC*	*NMC*	*NMC/IMC*
Computing area overhead	*Medium*	*Low*	*Low*
Form factor flexibility	*High*	*High*	*Low*
Development effort*	*Low*	*High*	*Very high*
Silicon qualification effort	*Low*	*High*	*Very high*

**including BIST*

Figure 3.1.: Proposed design methodology to automate the C-SRAM macro generation with relation to full custom design solutions. Automation simplifies the design phase but makes it more difficult to implement analog or precomputing functions. **Full Custom++** refers to non standard bitcell array such as 10T with multirow selection, **Full Custom** refers to standard bitcell arrays with digital parts (Arithmetic & Logical Unit (ALU), Finite State Machine (FSM), decoding, etc.) directly integrated in the memory cut. **Automated** is the case we are aiming for where SRAM memory and digital wrapper are two completely distinct units.

only operations. Furthermore, inter bitline communication for arithmetic operations is not feasible with this solution.

A digital wrapper has thus way more appeal from a system point of view as it can be easily customized, designed rapidly for quick system evaluation (reduced TTM), offers more operators and simply more design freedom. Besides, it is technology agnostic, i.e. it should be compatible with different Non Volatile Memory (NVM) technologies regardless of their underlying physics. In order to save the currently limited endurance of these emerging NVMs, a digital wrapper should provide its own small SRAM buffer as working memory. By tightly coupling a NVM with our digital wrapper and its SRAM, we achieve Near-Memory Computing (NMC) from a hardware point of view and IMC from a system architect point of view. We call the digital wrapper tightly coupled to SRAM a C-SRAM. As C-SRAM uses standard 6T SRAM bitcell, it is compiler compatible with main foundries, and the rest of the glue logic is based on Register Transfer Level (RTL) Intellectual Properties (IPs). When coupling our C-SRAM with a high capacity Storage Class Memory (SCM), both SRAM and digital wrapper can be implemented in the unused top metal layers of the NVM limiting the area overhead[1].

The main differences between **Full Custom** and **Full Custom++** designs and our **Automated** approach are presented in Figure 3.1. The **Automated** approach is the only one compatible with standard digital design flow that requires little development effort and relies on proven EDA tools. For an easier integration with industrial needs, the **Automated** approach tends more to the plug'n play philosophy which prevents redesigning a full memory array from scratch. We can even incorporate custom made SRAMs implementing IMC to further extend their computing capabilities with digital operators. Our solution also provides a more flexible approach as we can split the SRAM in several cut, either along the vector width or along the height (number of addresses) as shown in Figure 3.2. We believe that this flexible layout combined with the ease of plug'n play of the C-SRAM will offer the best trade-offs versus custom approaches presented in Chapter 2.

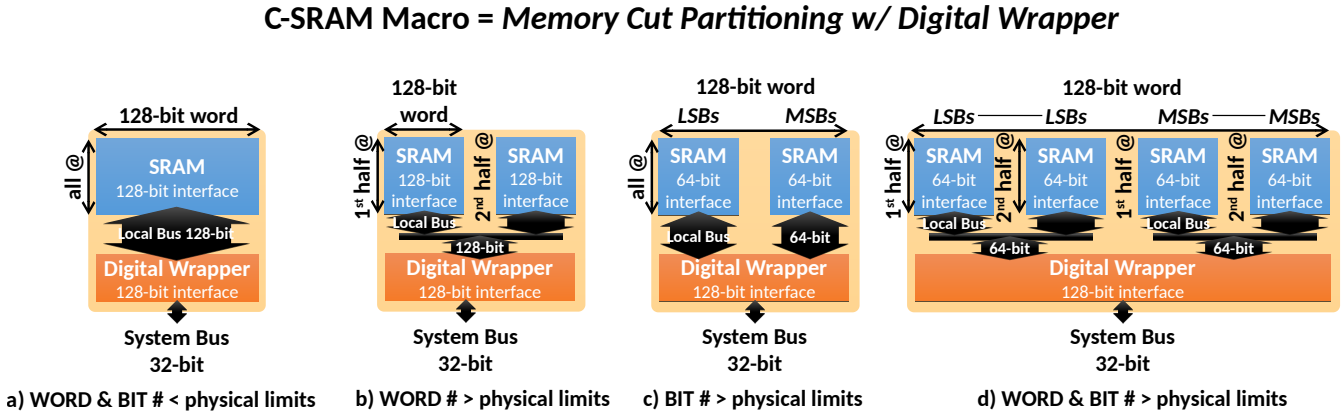


Figure 3.2.: Different ways of laying out the memories in the C-SRAM

[1] ↑This depends on the considered NVM but should work for Phase Change Memory (PCM) and Resistive Random Access Memory (RRAM)

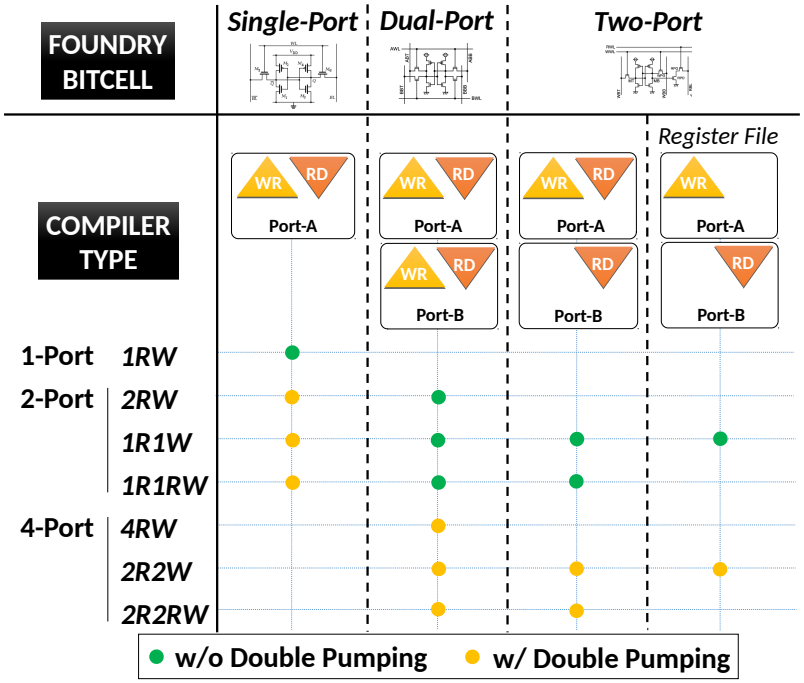
Still on flexibility about wrapping different kinds of memories, computing requires reading from memory to the processing unit, in general, two operands and writing back a single result to the memory. However, most SRAMs do not offer three ports but only one or two which may be dedicated to either read only or write only. Hence, our digital wrapper would spend a third of his time, in case of a one port memory, to reading and writing data instead of computing, greatly reducing the overall throughput of our solution. The double-pump technique can be used to virtually double the number of ports of a memory. A single port memory can then achieve two reads or writes in a single clock cycle by being active on both rising and falling edge (Figure 3.3a). Most manufacturers provide 2RW memories that are actually single port memories with double-pump technique. It is even possible to reach 6R6W memory using this technique [20]. By having more available ports, it is possible to fill a pipeline like architecture as shown in Figure 3.3b for the MAC operation which incurs three memory loads and a memory write. This permits us to lower the average Cycles Per Instruction (CPI) and thus increase throughput.

We have explained the rationales behind the choice of a digital wrapper around SRAM (C-SRAM) which can then be tightly coupled to a NVM. The main reasonings being design workflow, validation cost, development time as well as modularity and operators choices. SRAM purpose is to act as a working memory to save NVM's endurance. We can now proceed to the specifications of our C-SRAM which will act as a vector computing unit tightly coupled to NVM. Figure 3.4 presents what we want to achieve without the NVM.

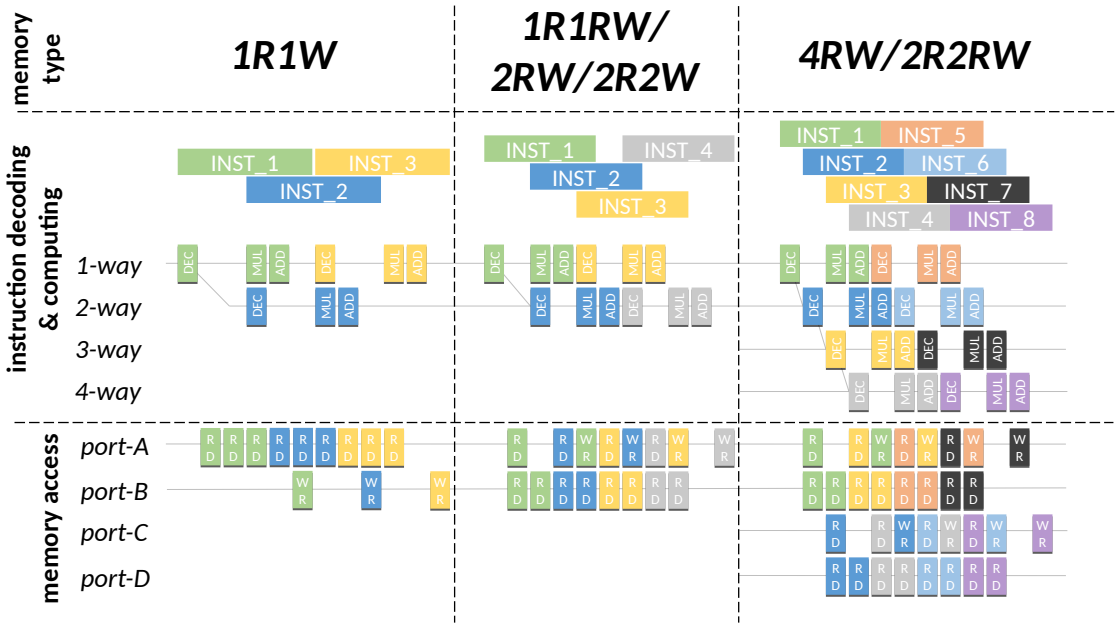
3.2. General design

3.2.1. Specification

The motivations behind the C-SRAM design has already been explained in previous section. However, we did not specify yet our target applications as well as some requirements such as vector size, which operators to implement, etc. We want to perform cryptographic applications as the use of IMC is believed to be a potential solution to avoid sensitive data transfer over buses. This can reduce side channels attacks vectors and opportunities. Moreover, treating big chunk of data at once may also further reduce correlation power analysis efficiency. Specific operators are required for this target but it only uses integer arithmetic. Another target is one of today's most prominent application which is neural networks and Artificial Intelligence (AI) applications. They both heavily performs matrix multiplication requiring only the multiplication and the addition operators. In order to save on silicon area and power consumption, we decide to use only integer operators as we believe that quantized neural networks will soon become the norm, at least in the inference phase. As such, we target generic computing and focus mainly on integer instructions as most applications, whether at the Internet of Things (IoT) node or the data center scale, exploit



(a) Obtaining more than 2-ports memories using double-pump technique. Double pumped memories are active on both clock edges granting two accesses per cycle and per port.



(b) Pipeline filling on the MAC operator for different memories considering no pipeline hazards. 1R1W memories are one-third filled (3 CPI), 2RW memories are half filled (2 CPI) and 4RW memories allow to completely fill the pipeline (1 CPI).

Figure 3.3.: Potential of the double-pump technique combined with our digital wrapper for better pipeline efficiency

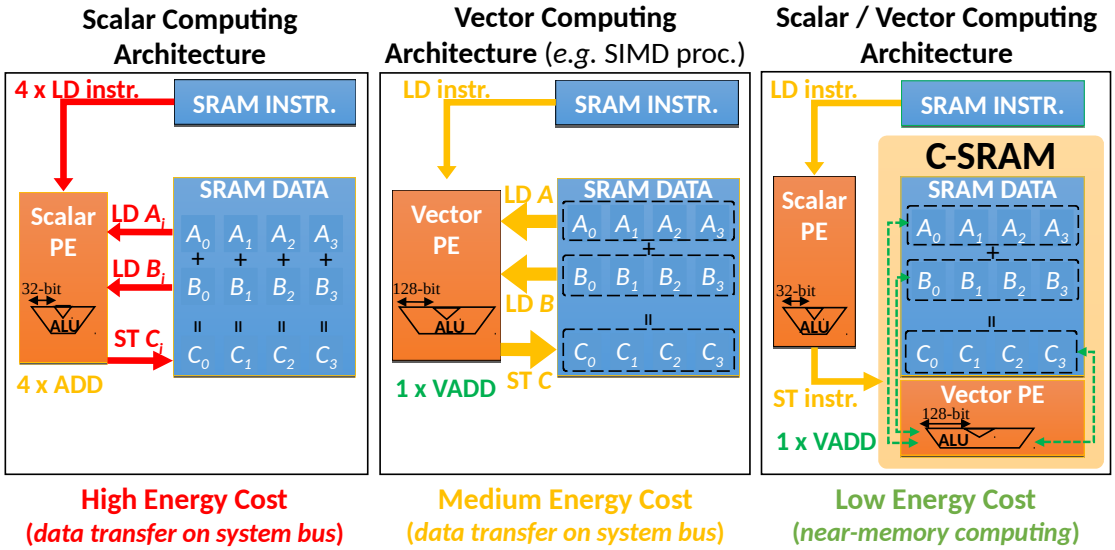


Figure 3.4.: Illustration of scalar (left), vector (middle) and scalar/vector (right) computing architectures. C-SRAM (right case) enables to drastically reduce data transfer on the system bus leading to significant energy savings based on near-memory computing.

them. Image processing, neural networks (inference only), cryptography, database, all make a heavy use of integer arithmetic. The limited area of integer operators is also a reason we headed in this direction. Although we do not want to implement control flow instructions, we still need to be able to merge 2 different computing branches into one result vector such as conditional assignment where it is applied to each element of the vector.

Now, we must also ask ourselves about the size of these hardware operators, i.e. the size of the operands and of the result. As mentioned before, we target quantized neural network applications mainly, but not only, which require relatively low precision such as 8 bits [154]. For cryptography applications, wider width is required, up to 128 bits or even 256 bits. We decide to first stick to 128 bits for some very specialized operators such as `hswap` needed for AES. For other common operators including the arithmetic ones, we choose to only implement the 8, 16 and 32 bits versions of them as these are the primitives for a lot of applications. From there, the list of required instructions is pretty straightforward with usual Arithmetic & Logical Unit (ALU) operations implemented such as:

- Boolean operators: `AND`, `OR`, `XOR`, `NOT` and their negated part (negated `NOT` is `COPY` or `MOV`) → 8 instructions;
- Advanced logical operators: `SLL` (Shift Left Logical), `SRL` (Shift Right Logical) and `Broadcast` → 3 sizes × 3 operations = 9 instructions;
- Arithmetic operators: `ADD`dition, `SUB`traction, `COMP`arison and `SRA` (Shift Right Arithmetic) → 3 sizes × (3 operations + 6 comparisons) = 27 instructions;

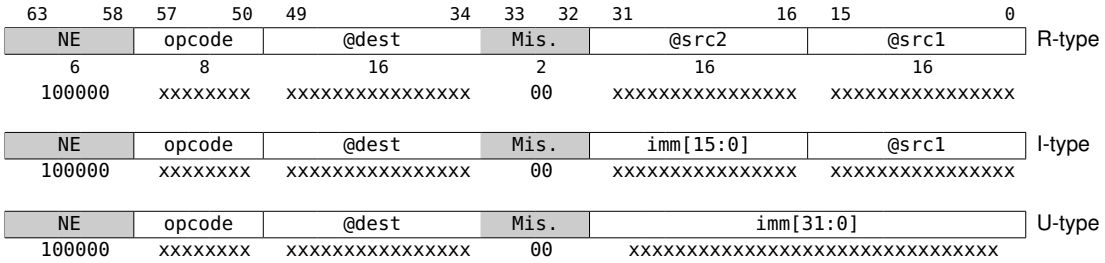


Figure 3.5.: Defined ISA for 32-bit system

- Advanced arithmetic operators: `MUL`tiplication and `MAC` (Multiply And Accumulate) → 3 multiplications for different signedness + 1 MAC = 4 instructions;
- Other operators: `HSWAP` (Horizontal swap) used for reduction operation and in some cryptographic applications which swaps vector elements → 2 instructions.

Finally, before starting to design our C-SRAM, we must as well completely define the Instruction Set Architecture (ISA), i.e. how we command the C-SRAM. We start from a previous work internal to the team [155] which uses both address and data buses to control the C-SRAM. It requires no hardware modification on the Central Processing Unit (CPU) side. For a 32-bit system, we have a total of 64 bits usable for our ISA as shown in Figure 3.5. The 32 most significant bits corresponds to the address bus bits while the following 32 are the data bus bits. The first 6 bits are equal to `0b100000`, with the first one indicating its a C-SRAM operation that is memory mapped. The other 5 bits are reserved for future use and may denote advanced use of the C-SRAM such as internal registers [57]. Then we have the opcode part and as we previously made the list of needed operations with the required width of the operands, it sums up to a total of 50 operations. 8 bits is thus enough to contain all the opcodes while maintaining some optimisations: we can use 3 bits in the opcode field to only denote the size of the operands (8, 16, 32, N/A). 8 bits opcode also leaves enough space to add more instructions if necessary. Then we got 16 bits of destination address and 2 bits set to 0 as we reach the end of the address bus bits, so we force the lower 2 bits to 0 to prevent the CPU from splitting our 4-byte access into 2 accesses if it was misaligned (i.e. if those 2 bits were not 0). After that we got a varying part depending on the type of the instruction. The type is encoded in the opcode and is used to distinguishes different instruction behaviors such as:

- R-type where both operands are data stored in memory so we need to send 2 addresses;
- I-type where the left hand side is stored in memory and the right hand side is a constant immediate;
- U-type where there is only one operand that is a constant immediate.

The size of the addresses fields is chosen so that we can address at least 1 MB with a vector size of 16 bytes. Hence, we need at least 16 bits that are then left shifted

by 4 positions within the C-SRAM to obtain the real physical address. Note that this ISA is only for research purpose as a real world system would have control pins similar to what is used for Dynamic Random Access Memory (DRAM) and other storage peripheries such as hard disks or smarts SSDs. We designed it to be easy to use with no hardware modification on the CPU side nor access to special system resources. Memory mapped periphery thus appeared as the best design choice in terms of complexity and ease of implementation on both side: hardware for C-SRAM and software for writing benchmarks and for our simulation platform (see Chapter 4).

3.2.2. ALU design

Even before starting implementation, it is good to know that `ADD`, `SUB` and `COMP` operators do not need to be designed independently but can all use a single global adder of the maximum size [156]. Indeed, using a unique adder of 128 bits, we can implement almost all the other arithmetic operators, except `SRA`. Doing a subtraction can be done by complementing the right hand side operand and using an input carry of 1. The comparison operator is done by using the subtraction and testing some bits to detect overflow (b is superior to a), underflow (a is superior to b) and equality (all bits of the result are 0). We actually need to implement only 3 full operators that are the adder, the multiplier and the shifter. All others are derived from these 3 and can be done in a single clock cycle with almost no penalty.

We first design the ALU with the chosen opcodes and instructions from previous section. Using [156], we implement only a full 64-bit adders (instantiated twice to reach 128 bits) in SystemVerilog and derive all other additions, subtractions and comparisons from it. Comparison is performed by using the result of the subtraction, with 2 more cases needed for different signedness of the operands which totals to 18 different opcodes for all possible comparisons and signedness. Finally for the multiplier, we must take care of the sign extension depending on the signedness of the operands as well as storing the result in a correctly sized vector. Indeed multiplying 2 8-bit data will yield a 16-bit result and different multiplier width are often present in ISAs. For addition, this is not the case as only 1 bit may overflow (or underflow), but it can easily be tested while multiply will most often yield overflowing result [157]. For the MAC operation which uses 3 input operands, we add an input to control whether multiplication and addition should be done in parallel for 4RW memories (Figure 3.3b). At the end, we just have a wide multiplexer tree to select the corresponding result. In order to reduce power consumption due to net switching, unused operator entries are zeroed when decoding the opcode.

3.2.3. Pipeline design

The general design is inspired by RISC CPU. As we do not want to redo a CPU in the memory, we do not implement any forwarding logic and hand it over to the programmer or the compiler. Moreover, we have to take into account the limited silicon area that is available in a memory chip. For the design to be easily adapted

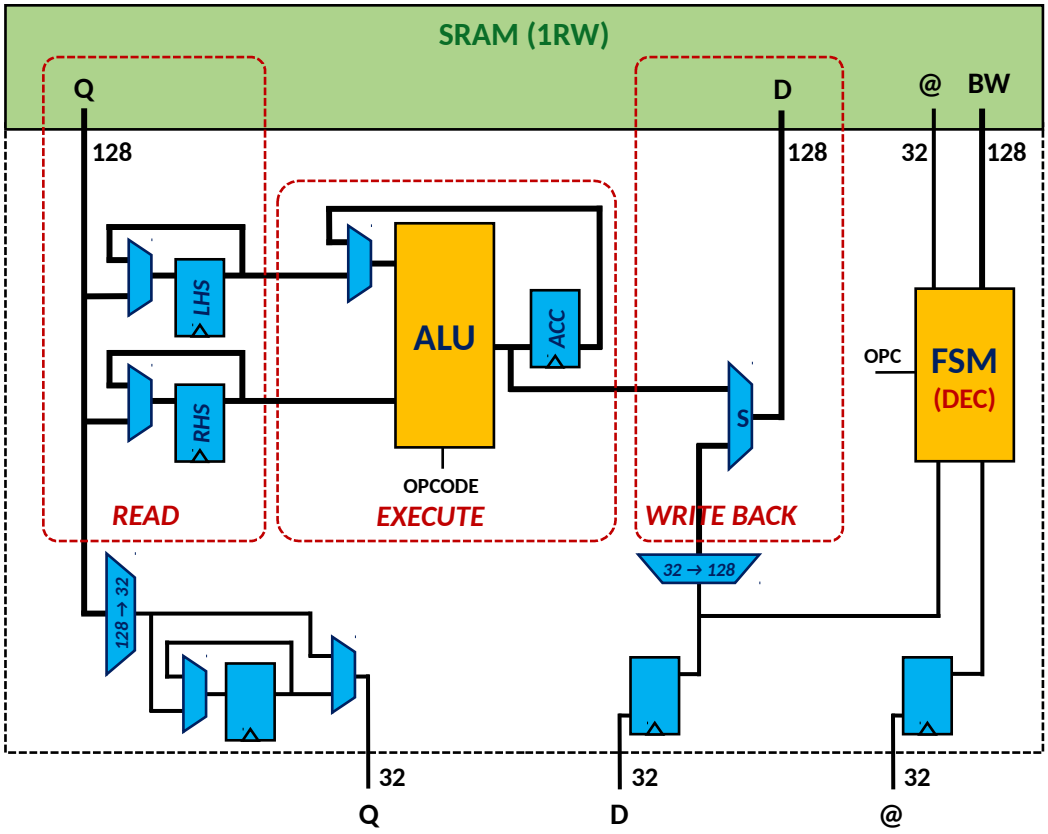


Figure 3.6.: Base implementation of our digital wrapper

to off-the-shelf memory compilers, we need to take into account the different types of memory which offer different number of ports. As such, the decode part must split the incoming instruction into different loads and writes accesses. The main job of the pipeline is to schedule all the loads and stores to the memory in order to get the maximum performance for incoming instructions. The implementation is based on the classic 4 stages pipeline scheme that is the base of most processors: Decode-Read-Execute-Writeback. Figure 3.6 represents the interconnection between a 1RW SRAM and the digital wrapper with some of its internal parts.

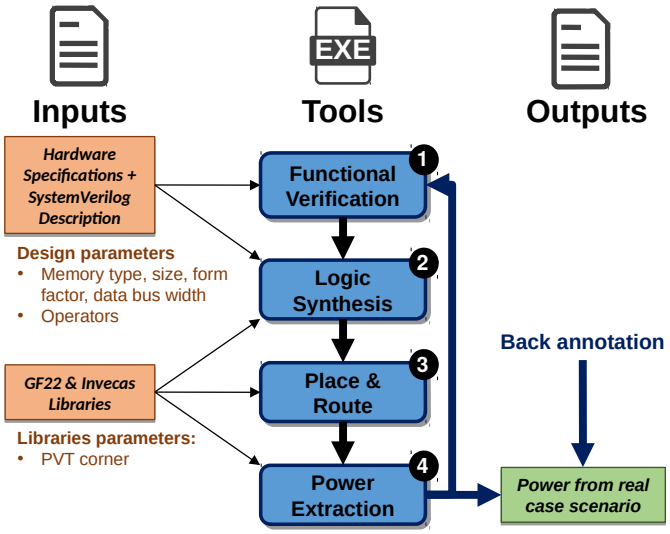
The purpose of the FSM is to manage the global pipeline, especially the **Read** and **Writeback** stages, as there are only limited number of ports to read and write data from and to the SRAM. As we decided to not implement forwarding from **Writeback** stage to **Read** stage, all data must go through the SRAM. To preserve consistency, the FSM will prioritize writes over reads so that previous operation results may be loaded with the correct value. Note that some advanced design also includes registers to store temporary data. These registers are addressable and the ISA is modified accordingly using the reserved bits [57]. We implement 4 versions of the FSM for different number of ports: 1RW, 1R1W/1R1RW, 2RW and 4RW. Each implementation is based on the hypothesis of a full pipeline, which is the worst case due to congestion at the SRAM. For the 4RW case, we also modify the ALU as it is possible to read 3 operands

simultaneously, allowing MAC instruction to be performed directly by the ALU instead of being split in one multiplication and one addition by the FSM. The FSM also serves to distinguishes between normal memory accesses and C-SRAM's instruction based on the previously defined ISA in Section 3.2.1. Again, to preserve consistency, all previous operations in the pipeline must be terminated before performing the memory access. If it is a read access, it might load a stale value when an ongoing instruction in the pipeline writes to that specific address (Read After Write (RAW)). If it is a write access, it might store a data that will get overwritten by an ongoing instruction (Write After Write (WAW)).

Read stage is composed of 2 flip-flops feeding the **Execute** stage. Below is also some circuitry to handle normal read accesses. First there is a selection of the 32 correct bits out of 128 based on the lower 2 address bits. This data word is then fed directly to the output port as well as an internal flip-flop to maintain the data for the following cycles, accordingly to the SRAM specification. Note that this is a special case corresponding to the SRAM we use, other SRAMs may maintain the data valid for only one cycle, thus this part could be removed. **Execute** stage has already been discussed in the ALU section. It feeds one flip-flop used to store temporary result from multiplication before accumulating for the MAC instruction. Then, we have the **Writeback** directly fed from the ALU and going right to the SRAM. Again, there is some circuitry below to manage normal write accesses with a demultiplexer to put the write data in the valid position based on lower address bits. Some parts are hence present twice: once in the SRAM and once in our digital wrapper; for instance, the read/write circuitry to manage normal accesses. Thus we have a small area overhead compared to the full custom case presented in Figure 3.1. Note also that the presented case with 128 bits memory here is just an example used in our measurement but our workflow and design IP can accommodate any memory width.

The implementation of the digital wrapper is done in SystemVerilog. In addition to the modules presented in Figure 3.6, we have a reset synchronizer and a memory controller interspersed between the SRAM and the wrapper. The design global inputs are clock, reset signal, chip enable, write enable, address and data bus. Global outputs are a busy signal to indicate if the C-SRAM pipeline is full and the data bus. Reset synchronizer is used to make sure that asynchronous reset rise with in sync with the clock signal. Memory controller is a wrapper module of all the available memories that select the correct one and connects the ports.

We defined the ISA specifications which include basic arithmetic operators (ADD, SUB, etc.) as well as 8-bit multiplication and support for MAC operation. Thanks to small tricks in designing the ALU, we need to instantiate only one full width adder, or split it in smaller adders if need be, rather than have multiple hardware operators for addition, subtraction and comparison. Pipeline design is rather conventional based on the **decode**, **read**, **execute** and **writeback** stages. Its only specificity is to manage memory accesses with C-SRAM instructions and normal accesses in an interleaved manner.



	Tool	Vendor	Version
1	ModelSim	MentorGraphics	10.5c
2	DesignCompiler	Synopsys	n-2017.09-sp5
3	Innovus	Cadence	19.13
4	Star-RC	Synopsys	o-2018.06-sp2
	PrimeTime Power	Synopsys	o-2018.06-sp5

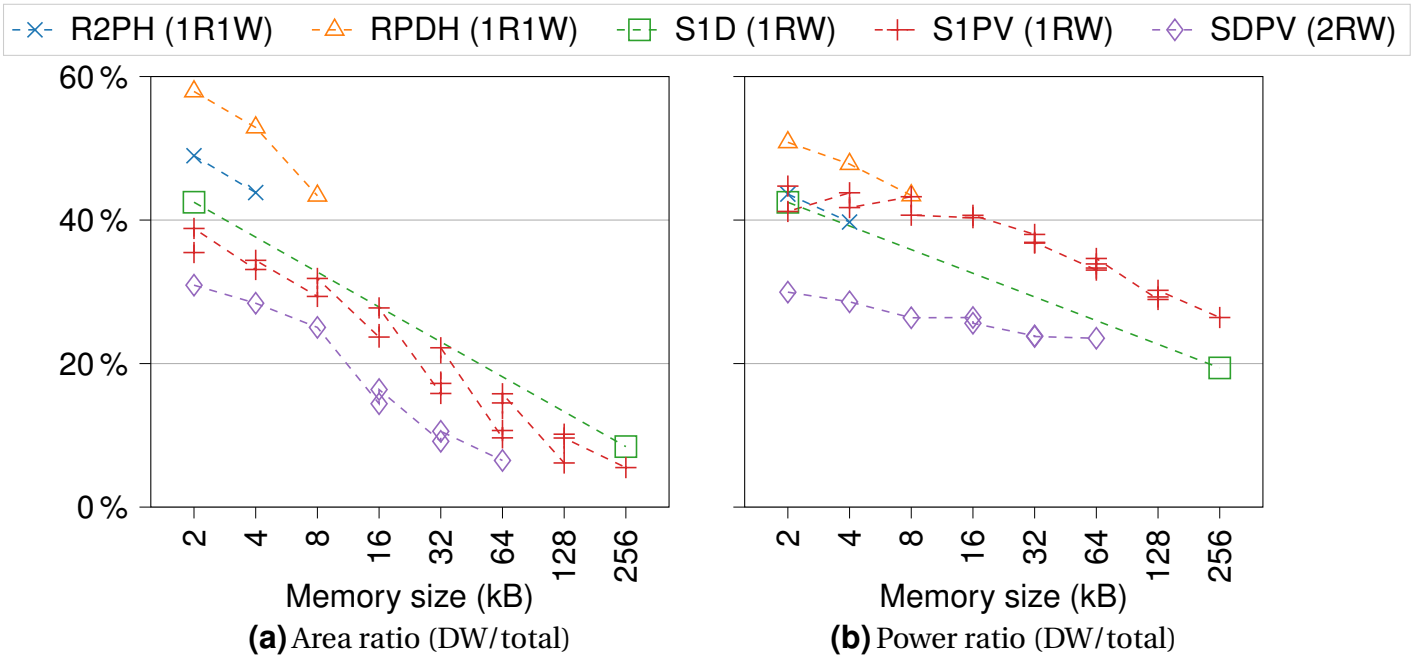
Figure 3.7 & Table 3.1: Design workflow used for C-SRAM digital wrapper

3.3. Experimental results

For the exploration of different memories, we used the 22 nm FDSOI design platform technology from GlobalFoundries for both standard bitcells and the memories. The memory models are provided by Invecas.

3.3.1. Workflow

The general workflow is given in Figure 3.7. For the functional verification, we use ❶ ModelSim and test all opcodes independently. For the sake of safety, the testbenches are written in `tcl` language so that a mistake in SystemVerilog may not be copy-pasted directly. Synthesis is done by ❷ DesignCompiler using GF22 libraries with the target frequency set to 500 MHz for all designs with no particular constraint. For the place and route, we use ❸ Innovus and we lay out the I/O pins along the largest dimension as most memories have a rectangular shape. The design is constrained to have a density around 50 %. Once the place and route is finished, we extract a SDF file using ❹ StarRC. Then both the place and routed Verilog and the extracted SDF files are used for back annotated post place and route simulation using ❶ ModelSim to create a VCD file. The VCD file is then processed by ❹ Primetime Power to extract power value from our testbenches. A complete run takes about 5 hours.



(a) Area ratio (DW/total) **(b)** Power ratio (DW/total)

Figure 3.8.: Area and power overhead for different kind of SRAMs (lower is better)

3.3.2. Simulation results

We perform full digital workflow for 5 different types of memories totaling 24 cuts. The used memory types are:

- R2PH which is a 1R1W Two-Port (noted TP), optimized for performance (blue cross ×);
- RPDH which is a Single-Port (SP) double pumped into a 1R1W pseudo TP, also optimized for performance (orange triangle △);
- S1D and S1PV that are 1RW single-port optimized respectively for density and for performance (respectively green square □ and red cross +);
- SDPV which is a 2RW Dual-Port (DP) optimized for performance (purple diamond ◊).

3.3.2.1. Physical design extraction

First, we note that the area overhead is limited, especially for big memories, as shown in Figure 3.8a, but can be up to 50 % for small ones (<8 kB) and only 5 % for the biggest ones (>128 kB). This is to be expected as our digital wrapper takes around 5000 μm^2 while the smallest memories are around 4000 μm^2. As our design is not fully optimized for density, we can expect to lower this ratio by pushing the place and route constraint further. For the power part presented in Figure 3.8b, the digital wrapper can represent up to 50 % for small memories and decrease down to 20 % for the biggest memories.

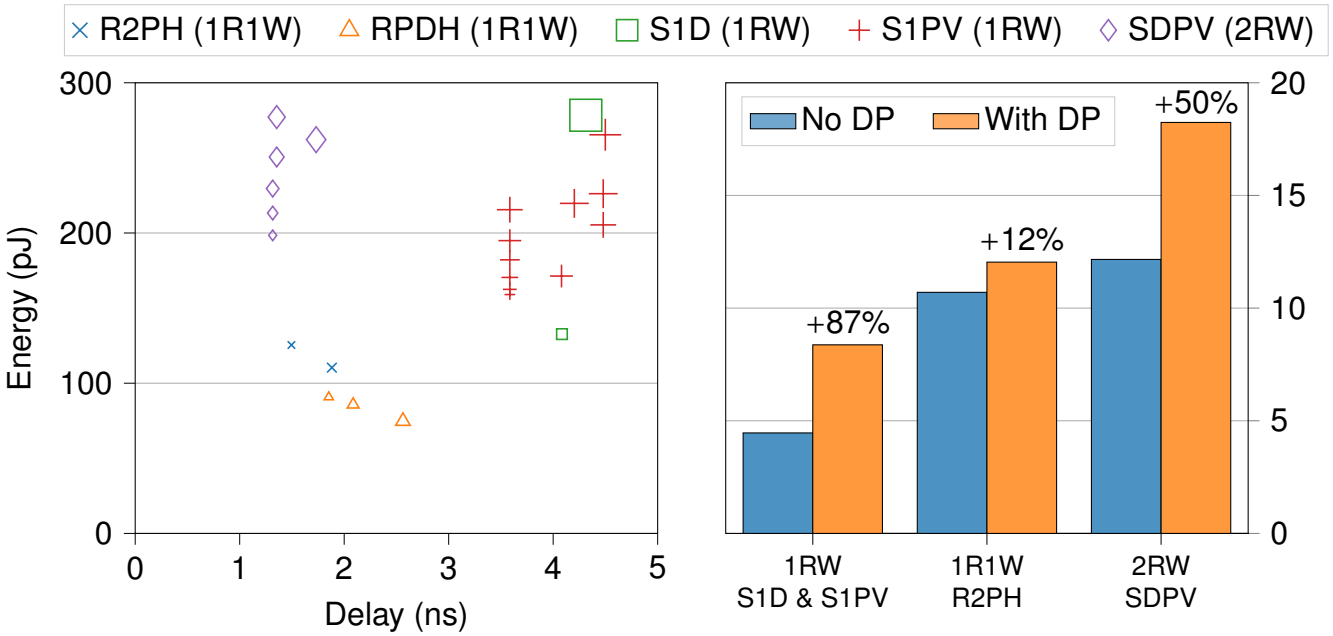


Figure 3.9.: Energy versus delay for MAC instruction for all C-SRAMs (marker size is proportional to memory size, lower left is better)

Figure 3.10.: Throughput comparison of different SRAM types with and without double pump technique (higher is better)

The memories are quite optimized in terms of power consumption, especially for the static power inherent to SRAM bitcells, and we guess that is the reason why our digital wrapper can represent such a big part of the global power consumption. For both area and power, the overhead decreases exponentially with the memory size.

Figure 3.9 plots the energy and delay required to perform a multiplication and accumulation considering the maximum frequency reachable on the given memory. As such, we observe a Pareto front where an optimum in either energy or delay can be chosen. We remark that the most energy efficient memories are the smallest memories while capacity has less impact on delay. The 1RW type memories have the worst delay and energy as they are the biggest memories with sizes up to 256 kB for instance. The bigger the memory, the slower it is and the more energy hungry it is to power correctly all the bitcells. Unsurprisingly, the 2RW memory is the fastest as it can read 2 operands per cycle, but on the other hand, it is also the biggest energy consumer. From this Pareto front, we conclude that the optimal designs are the C-SRAMs with 1R1W memories with the lowest energy to compute a MAC instruction at around 100 pJ and an average delay of 2 ns. However these memories are very limited in size with the maximum capacity at just 8 kB. If we take into account the double-pump technique, we could suffer a 25 % frequency reduction yet still increase the throughput (Figure 3.10), from 12 % for 1R1W to 87 % for 1RW memories. Indeed, what limits the throughput is the number of memory accesses in a single clock cycle, as shown in

Figure 3.3b. Doubling this number can yield very interesting gains at low cost. An important point to note is that our designs are only limited by the maximum reachable memory frequency to respect timing constraints. Our digital wrapper has no effect on this maximum frequency.

3.3.2.2. Performances versus state of the art

We plot the area normalized throughput versus the energy efficiency in Figure 3.11a, still for the 8-bit MAC operation. The lowest normalized throughput is around 10 GOPS/mm^2 for the largest 256 kB 1RW SRAM as 95 % of the area is taken solely by the memory (Figure 3.12). This memory also has the lowest energy efficiency at 0.3 TOPS/W as most power is driven by the SRAM array. We reach a maximum of 425 GOPS/mm^2 and 2.05 TOPS/W with a 2 kB (128 × 128 bits) 1R1W TP SRAM (R2PH). This memory has an area of 6928 µm^2 while the smallest one has an area of 4333 µm^2 (RPDH of 2 kB (128 × 128 bits)), therefore the smallest memory not necessarily has the best normalized throughput. Indeed, RPDH is a single port memory double-pumped into a pseudo two port, thus it is much smaller but suffer from a lower frequency (−20 %) but also draws less power (−24 %). All in one, the smallest RPDH memory is slightly less efficient than the R2PH one. The bottom left corner in Figure 3.11a is filled with all the 1RW single port memories (green square □ and red cross +) which achieve low throughput and efficiency due to their greater memory size (up to 256 kB). We also spot the 2RW dual port (purple diamond ◊) in the same corner although we would expect them to have better throughput and efficiency. These memories have way more physical footprint (up to 2.5× as they use 8T bitcells) and also consume much more power compared to the single port memories (up to 3×). So the expected gains from doing 2 accesses per cycle are crushed by the huge area and power flaws compared to other memories. This is coherent with our analysis in Chapter 2 and section 3.1 on the use of non standard bitcell. Similarly, we present the memory size versus the energy efficiency in Figure 3.11b. We observe that the memory size has a huge influence on the efficiency for the 1R1W memories whereas it is less important for 1RW and 2RW memories. The duplication of some memories is due to different form factors that yield same memory size. We notice that this form factor can vary the efficiency with a high impact (±50 %) for small (≤64 kB) 1RW memories while it has little impact (<10 %) on 2RW memories. From these two figures, we deduce that the most important design choice is the memory type, followed by the memory size and finally the form factor.

To compare ourselves with the state of the art, we reproduce the figure from [149] in Figure 3.11c and Figure 3.11d. From a quick look, it appears that we perform worse than most non IMC and IMC implementations. Indeed, we span roughly one order of magnitude for both normalized throughput and energy efficiency, respectively between 10–400 GOPS/mm^2 and 0.3–2 TOPS/W, whereas non IMC energy efficiency is often around 10 TOPS/W with a similar normalized throughput as C-SRAM. IMC implementations outperform non IMC ones as they are supposed to, with most of them over 10 TOPS/W and 100 GOPS/mm^2 and achieving 658 TOPS/W

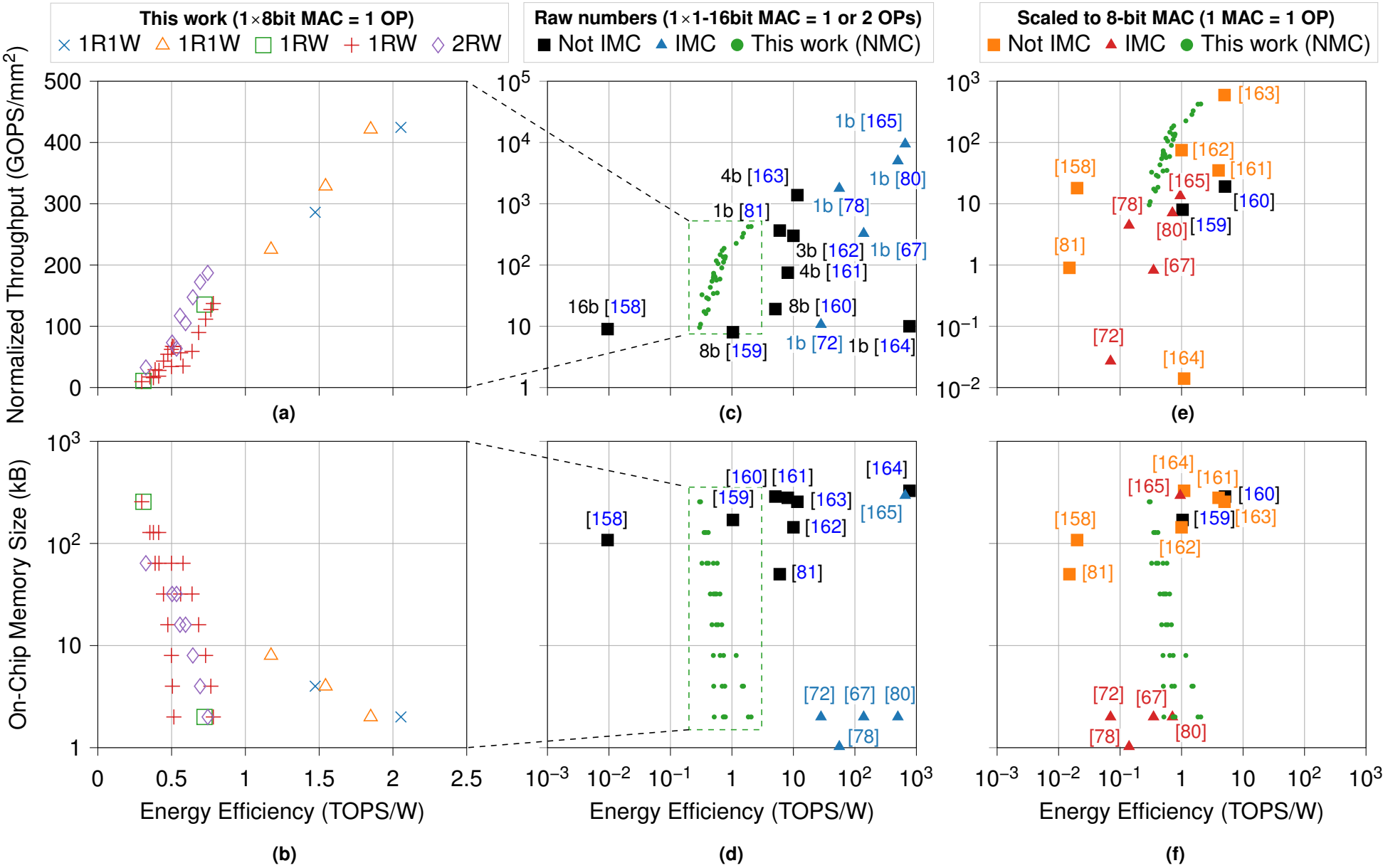


Figure 3.11.: Throughput (upper row) and size (lower row) versus efficiency of our solution and state of the art works on MAC operation. **(a)** and **(b)** show detailed plots of C-SRAMs performances. **(c)** and **(d)** are comparison with state of the art accelerators and IMC implementations from [149]. **(e)** and **(f)** are scaled version of **(c)** and **(d)** to 8-bit MAC such that all papers use the same definition for an operation that is a complete 8-bit MAC.

and 9438 GOPS/mm^2 for the best one [165]. However, the definition of an operation changes from a paper to another as well as the operator size. Some are binary or ternary while some are using 8 bits or even 16 bits; moreover, sometimes a MAC is counted as two operations (multiplication and addition) or as a single one. The original figure from [149] seems to consider only reported numbers by authors without delving into the details. To deal with this definition change, we scale all results to 8 bits MAC and consider that a MAC is only one operation. To convert to an equivalent of 8-bit operations, we consider the number of logical operations needed to perform an 8-bit MAC, which is roughly equal to 400 for 1-bit operations. However, some of these 1-bit papers can only achieve a subset of all logical operations, meaning they need several operations to perform a single bitwise AND for instance. A more conservative factor is around 700 as in [165] although they do not explain how they got this ratio nor if it applies to 8-bit operations. I have considered to scale our work by the following factors which would place our work in the top right corner of both figures. But 8-bit MAC is not equivalent to 700 1-bit operations, it is not reciprocal and thus would not be honest.

Hence, we scale accordingly the non IMC works. [158] is a 16-bit reconfigurable Convolutional Neural Network (CNN) and although it is not equivalent to two 8-bit MAC operations, we multiply it by 2 for simplicity. [159] and [160] are 8-bit neural network accelerators and thus are not scaled. [161] is a reconfigurable 4 to 16-bit CNN but the reported numbers are for 4-bit data, so we divide them by 2, although once again it is not equivalent to half a 8-bit operation. [81] is a binary/ternary DNN that we scale down by 400. [162] and [163] are 1-16b CNN accelerators but the reported numbers are for 3 or 4-bit data, however the authors also provide the data for 8-bit operations. [164] is a binary CNN scaled down by 700. For the IMC implementations, [72] is a CNN accelerator using 7-bit inputs and outputs but filters are 1-bit and use analog computing, we therefore scale it down by 400. [67] and [78] are binary CNNs scaled down by 400. [165] is also a binary CNN that is scaled down by 700 according to their own numbers. [80] is a classifier using 1-bit data internally scaled down by 700. First, we look at the normalized throughput versus energy efficiency in Figure 3.11e and see that our solution is actually on par with state of the art. Some non IMC accelerators do have a better energy efficiency, but our approach offers more normalized throughput. Indeed, we are using very compact memories and our digital wrapper area is limited while not limiting memory frequency. Only a single work outperforms our C-SRAM in both metrics [163]. As such, we can conclude that application specific accelerators have a better efficiency but are limited in their normalized throughput due to the high area needed for specific operators. For the IMC works, as they all uses binary implementation, their 8-bit performances are quite low, one to two orders of magnitude below our works. Figure 3.11f shows the memory capacity versus the energy efficiency. We observe that state of the art IMC has very low (2 kB) on chip memory which greatly limits its use while our approach proposes numerous memory sizes. However, non IMC accelerators have more memory as they use unmodified SRAMs. Our maximum sized memory has 10× lower energy efficiency than best non IMC papers.

We implemented a digital wrapper around SRAM memories that can be used in coordination with NVM memories to benefit from both memory type advantage. This digital wrapper performs generic computation and is suitable for neural network applications using MAC instruction, cryptography, image processing, etc. Our design is rather compact thanks to smart ALU implementation. We compared ourselves against the state of the art and showed that our solution has similar performance using a fast and automated design approach. This design method and its automation has been patented [166]. Future works include multi-clock design to give the memory and the wrapper each their own clock. An extension of our current design with fixed point, then with floating point operator and even variable precision based on the application targeted or the system constraint can be envisaged. Some place and routed floorplans presented in Figure 3.12 illustrate some flexibility with different memory types and number of memories. The work carried in this chapter was published in DATE 2020 [167].

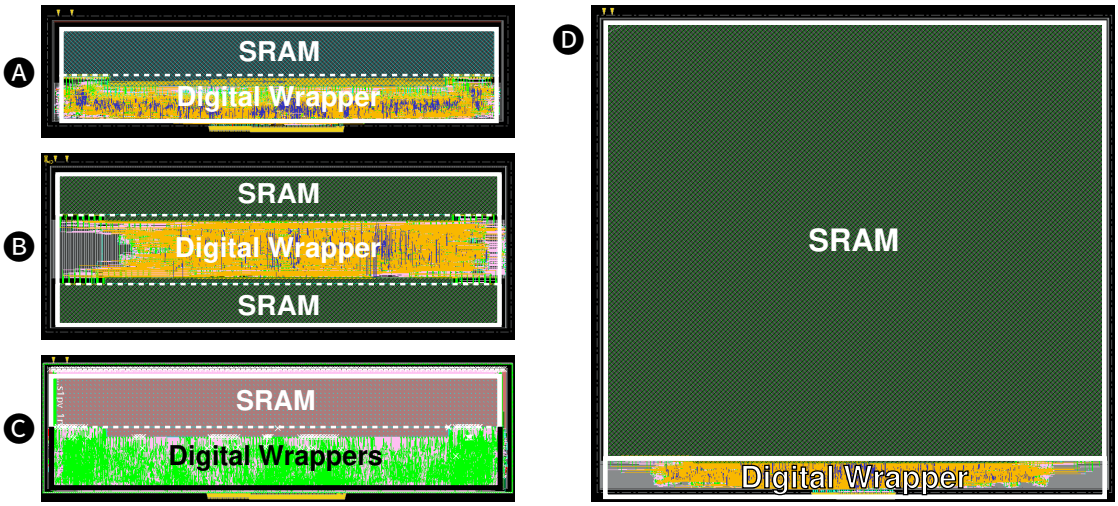


Figure 3.12.: Different place and routed floorplans obtained with our workflow: A is a 2 kB (128 × 128) 1RW memory, B is 2×1 kB (2 × 128 × 64) 2RW memories, C is a 2 kB routed on only 2 metal layers, D is a 256 kB 1RW memory.

4. Simulation platform & Tools

Donc, on est d'accord : artichette, tichette de 2, tichette de 3, tichette de 21, michette, chiédèque, mique, sganadabarlane, résiné et raitournelle.

— Arthur IN *Kaamelott* BOOK IV, EPISODE 71, « *Perceval fait raitournelle* »

The good thing about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do.

— Ted Nelson

Different platforms and tools are discussed in this chapter to better reflect the wide span of existing solutions to model memory systems. We explain the differences between in situ measurement, exploration, simulation and instrumentation that are all distinct possibilities to study the impact of our proposed solution on the memory system. We also introduce the benchmarks that are used in our experiments, in relation to the context given in Chapter 1. Finally, we develop our own evaluation tool, extending work carried in Chapter 3, whose results are examined in Chapter 5.

Contents

The C-SRAM we developed in Chapter 3 is a hardware component, but we must first test it through architectural simulation to acknowledge its advantages or deficiencies in a real system. We introduce the benchmark suite that is used in Chapter 5 for all our experiments in section 4.1. It includes different pattern accesses benchmarks and a real case application. Then we also present the existing simulation platforms and detail how they do not suite our needs in regard to the memory system (section 4.2). We also talk about real measurements using hardware counters or more theoretical aspect with analytical models. In section 4.3, tools required to model all levels of the memory hierarchy are introduced along with an extension of the work on the C-SRAM. Finally, we develop our own simulation platform in section 4.4 and perform a validation stage against other known tools.

4.1. Used benchmarks

As presented in Chapter 1, what pushes for In-Memory Computing (IMC) is the emergence of *big data* applications such as neural networks. To simplify, we can take a look at the kernel functions used in typical neural networks applications given in Figure 4.1. We observe that more than 80 % of the algorithm time is spent performing matrix multiplications. This is coherent with the state of the art in Chapter 2 where most works focused on Matrix Vector Multiplication (MVM). Note that even Convolutional Neural Networks (CNNs) uses matrix multiplications instead of convolutions and the presence of a *matrix assign* operation. This operation is also named *image to column* and its purpose is to lay out data correctly so that convolution is transformed into a simple matrix multiplication. Other data intensive targeted applications include classic (i.e. not Artificial Intelligence (AI)) image processing such as filtering, database searches, problem optimizations, etc. Some classic computer algorithm like sort are also excluded as they manipulate individual elements which our vectorized approach developed in Chapter 3 cannot handle efficiently. Comparison can be done in an efficient vectorized way, but for sorting, scalar data still has to be moved one by one.

We considered seven benchmarks with different access patterns and computing complexity: three benchmarks of linear complexity (streaming like) **Hamming Weight**, **shift-or** and **axpy** as custom kernels, three kernels from Polybench [169] including two

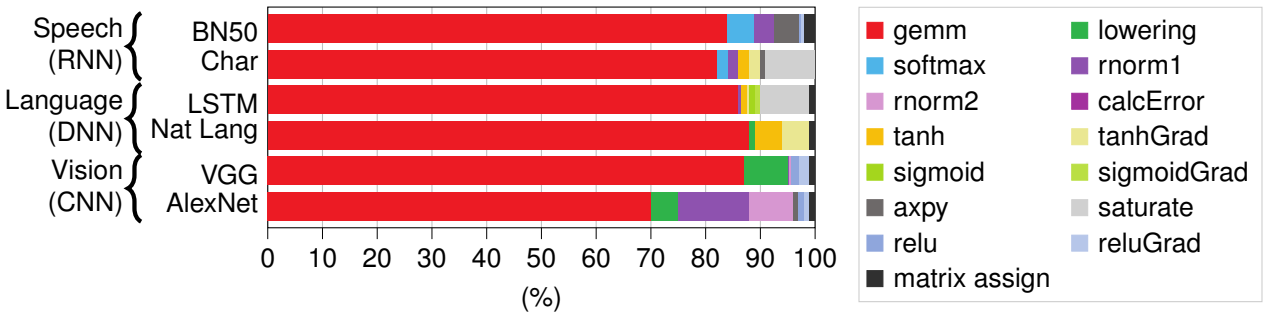


Figure 4.1.: Neural network core functions time distribution. From [168]

quadratic ones, **atax** and **gesummv**, and one cubic kernels, **gemm**. Finally, we also use a real application case, **darknet** [170], which is an open-source implementation of different CNNs. Some of them are *compute bound*, i.e. they are limited by the speed of the Central Processing Unit (CPU) performing operations while others are *memory bound*, limited by the speed at which data is delivered by memory. We expect memory bound kernels to benefit more from IMC as data is served at a way higher throughput and these benchmarks apply simple operations.

4.1.1. Linear benchmarks

Linear benchmarks have a streaming access pattern where the input data is read only once and mostly involve data reduction, i.e. taking an input vector of data and apply an operation to all members to get only a few final results.

Hamming Weight is an algorithm used in information theory to compute the number of symbols that differs in two vectors of same length, with bitwise XOR operation. It is also used in Binary Neural Network as the `popcnt` operation where the second vector is replaced by a null vector. The output result is a scalar integer yielding the number of differing bits in both vector. It is a compute bound benchmark due to a lot of bit twiddling operations with the scalar version having only 3.7 % of memory accesses relative to the number of instructions.

Shift-or is an implementation of the *bitap* algorithm [171] used to match patterns in DNA sequences. It takes two inputs, a sequence to search for and a file containing the sequence to be searched for. It uses an efficient compression of DNA sequences with 2 bits encoding. It is also a compute bound kernel totaling a little less than 17 % of memory accesses.

AXPY is a vector-scalar product kernel computing $\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$ with $\mathbf{x}$ and $\mathbf{y}$ vectors and α a scalar. It is part of the Basic Linear Algebra Subprograms (BLAS) library [172]. The difference with *Hamming Weight*, in terms of access pattern, is that the result is of the same size as the input, thus it is a write intensive kernel. This benchmark is memory bound with 43 % of memory accesses but it is mainly limited in terms of throughput by the intensive and slow writes to the Storage Class Memory (SCM).

4.1.2. Quadratic benchmarks

Quadratic benchmarks are mostly MVM kernels used in combinatorial optimization or image processing applications. The output result is in general a single vector out of an input matrix and vector.

ATAX is a Polybench kernel computing a $\mathbf{y} \leftarrow \mathbf{A}^T(\mathbf{Ax})$ with $\mathbf{A}$ a matrix and $\mathbf{x}, \mathbf{y}$ vectors. It is used in linear solvers and also in some transformations for image processing.

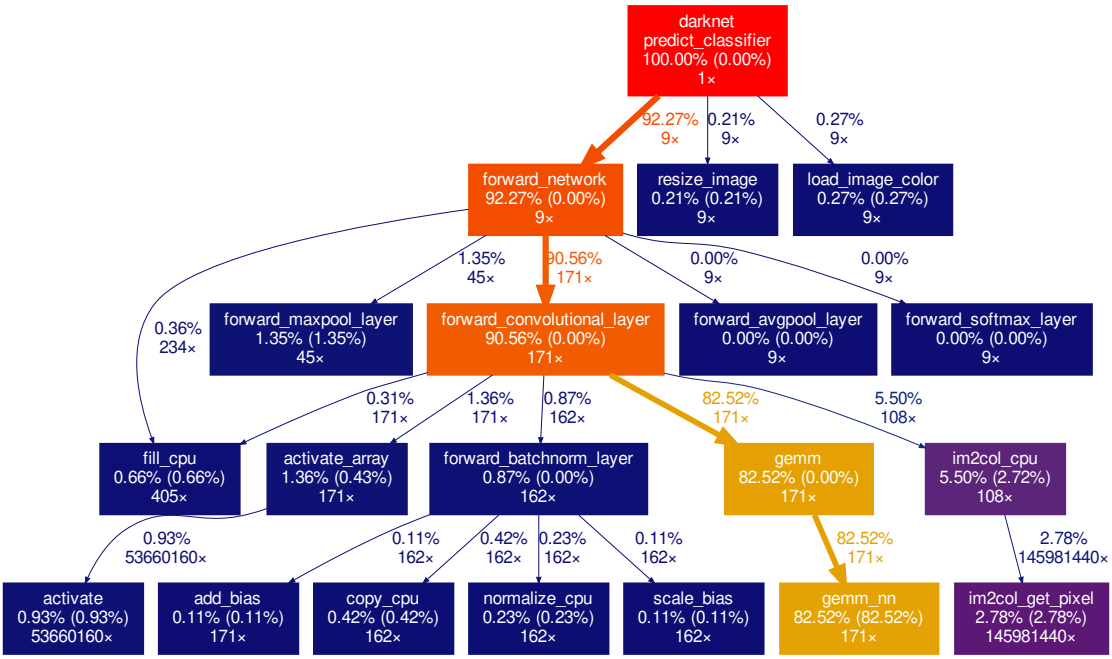


Figure 4.2.: Darknet callgraph for image classification. It does not add up to 100 % as initialisation and small functions have been removed for simplification. First number is time spent in the function including sub function call, parenthesis number is time spent in the function itself excluding sub function call, last number is the number of calls to the function.

With 38 % of memory accesses, it is a memory bound kernel where the multiple uses of the input matrix is the bottleneck.

gesummv(GEneral SUMmed Matrix-Vector) is a Polybench kernel computing a sum of matrix-vector multiplications as $\mathbf{y} \leftarrow \alpha\mathbf{Ax} + \beta\mathbf{Bx}$ with $\mathbf{A}, \mathbf{B}$ square matrices, $\mathbf{x}, \mathbf{y}$ vectors and α, β scalars. Memory accesses represents 36 % of the instructions which makes this benchmark memory bound. The accesses to multiple matrices and vector at each iteration are the bottleneck.

4.1.3. Cubic benchmarks and real application

Cubic benchmarks are matrix multiplications kernels widely used in image processing, neural networks and scientific simulation. It is the base operation of numerous algorithms and is often heavily optimized in BLAS libraries. For CNNs, it represents 80 % of the total time with the rest being laying out the input matrices to transform the convolution into a matrix product.

gemm(General Matrix Multiplication) is a Polybench kernel computing a matrix multiplication as $\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$ with $\mathbf{A}, \mathbf{B}, \mathbf{C}$ three matrices and α, β scalars. It is also a

Table 4.1.: Benchmarks parameters

Benchmark	Parameters	Input size	Output size	Loop iterations	Bound
hd	$n = 2^{30}$	$\mathbf{x} : n$ $\mathbf{y} : n$	1	n	Compute
so	$n = 2^{31}$	n	1	n	Compute
axpy	$n = 2^{30}$	$\mathbf{x} : n$ $\mathbf{y} : n$	n	n	Memory
atax	$m = 3 \cdot 2^{14}$ $n = 3 \cdot 2^{14}$	$\mathbf{A} : m \times n$ $\mathbf{x} : n$	n	$2mn$	Memory
gesummv	$n = 2^{15}$	$\mathbf{A}, \mathbf{B} : 2n^2$ $\mathbf{x} : n$	n	n^2	Memory
gemm	$m = 3 \cdot 2^{14}$ $n = 2^{15}$ $p = 2^{15}$	$\mathbf{A} : m \times p$ $\mathbf{B} : p \times n$ $\mathbf{C} : m \times n$	mn	$m(n(p+1))$	Memory

memory bound benchmark with 50 % of memory accesses and is only limited by the memory bandwidth.

Finally, we also use a real case application called **darknet** [170] which is an implementation of several neural networks although we only use CNN with Image-Net database [173]. As said for *gemm*, we can transform convolution into matrix products with a function named *image to column* (`im2col`) which takes around 5 % of the execution time, as shown in the darknet callgraph (Figure 4.2). This function lays out the input matrix as well as the filter in the memory so that the matrix multiplication can take place with the right parameters. The rest (80 %) is mostly matrix multiplications and other operations required for the network to operate. Loading files from disk is negligible with less than 0.5 % including resizing the input layer. The network contains 25 layers of which 19 convolution layers interlaced with 5 maxpool layers and a final average layer followed by a softmax layer. As it is mainly matrix multiplication, it is a memory bound application with 42 % of memory accesses.

For all the benchmarks, we used a dataset between 1 GB and 2 GB thus with an amount of data superior to the Dynamic Random Access Memory (DRAM) capacity to enforces transfers between DRAM and SCM. It is also representative of today's applications dataset size and behaviors. A small recap is available in Table 4.1.

4.2. Existing platforms

In this section, we present the different system evaluation platforms and provide a short focus for each one regarding advantages and drawbacks. To understand what makes a platform a good or a bad choice, it is necessary to know the difference between

instrumentation, simulation and performances measurement. In situ measurement is done at the hardware level and is presented in Section 4.2.2 but is limited to existing events. Exploration platforms are what we call hardware model tools that are described in section 4.3; they are designed to explore different designs, devices and technology parameters effects on the overall circuit. On the other hand, simulation and instrumentation tools are designed to run a program, commonly referred as a benchmark, and to extract metrics of interest such as energy, timing, disk accesses, etc. Several platform are considered in Section 4.2.3.

4.2.1. Analytic model

It is in theory possible to get an approximation of the numbers of accesses in a given memory using an analytic model. Yet, it is only applicable for some simple applications like the one we use and described in section 4.1, and not for real world applications where access patterns are unpredictable and depend on user inputs. If we only take into account the major part of our applications, i.e. the part that takes the most time and thus the most energy consuming, which is often matrix multiplication as shown in Figure 4.1, we can deduce some analytic models. The regularity of the algorithm makes it possible to know which data will be used and when, although the hard part lies in the cache management policy and the dependence for each cache. Analytical models exist [174–176] but are not used because they cannot perceive the randomness of a real system such as address alignment changes and erratic program behaviors. In our case, they would not be able to grasp the data exchange between the C-SRAM and the caches that entirely depends on the program analyzed, i.e. it is not predictable. Hazards such as system interactions and other programs interactions make this an inconvenient and inaccurate solution.

4.2.2. Hardware counters

As our objectives are to measure timings and energy spent for a given benchmark, the hardware counters available in the reference architecture, namely an x86-64 machine, can be used to extract these measurements. The UNIX utility `perf` [178] can be used to this end. Other solutions include the PAPI [179] instrumentation library as well as Intel's VTune profiler.

`perf` is a tool that manages hardware counters on Linux systems. It can easily extract given events such as L1D hits and misses. A list of predefined events can be obtained with the command `perf list`. In addition, `perf` can be given an hexadecimal register number to monitor all kind of events using the command `perf stat -e rXXX ./exe`. The list of all the available counters is accessible in the Intel documentation [180]. However, there are limitations in the number of available counters as well as interactions with other processes on the same machine. For instance, on a typical x86 desktop machine, there are only 4 hardware counters available, named Performance Monitoring Unit (PMU). Measuring more events either means that `perf` will interleave the events measurement (i.e. counter 1 will measure event A for 15 ms then will

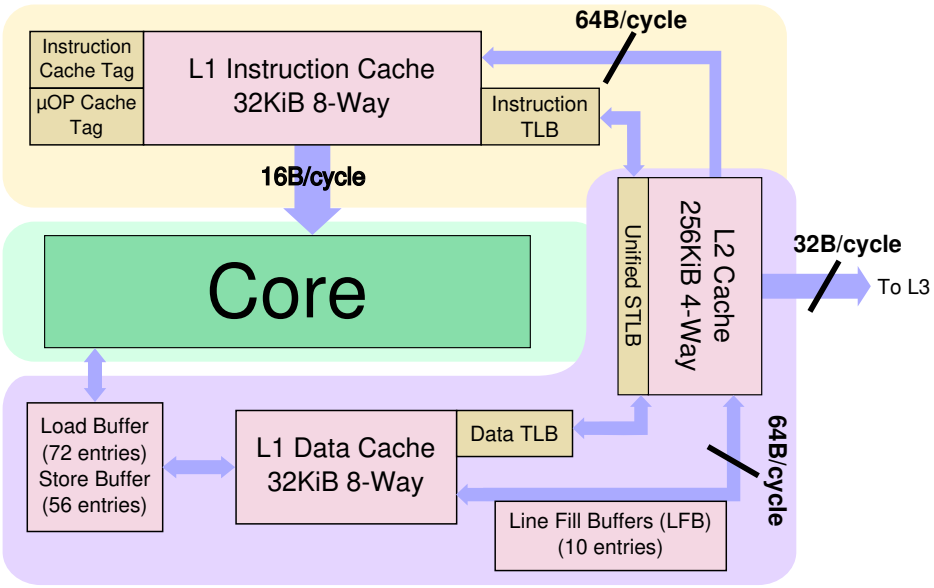


Figure 4.3.: Simplified view of an Intel Skylake core memory system. From [177]

measure event B), or the benchmark must be run several times depending on how many events are measured. As such, it is a first hindrance that can be circumvented, but not in a practical manner given next problem. For single core events like L1 and L2 caches hits and misses, `perf` is an accurate solution. However if we want to measure L3 or even DRAM events, then the counters are shared by all CPUs and processes. This means that we won't be only measuring our benchmark application, but also the entire system, i.e. all other processes as well as the operating system. This sharing process of PMUs limits precise counting of events for L3 and DRAM loads and stores but also prevents parallel execution of different benchmarks on the same machine.

Moreover, if we take into account the prefetchers in the architecture, then the counters are biased based on the access pattern of the application. Indeed, the prefetched data will not be counted as a miss when the CPU requests it as it has been prefetched. But some prefetchers may also alter directly the counters and this is unfortunately insufficiently documented. According to [181], the L1 ICache can miss several times because of prefetchers and thus make L2 accesses smaller than L1 ICache misses which sounds illogical. Other architectural improvement units such as the L1 Line Fill Buffer (LFB) (Figure 4.3) may as well cover up some part of the real L1 misses and L2 accesses. The L1 LFB purposes is to track misses in L1 and prevents multiple misses to the same line to be serviced multiple times from the L2. Similarly, the Load Store Queue (LSQ) of the CPU can hide some accesses that are still visible from the CPU point of view. Last but not least, the documentation is quite often obscure and incomplete, both software and hardware ones. The list of events given by the command `perf list` does not even specify L2 cache events, but only L1 and Last Level Cache (LLC) which is in fact the L2 cache. The Intel's manual [180] list over 188 different event types for Skylake architecture but testing all the 2^{16} possible events yields 20712 events returning non zero which implies that around 20000 of

them are either undocumented or hidden. Thus, we do not know what those counters count and figuring it out would be time consuming.

For power measurement, some CPUs provide Running Average Power Limit (RAPL) interface to measure real time power consumption of a complete CPU, which for multicore includes all cores. Power measurement is divided in power planes that furnishes in-situ measures for different parts such as the whole core, DRAM or Graphic Processing Unit (GPU). First of all, just like performance counter, RAPL interface is not standardized. It heavily depends on the CPU model but also on the Operating System (OS) kernel version. Moreover, as it can only measure power for a really coarse part of the system, it is complicated to determine which portion comes from the process we want to benchmark. A work around could be something similar to correlation power analysis used in cryptography to extract meaningful data from multiple (e.g. 1000) runs. The same apply for DRAM power plane that is also shared by the system.

In conclusion, performance hardware monitoring counters are great tools that are hindered by many obstacles. These include a limited number of counters that requires running the same benchmark multiple times, a system-wide interaction when monitoring event that can be triggered by any process preventing parallel work and an abstruse documentation. From our own testing, only core level counters are reliable, i.e. L1 and L2 event counters from the Intel's documentation.

4.2.3. Simulation platforms

4.2.3.1. General definitions

We define the precision level of simulation platforms as either *cycle accurate* or *instruction accurate*. Cycle accuracy is the most detailed high level simulation where each simulation step is exactly one clock cycle. This allows very fine grain analysis of systems but at the cost of a very time consuming simulation. Instruction accurate tools where each instruction is a simulation step are less detailed but way faster than cycle accurate tools. However, it cannot be used, for instance, to model precisely a CPU internal pipeline. Everything that is shorter than an instruction, or that results from an instruction is often seen as instantaneous or to take exactly one cycle. It is still possible to model these events, but they cannot be modelled exactly in relation to the CPU, i.e. detailed model for those cannot be kept in synchronisation with the CPU.

On one hand, simulation will emulate a, possibly detailed, CPU model and any other relevant subsystems. This can include, in addition to a base CPU model that is mandatory, buses, caches, memories, IO, etc. In return, we have in depth insights of what is going on in the system at the price of a very slow simulation and a high memory usage which includes the benchmark, its data and all simulated parts of the system. Simulation allows really fine grain study with cycle accuracy or coarse grain analysis with instruction accuracy. The slowdown compared to a real hardware execution ranges from a few thousands to millions times slower. Nonetheless, it offers flexibility of use allowing to simulate any system and hardware on any other system, i.e. it is not restricted to the hardware and operating system it is being run on. For instance, it is

Table 4.2.: Simulation vs Instrumentation

	Simulation	Instrumentation
Cross-Platform	Yes	No
Accuracy	Cycle or Instruction	Instruction
Include OS	Yes	Yes*
Slowdown	1000× to 10^6×	1× to 1000×

* With root privileges

possible to simulate a benchmark intended to run on system A with hardware X on system B with hardware Y even though it would not be possible in reality. Simulators often propose two execution modes: a System Emulation (SE) mode where all system calls are translated to the host system and only the benchmark is simulated; a Full System (FS) mode where a full operating system is simulated alongside the benchmark. In our case, full system is of little interest as we do not change any operating system part and are only interested in the user level memory accesses.

On the other hand, instrumentation is different from simulation as it is tightly coupled to the underlying system and hardware it is running on. It inserts instrumentation code within the benchmark to call some routines at regions of interest. For instance, it is possible to insert a call before or after every branch to study the impact of branch predictors in a CPU. In our situation, we can insert code for every memory access and also for each instruction as they all access at least themselves, i.e. each instruction is loaded from memory. The instrumentation code models an architecture, similar to simulation, including the subsystems of interest. Contrary to simulations, only instruction accuracy is reached as it does not offer any insight to what is happening during the execution of an instruction. The benchmark thus runs natively on the hardware with less slowdown compared to simulation, mostly depending on the instrumentation routine code, up to a thousand times slower than a real execution. Instrumentation can attach to an existing process to trace interesting part of an application that cannot be done with simulations, but they can also save a state and restore it to skip initialisation phase for instance. A summary of differences between simulation and instrumentation is presented in Table 4.2.

4.2.3.2. Considered platforms

We now list all the platforms that we have considered for our system study. The selected platforms must be extendable to support our proposed C-SRAM solution and be able to accurately depict memory systems.

QEMU [182] is an emulator using Dynamic Binary Translation (DBT) to translate a binary for processor A to run it on processor B. It supports user mode emulation (that we denote as SE) and FS mode for emulation. Supported targets include x86, ARM,

RISC-V which are the main existing core architectures and it also supports different OSes like Linux, Windows and MacOS. It provides a device emulation supports for Non Volatile Memory (NVM) Express, an interface for permanent storage through PCI buses, and NVM subsystems which can be of interest for us. To translate original binary to host code, it uses pre-compiled micro operations to move data and perform basic operations to make up for the original instructions. However, QEMU is a huge machinery that has a steep learning curve and requires a lot of development effort. Using DBT, QEMU achieves quite high-speed simulation.

gem5 [183] is a cycle accurate simulator supporting SE and FS mode. Contrary to QEMU, it does not rely on binary translation but is a modular event driven simulator. It supports many targets including x86, ARM and RISC-V, although RISC-V support seems incomplete. gem5 is a famous simulator in the system architecture research field but its memory system is flawed, which is a red flag for my experiments. For instance, memory accesses are not reliable as they are not constant for the same benchmark when ran multiple times and have a huge error rate [184]. In my experiments, caches accesses were varying without any reason and when using invalidated lines first, it still showed an increase in miss rate which is illogical. On the development effort, I estimate it to be medium to hard. For the simulation speed, as it is a full system simulator and has to manage a lot of subsystems, it is thus very slow.

ArchSim [57, chapter 7] is a team made simulator for RISC-V architecture using SystemC-TLM. It is mostly focused on low level hardware modelisation with transaction accuracy, somewhat intermediate to cycle and instruction accuracy. It was developed for evaluation of IMC-Near-Memory Computing (NMC) architecture so would fit our purpose quite well. However, as it aims for high accuracy, it is seemingly slow for our target case with big data usage. As it is team made, I estimated the development effort to be low to moderate.

Other considered simulators are **ZSim** [185] that is based on Pin [186]. It focuses on fast manycore simulation with detailed Out of Order core model and sacrifices some accuracy using memory accesses reordering. **Sniper** [187] is also a multicore simulator centered on detailed core model. Finally, **LLVM** [188] is a high level (for hardware people, low level for software people) compiler framework that generates Intermediate Representation independent of the target CPU. It can be used as a 0-order evaluation method [189] but its high level nature makes it a bad choice for our study.

Pin [186, 190] is an instrumentation tool using DBT, akin to QEMU. It only supports x86 CPU but works on all three major OSes: Linux, Windows and MacOS. It uses a modular approach by separating the instrumentation engine, which is closed source, from the instrumentation routine that we write. The translation engine provides a lot of instrumentation functions to test various kind of instruction, from branches to memory accesses and including Single Instruction Multiple Data (SIMD) instructions and more. It receives a list of assembly instructions to instrument along with the code to insert (instrumentation routine). Pin also enables us to modify the instrumented benchmark data on the fly to simulate our C-SRAM instructions. From the examples available, it is quite trivial to use and to adapt to our needs, hence, I estimate the development effort to be low to moderate. As it uses instrumentation, it is a high

Table 4.3.: Platforms overview

Platform	Memory interfaces	Accuracy	Speed	Development effort
QEMU [182]	Callback functions	Instruction	High	High
gem5 [183]	Ruby	Cycle	Slow	High
ArchSim [57]	TLM sockets	Transaction	Medium	Moderate
ZSim [185]	None	Instruction	Very high	Moderate
Sniper [187]	N/A	Cycle	Slow	Moderate
LLVM [188]	None	Instruction	Medium	Low
Pin [186]	Callback functions	Instruction	High	Low

speed *simulation* tool. **Pin** thus suits our needs quite well, and we choose this platform for our experiments.

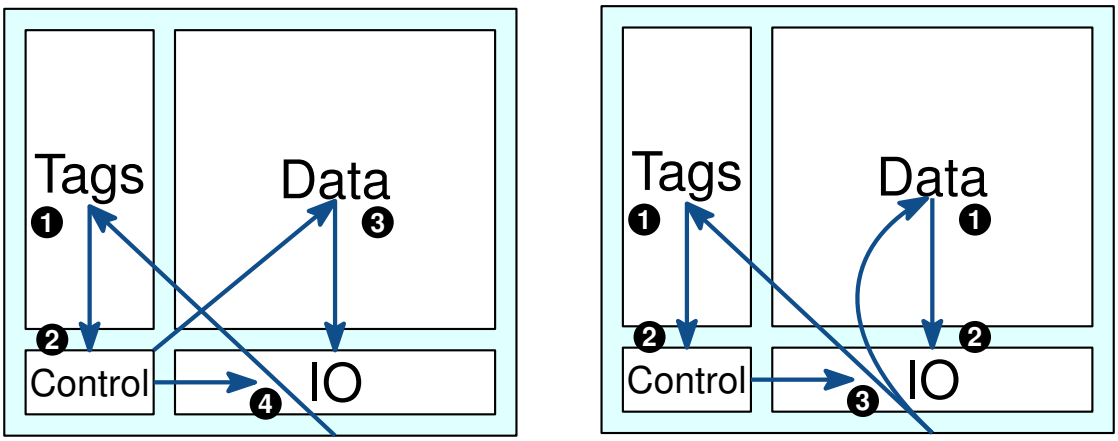
We provide an overview of the different existing platforms in Table 4.3. Note that as IMC-NMC is an emerging field of research, programming models and interfaces to compute in memory are not standardized yet. This requires more work to adapt the platforms to incorporate memory computing. Moreover, most platforms are dedicated to compute centric architectures which can be problematic to switch to data centric architecture. It can also be a *bad* surprise to realize that the chosen platform is actually not suited for our needs and that is why we picked Pin as it fits most of our requirements. We are also not going to develop a model as detailed as what can be provided by years long existing platforms such as gem5 or QEMU.

4.3. Hardware model tools

To describe accurately the different memories in a real system, we base our work on hardware model tools that are each specialized in a different circuit so that we only have to connect them together. For the caches and NVM, we use NVSim [191] which is based on Cacti [192]; for the DRAM, we perform a comparison between different existing tools to figure out which one suits our needs the best and is also reliable. In the final part, we discuss how we extend our work presented in Chapter 3 to handle wide IO or larger memory sizes for our C-SRAM.

4.3.1. NVSim

NVSim [191] is a "*circuit-level model for NVM performance, energy, and area estimation*" supporting new emerging NVM technologies, including Phase Change Memory



(a) Sequential access **(b)** Parallel access

Figure 4.4.: Cache access types

(PCM), Magnetic Random Access Memory (MRAM) and Resistive Random Access Memory (RRAM). It provides a very detailed interface with plethora of parameters allowing fine control over bitcell, transistor, form factor with array dimensions and routing type. Design optimisation goals gives an easy way to explore different layouts depending on the chosen importance of some metrics (latency, energy, area, etc.). Details can be found in the referenced paper [191, 192]. As it is based on Cacti [192] from HP Labs, which was primarily designed for cache parameters estimation, we can also use it to extract the cache parameters we use in our experiments. For all the extracted data, we use a 22 nm node so that energy and timing parameters match those of our C-SRAM. We report the caches design parameters in Table 4.4. L1 caches are designed for optimized latency, i.e. both data and tag banks are accessed simultaneously (Figure 4.4b), as they must serves data quickly and are expected to have a high hit ratio. Difference in L1 and L2 access reside in the routing and the place where data selection occurs, IO for L1, matrices for L2. L3 cache is optimized for area as it has the greatest capacity and will still perform better than the DRAM. It also uses low standby power transistors instead of high performance ones as the leakage power is dominant given the memory capacity. L3 access is sequential (Figure 4.4a) as it may have a high miss ratio so it makes less sense to perform data bank access in the same time as tag bank. For the Static Random Access Memory (SRAM) cell, we use the default value provided by the tool which itself relies on the ITRS Mastar model values [193].

For the NVM as long term storage, we choose to use Phase Change Memory (PCM) as it is already being used in commercial solution and is the most appropriate technology for this usage. We select a 20 nm PCM [194] that has all the necessary data to be fed into NVSim. It features a 1.8 V 1D1R (see Figure 1.14c) PCM with a reset current of 100 µA during 100 ns pulse and a set pulse of 150 ns using same current. The R_{on} and R_{off} of the cell are set to respectively 10 kΩ and 1 MΩ although they are not stated in the original paper. Similarly, we set the voltage drop through the access diode to 0.3 V. Device type is set to Low Operating Power (LOP) as in the reference. To avoid some

Table 4.4.: NVSim's parameters used to design caches

(a) Design targets

	L1	L2	L3
Capacity	32 kB	256 kB	8 MB
Associativity	8-way	4-way	16-way
Access mode	Fast	Normal	Sequential
Optimisation target	Latency	Latency	Area
Device	HP	HP	LSTP

(b) Extracted parameters

		L1	L2	L3
Area		0.0252 mm^2	0.162 mm^2	5.06 mm^2
Hit	T	3.43 ns	24.3 ns	48.2 ns
	E	20.4 pJ	52.0 pJ	1.48 nJ
Miss	T	0.880 ns	0.783 ns	5.11 ns
	E	24.1 pJ	56.5 pJ	1.62 nJ
Leakage		37.0 mW	262 mW	816 mW

extravagant results, we reduce the memory size down from 1 GB to 128 MB. Internal structure is forced to have subarrays of 2048 rows per 4096 columns. Other options are not set and the mode is set to explore all the possibility. We prune the results to eliminate those with ridiculously small or big energies and latencies, flat shape or all matrices active together. This results in hundreds of possible solutions and we take the closest one to the median of energies and latencies shown in Table 4.5a.

To make a 4 GB PCM, we use 32 cuts of the original 128 MB PCM. To this effect, we use a prior work based on SRAM tiling [195] to account for the delay and energy incurred by the tiling and the wiring. Coincidentally, 32 is the optimal number of cuts when using big memory size by tiling smaller memories. From this previous work, we see that we need to apply an energy overhead of 37 % and a timing penalty of 85 % to the 128 MB PCM to get the numbers of our final 4 GB. This corresponds to the 512 B IO in Table 4.5b. All inferior IO size use the same amount of timing and energy which is actually wrong as the energy would be smaller when shrinking IO. Latency on the other hand should remain constant. However, for the IO of our simulation going up to 4 kB, we need to widen the original 128 bits IO. To do this, we apply a 2× widening factor for energy as we read or write twice the original amount of data, and a pessimistic 70 % for timing each time we need to double the IO. The final metrics obtained are given in Table 4.5b with the corresponding widen factor applied.

We model a 3 levels cache hierarchy and a 4 GB Phase Change Random Access Memory (PCRAM) with wide IO. The technological parameters for the PCM are extracted from [194] and fed into NVSim [191] to make a 128 MB PCM. To extend it to a 4 GB PCRAM, we use tiling method and apply appropriate malus to account for it [195]. The final energies and timings used in our experiments are laid out in Table 4.5b for the PCM and in Table 4.4b for the caches.

Table 4.5.: Energy and latency of the selected PCRAM

(a) Original 128 bits IO 128 MB PCRAM

	Energy	Latency
Read	0.875 nJ	142 ns
Write	4.88 nJ	512 ns

(b) Scaled 4 GB PCRAM

IO Width	Read		Write		Widen Factor
	Energy	Latency	Energy	Latency	
512 B	38.5 nJ	263 ns	215 nJ	949 ns	0
1 kB	77.0 nJ	447 ns	429 nJ	1.61 µs	1
2 kB	154 nJ	759 ns	858 nJ	2.74 µs	2
4 kB	308 nJ	1.29 µs	1.72 µJ	4.66 µs	3

4.3.2. DRAM

DRAM is a complex part in a computer system as it is the core memory but does not uniquely receive read or write command. It is based on a convoluted state machine [196, Figure 2] where each transition must respect timing in regard to previous and next transitions, hence the reason why there is a memory controller that takes up a large part of a CPU (see Figure 1.12b). To model the DRAM, multiple tools exist similar to simulation platforms. We present them succinctly here.

Ramulator [197] is a DRAM simulator whose source code is publicly available on Github [198]. It can handle many different DRAM standards from the old DDR2 up to recent GDDR5 as well as some academic work such as **RowClone** [84]. It simulates high speed states machines with transition timing and rules specified in lookup tables. One of its most useful features is that it can transform access traces into DRAM command traces which can be fed to other simulator that only accepts those (DRAMPower and VAMPIRE for instance). **VAMPIRE** [199, 200] is a trace based "*DRAM power model based on the power consumption of real DRAM modules*". It is probably the most accurate DRAM simulator of all as it is backed by real measurements of commercial devices. The experimental setup presented in the paper also shows that there is a great variability between different vendors but also among the same model. This is due to the fabrication process. **DRAMPower** [201, 202] is a model also backed by real measurements and circuit level simulations. Contrary to VAMPIRE, it provides its own scheduler allowing users to feed it with access traces rather than DRAM commands. It also accounts for intrinsic variations of power estimation using Monte-Carlo analysis. **NVMain** [203, 204] is more focused on NVM memory and their NVM Express interface that is similar to DRAM's one. It extends DRAM simulation for NVM with endurance, hard faults simulation and Multi Level Cell (MLC) for Flash devices but can still be used for normal DRAM operation. **DRAMSim2** [205, 206] is a cycle accurate model that comprises of its own DRAM controller and scheduler, the data storage part and takes into account the bus as well. It is validated against manufacturer's Verilog models. **DRAMSys** [207, 208] is a model based on SystemC/TLM that computes timing, power

Table 4.6.: Comparison of different DRAM simulation tools

Tools	Access or DRAM trace	Power output	Timing output	Others
Ramulator [197, 198]	Both	No	Yes[a]	Can transform an access trace into a DRAM command trace
VAMPIRE [199, 200]	DRAM	Yes	Yes[a]	Uses measurements from commercial devices. Miss `refresh` command...
DRAMPower [201, 202]	DRAM	Yes	Yes[a]	
DRAMSim2 [205, 206]	Access	Yes[b]	Yes	
DRAMSys [207–210]	Access	Yes	Yes	Windowing mode available
NVMain [203, 211]	Access	Yes[c]	Yes[a]	Very slow for huge traces

a. Reported in cycles only b. Reported per window only c. Reported unit is "mA*t"

and also temperature profiles of DRAM modules.

The considered tools are listed in Table 4.6 with their accepted input and outputs. Tools that only take a DRAM command trace as input must be preceded by a call to **Ramulator** to transform a normal access trace. We perform a comparison study in Section 4.4.4.2 that also includes our own work. This study is to assess the accuracy levels of these tools for both timing and energy estimation.

4.3.3. C-SRAM

As presented in Chapter 3, our developed C-SRAM has a fixed width of 128 bits. To enable wider IO, we can of course just put multiple ones asides using the same control and address buses to make up a wide data bus. But this does not hold as there is also a constraint on the C-SRAM total size. If for instance we want to make a 4 kB wide IO C-SRAM but with a total size of only 16 kB, we cannot just put 128 cuts as each one has a minimum size of 2 kB with an IO of 128 bits; that would yield a 256 kB C-SRAM. Similarly for the biggest size we target of 8 MB and the widest IO of 4 kB, we would need 256 cuts of 32 kB with a width of 16 bytes. However, this solution is not the optimal one as demonstrated in a team's previous work [195, Figure 7.b]. In order to optimize both the timing and energy access costs, we need to tile them in a mesh grid. Using this previous tiling work, we extract how we should tile our C-SRAMs and how to scale up the energy and timing costs for an access (Table 4.7). We also take into

Table 4.7.: Tiling timing (T_{tf}) and energy (E_{tf}) factor overhead

Size	16 kB	32 kB	64 kB	128 kB	256 kB	512 kB	1–8 MB
N_{cuts}	8	16	16	32	32	32	64
E_{tf}	10 %	10 %	19 %	19 %	37 %	37 %	37 %
T_{tf}	30 %	30 %	56 %	56 %	85 %	85 %	85 %

account the widening effect by dividing the wanted IO width with the number of cuts and by the base IO width of 16 B:

$$Widen_{factor} = W_f = \frac{Width}{N_{cuts} \times Width_{base}}$$

If W_f is superior to 1, then we cannot reach the desired width using only our cuts so we must widen them by W_f. From our exploration in Chapter 3, we also extract meaningful widening penalty to apply to each widened cut. For cuts with a width ranging from 8 B to 32 B, we observe a 30 % energy penalty (E_{Wf}) and a 10 % to 20 % timing penalty (T_{Wf}) when doubling the IO width. We thus choose to apply a 30 % energy penalty and a 15 % timing penalty each time we double the IO width while keeping the same total size. Putting it all together, we obtain the final formula:

$$E = (E_{base} \times E_{Wf}^{W_f}) \times E_{tf} \times N_{cuts}$$
$$T = (T_{base} \times T_{Wf}^{W_f}) \times T_{tf}$$

The timing is not multiplied by the number of cuts because they are all accessed at once, so only the energy depends on the number of cuts. The energy being multiplied by the number of cuts is actually an error as only $\frac{N_{cuts} \times W_{base}}{W}$ are accessed simultaneously, which can be all the cuts if we had to widen them. So we are actually over estimating C-SRAM's energy. Figure 4.5 shows the energy and timing for all the width and total memory size we used. The lines are the results of the widening while the concentrated points at the start of the line would stack vertically if we did not activate them all at once.

We have presented multiple tools to accurately model the different parts of a real system with all its memory components. We use NVSim [191] for the caches and the NVM. We perform a tool comparison for DRAM in Section 4.4.4.2 after introducing our platform. We finally presented an extension of our work on C-SRAM to model wider and larger memories using tiling method.

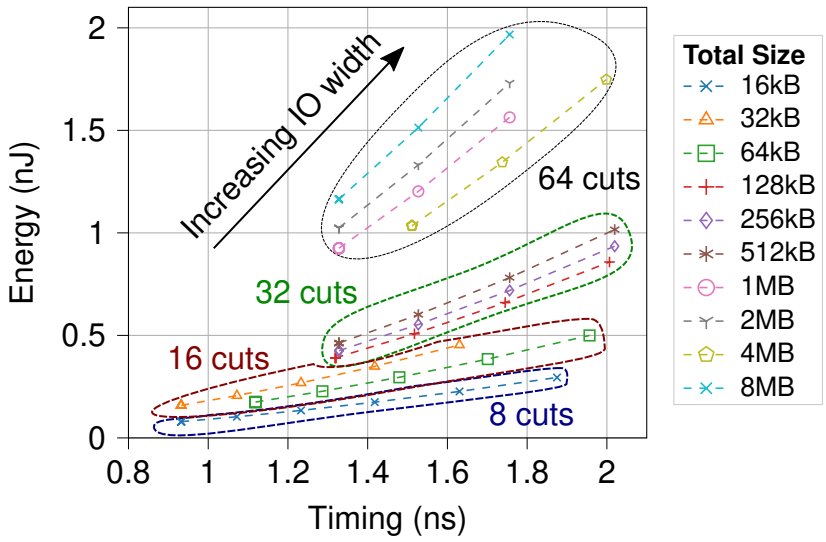


Figure 4.5.: C-SRAM tiling energy vs timing access costs

4.4. Platform

Our platform is based on an instrumentation tool (see Section 4.2.3): Intel Pin [186, 190]. We develop several instrumentation routines and we connect everything from C-SRAM development in Chapter 3 to hardware model tools in section 4.3. This is the keystone to this manuscript.

As previously described in Section 4.2.3, Pin is an instrumentation tool that allows to instrument each type of instruction. It inserts callbacks to user specified functions before running the program. We are only interested for our main study in memory accesses which include loads and stores instructions but also all instructions as each one is loaded from memory. With Pin, this is simply done as shown in Listing 4.1. We retrieve effective addresses and sizes of memory accesses for all predicated on instructions. Predicated on prevents false access for instructions such as conditional move that depends on a predicate unknown at the time of instrumentation. In order to track accurately where data comes from, we also need to know the addresses which comes from the program itself. Pin once again gives us a way to do so and we can retrieve addresses boundary for every loaded library and for the main program itself. This will be used in the second version of our platform in Section 4.4.3.

4.4.1. Software interface for benchmarks

First, we introduce the software interface that is needed to communicate with the proposed platform. This was already partly presented in previous publications [57, 144, 155] as well as in Chapter 3, but we extend it to 64-bit architecture and add some more instructions.

In a real system, there are only two ways to communicate to a device, whatever type it is (keyboard, memory, etc.): either through a memory mapped interface or through

Listing 4.1 Instrumentation code in Pin

```
// Callback functions
void MemAccess(uint64_t addr, uint32_t size, bool accessType);
void InsRead(uint64_t addr, uint32_t size);
// all instruction fetches access I-cache
INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)InsRead,
    IARG_INST_PTR,                      // instruction address
    IARG_UINT32, (UINT32)INS_Size(ins),   // instruction size
    IARG_END);
if(INS_IsMemoryRead(ins) && INS_IsStandardMemop(ins)) {
    // only predicated-on memory instructions access D-cache
    INS_InsertPredicatedCall(ins, IPOINT_BEFORE, AFUNPTR(MemAccess),
        IARG_MEMORYREAD_EA,                     // read effective address
        IARG_MEMORYREAD_SIZE,                   // read size
        IARG_UINT32, CACHE_BASE::ACCESS_TYPE_LOAD, // indicate this is a read
        IARG_END);
}
```

additional wires (I2C, USB, PCI, etc.) that are controllable by the CPU with special instructions. In the first case, it is the responsibility of the Memory Management Unit (MMU) to send the messages to the corresponding devices through either the system bus or through additional wires without needing CPU special instructions. For our simulation purposes, it is way easier to go through a memory mapped interface that would work on almost all platforms thanks to its independence from the target CPU.

To extend the Instruction Set Architecture (ISA) previously defined in Figure 3.5 to 64 bits, we widen all the addresses fields from 16 to 28 bits. With a linesize of 16 B, this allows to address a total of 4 GB of data. The opcode remains on 8 bits and the 3 low order bits of the address field are zeroed to prevent misalignment issue as we now store a 8 B word. The remaining upper bits are not used to limit the size of the memory space to use (Figure 4.6). Indeed, memory mapped interface needs to reserve said memory and the OS might refuse to give it to a process if this space is too large. To prevent this problem, we use `anonymous` mapping which directs OS that the mapping should only be allocated virtually, i.e. memory is only page allocated when we read or write to a page. This mapping is performed on the benchmark side and the platform will only catch the writes, address and data, to this mapping and interpret them according to the ISA. To ease the benchmarks development, we define vector types that are exchangeable so that a single code can be recompiled for different vector sizes (Listing 4.2). We employ C macros to encapsulate all defined functions and each macro or function takes the destination address and the operands addresses if applicable (Listing 4.3). We also provide equivalent SIMD instructions encapsulated within these macros that allows easy switching between reference SIMD and our C-SRAM version. However with Pin, the benchmark and the platform each have their own memory space, i.e. address a does not have the same data for both the benchmark and the platform. Thankfully, Pin provides adequate functions to access benchmark's data and to modify it as well which allows us to emulate the received instruction and to modify the benchmark's memory consequently (Listing 4.4).

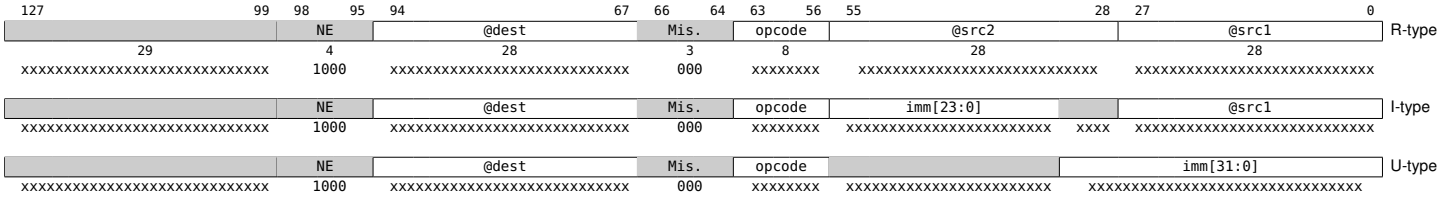


Figure 4.6.: Extended ISA for 64-bit system. Upper 64 bits are address bits and lower 64 are data bits.

Listing 4.2 C vector types

```
// defined when using SIMD AVX-512
#define VECTOR_SIZE 64
#define VECTOR_TYPE __m512i
// defined when using CSRAM
#define VECTOR_SIZE 1024
#define VECTOR_TYPE __CSRAM_Line
typedef int8_t __CSRAM_Line __attribute__((vector_size(VECTOR_SIZE)));
typedef union {
    VECTOR_TYPE v;    // provided for SIMD compatibility
    int8_t      i8  [VECTOR_SIZE/sizeof(int8_t  )];    // easy access to 8-bit elements
    int16_t     i16 [VECTOR_SIZE/sizeof(int16_t )];
    int32_t     i32 [VECTOR_SIZE/sizeof(int32_t )];
    int64_t     i64 [VECTOR_SIZE/sizeof(int64_t )];
} CSRAM_Line_t __attribute__((aligned(VECTOR_SIZE)));  // force alignment of vector
```

Listing 4.3 C macro example: 8-bit vector addition

```
#define vadd8(_dest, _src1, _src2) _dest = _mm512_add_epi8(_src1, _src2) // SIMD AVX-512 macro
#define vadd8(_dest, _src1, _src2) do { /* CSRAM varying vector length macro */                \
    /* create 128 bits word through macro  */                                                  \
    unsigned __int128 _storeword = _MAKE_CSRAM_ISA_RTYPE(CSRAM_OPC_ADD8, _dest, _src1, _src2); \
    /* get 64 address bits */                                                                  \
    volatile uint64_t*_storeaddress = (uint64_t*)((uint64_t)(_storeword >> 64));               \
    /* get 64 data bits */                                                                     \
    uint64_t _storedata = (uint64_t)(_storeword);                                              \
    *_storeaddress = _storedata; /* store instruction, i.e. send it to C-SRAM */               \
    asm("" ::: "memory");        /* memory fence, tell compiler that memory values changed */  \
} while(0)                       /* syntax quirks */
```

Listing 4.4 Emulation of received instructions

```
uint8_t csram_row[3][VECTOR_SIZE];    // local buffer
// retrieve ISA fields through some macros
uint64_t dest = _get_csram_address(RTYPE_DEST, writtenAddr, writtenData, 0, 1);
uint64_t src1 = _get_csram_address(RTYPE_SRC1, writtenAddr, writtenData, 0, 1);
uint64_t src2 = _get_csram_address(RTYPE_SRC2, writtenAddr, writtenData, 0, 1);
uint32_t opcode = _get_csram_opcode(writtenAddr, writtenData);
PIN_SafeCopy(csram_row[0], src1, VECTOR_SIZE);    // copy src1 into local buffer
PIN_SafeCopy(csram_row[1], src2, VECTOR_SIZE);    // copy src2 into local buffer
pin_compute_switch(opcode, csram_row[0], csram_row[1], csram_row[2]); // compute instruction
PIN_SafeCopy(dest, csram_row[2], VECTOR_SIZE);    // copy local buffer to dest
```

4.4.2. First version with hard coherency

4.4.2.1. Reference implementation

First, we describe the reference implementation that we benchmark against and that also serves as a codebase for the C-SRAM version. We start by defining a base class to represent caches with a Least Recently Used (LRU) replacement policy which is a simple array of sets which themselves are array of tags. Tags contain an address, a valid and a dirty bit to track modification. Our implementation of caches is write-back with store allocate policy. L2 and L3 are neither inclusive nor exclusive. Each access to memory goes through L1 cache which first look for valid data. When data is missing, a cache line is allocated according to LRU policy and previous data, if existent and dirty, is written back to the upper level, as shown in Algorithm 4.1. This operation is repeated until a cache, DRAM or ultimately SCM gets a hit. Only dirty data is written back as non dirty data always has an up to date copy in either DRAM or SCM. We do not model hit/miss buffers and write buffers.

Algorithm 4.1 Cache access pseudocode

```
1:  Returns tuple of boolean to indicate hit/miss and replaced tag if miss
2:  function CacheAccess(addr, accessType)
3:      if ¬Find(addr) then
4:          oldtag ← Replace(addr, accessType)      ▷ Insert addr in cache and retrieve
    replaced tag
5:          if oldtag.dirty then
6:              return false, oldtag          ▷ If oldtag is dirty, it needs to be written-back
7:          else
8:              return false, 0                   ▷ null address is equivalent to no address
9:          end if
10:     end if
11:     return true, 0
12: end function
```

4.4.2.2. C-SRAM

To handle C-SRAM in our platform, we ensure global coherency between caches, DRAM and C-SRAM, i.e. we make sure that data that can be modified in caches by the CPU and in C-SRAM are always up to date. To enforce this, any data written by the CPU is invalidated in the C-SRAM and reciprocally. We use DRAM as the merging point for data. When a C-SRAM instruction is met, we first check if C-SRAM already has the data. If so, then it has not been modified by the CPU as it would have been invalidated and the data is thus up to date. Otherwise, the source operands are flushed back from the caches to DRAM first. To flush them, only dirty data is written back to minimize data movement, but data goes through each cache level from L1 to L3. Then, we check for DRAM and in final resort, the data comes from the SCM as described in

Algorithm 4.2. For the destination address, it is simply invalidated from all caches as it is now stale. Finally, Pin allows us to access benchmark memory and to modify it to simulate the received instruction as shown in Listing 4.4.

Algorithm 4.2 Simplified CSRAM instruction

Ensure: data coherency when executing C-SRAM instruction

```
1:  function LoadSrcToCSRAM(addr)
2:      if ¬csram.Has(addr) then
3:          caches.FlushDirty(addr)                ▷ Flush back to DRAM
4:          if dram.Has(addr) then                 ▷ DRAM has uptodate data
5:              csram ← dram[addr]                 ▷ Transfer from DRAM to C-SRAM
6:          else
7:              csram ← scm[addr]                  ▷ Transfer from SCM to C-SRAM
8:          end if
9:      end if
10: end function
```

4.4.3. Improved version with soft coherency and real disk accesses

The first version of the platform suffers from a few problems:

- It does not track real disk accesses performed by system calls;
- Not all data is coming from the SCM, it can be created *ex nihilo* in DRAM with memory allocation;
- Swapping data from DRAM when it is full to SCM is counted incorrectly;
- Hard coherency makes it somewhat slow.

To remedy those, we develop a second version of our platform to allow more accurate SCM tracking. To improve simulation speed, we also decide to hand the coherency to the programmer using software based coherency, similar to what is done with GPU.

4.4.3.1. Modifications to reference

First, we need to handle system calls as they give information on which addresses come from real files and which are just anonymous mappings not backed by any file. The executed instructions all come from either the binary itself or from dynamic libraries that are loaded at program start. Pin provides function to get executed files (main program and shared libraries) addresses. We note that in our quite old Linux distribution, executables addresses are always at the same offset (`0x40000`) but dynamic library have variable addresses. Those are loaded through the `mmap` system

call which returns the address to which it mapped the file. To track all of these, we use a repertory which contains all segments (i.e. start and length) of address that are known and mapped and their origin (file or heap/stack). If a requested address is not in this repertory, it is either in the heap or in the stack both of which are allocated directly in DRAM without loading from SCM. In this case, the requested address is most likely written as read data is uninitialized. `mmap` does not require any coherency handling as no data is used before the system call, i.e. it comes from the SCM if it is a file backed memory map or is created *ex nihilo* in DRAM otherwise.

The other two main system calls that we handle and which access SCM are `read` and `write`. In case of a `read`, we invalidate all corresponding addresses in both caches and C-SRAM as correct data now lays in DRAM. For a `write`, we write back corresponding addresses from caches and C-SRAM into DRAM without invalidating them as they are still valid there. We also mark them non dirty in caches and C-SRAM but mark DRAM as dirty for our third problem which is swapping. Swapping occurs when DRAM is full and we need to evict some data back to SCM. Not all data need to be written back. For instance, if data originates from disk and has not been modified, then it does not need to be written back as the disk still have the original data, so we can safely evict those. Using the repertory, we can find which data can be safely replaced. We extend the repertory with a timestamp entry to also account for oldest data first with a minimum threshold randomly set to 10^9 instructions (LRU like policy). Before swapping, we must bring back any data left in the caches or in the C-SRAM back into DRAM. If no data can be evicted, i.e. all are either dirty or not file backed, then we pick the oldest one and write it to SCM. When the data is needed again, it is brought back into DRAM and associated entry in repertory is updated. Now, we also need to handle system calls referencing data that is currently swapped. When `read`ing into data that is swapped, we can just invalidate corresponding entry in the repertory, i.e. mark them as non swapped. `Write` swapped data needs to first bring it into DRAM before writing it back into SCM as they do not share the same addresses.

4.4.3.2. Software based coherency

Software based coherency is realized using similar method with C macros to allow the benchmark to communicate with the platform. We define eight coherency primitives, four for caches and four for C-SRAM, which takes an origin address and a length to apply to:

- Invalidate which deletes all data in the specified range;
- Writeback which writes all dirty back into DRAM;
- Flush that does both writeback and invalidation, provided for convenience;
- MovDRAM to move data from C-SRAM to DRAM and its counterpart MovC-SRAM.

When using data in the C-SRAM that was previously modified by the CPU, the programmer must first writeback data from caches to DRAM before issuing a MovCSRAM. In the current setup, C-SRAM writeback actually performs the same operation as MovDRAM. For all these additions to our platform, we take care at the fact that each one can now trigger another one. For instance, any coherency instruction might trigger swapping data in or out of DRAM. System calls also require special attention and we decided that they should perform their own coherency. `Read` first invalidates data in caches and in C-SRAM in addition to the repertory entry if part of the specified range is swapped. `Write` writebacks all data back to DRAM before issuing data to SCM. The resulting platform collects all memory events, namely loads, stores, writebacks and coherency checks.

I have developed a simulation platform based on Intel Pin [186] instrumentation tool. This platform models a 3 levels cache hierarchy, a DRAM main memory and a top level NVM for the reference implementation and with our C-SRAM inserted at different places for our future experiments in Chapter 5. This platform supports our own IMC implementation with memory mapped devices. It also features system call handling, disk accesses and swapping to accurately account all memory and SCM accesses. The first version of the platform led to results published in DATE 2021 [212].

4.4.4. Caches and DRAM validation

We have presented all the tools we use, we now perform a light validation of our developed platform against other references. Section 4.4.4.1 is about cache hierarchy while Section 4.4.4.2 deals about DRAM simulators and their accuracy against either ground truth or what can be realistic, notably power.

4.4.4.1. Caches validation

To validate the caches behavior, we compare ourselves against the ground truth provided by the hardware counters (Section 4.2.2). We also add an estimation using a basic analytic model to show how it poorly performs (Section 4.2.1). We use a classic square matrix multiplication using `ikj` loop order given in Listing 4.5. We make the hypothesis that matrices size are infinite which means that there is no reuse from the second innermost loop iteration (`k`) to the following. Matrix **C** is accessed N^3 times for both reading and storing (`+=` operator), the first access is a compulsory miss while the following ones are hit until we reach a cache line boundary, so we have a total of $\frac{N^3 \times (B-1)}{B}$ hits and $\frac{N^3}{B}$ misses where B is the block size in matrix elements. For storing, as we have already loaded the current element, we have 100 % hit which equals to N^3 store hits. Matrix **B** is also accessed N^3 and the same logic applies which yields the same number of hits and misses. Finally, matrix **A** is accessed N^2 and has a 100 % miss rate as we supposed no reuse from outer loops iterations. Using this hypothesis, we can deduce that this model predicts a 100 % miss rate for L2 and L3 caches.

To extract hardware counters as our ground truth, we use those provided by the Linux kernel and some others extracted from the Intel reference manual [180]. For the L1 cache, we only use counters provided for convenience by the Linux kernel and that we can use after having compiled the previous code without optimisation. We use the following counters for each cache level:

L1 `L1-dcache-loads`, `L1-dcache-load-misses`, `L1-dcache-stores` and `L1-dcache-store-misses` all provided by the `perf` tool utility;

L2 `r0124` which measures load hits, `r0224` for load misses, `raa24` for store and load misses, `rff24` for all L2 accesses. We then subtract `r0224` from `raa24` to get the number of store misses and we subtract both `r0124` and `raa24` from `rff24` to obtain the number of store hits;

L3 `LLC-loads` and `LLC-stores` for L3 as I could not find any corresponding counter for load/store hits/misses in the documentation or they were not available on my machine.

To mitigate the natural variations, we average each counter on 10 runs on an otherwise idle machine. The results are presented in Figure 4.7. Looking at our model for the L1 cache in Figure 4.7a, we see that our platforms shows very accurate results except for store misses. We largely underestimate them for big matrices which is actually a weird behavior. Indeed, a study of the generated assembly code shows no store that can miss. This once again demonstrates that hardware counters are not reliable. One possibility is that a context switch from our process to another one provoked those misses but it would also offset other events which is not observed. Analytical model (Figure 4.7b) slightly underestimates load hits which comes from our matrices actually fitting in the L1 cache for a loop iteration which allows more hits in following iteration.

For L2 cache (Figure 4.7c), we largely overestimate load misses and underestimate load hits. Another strange result given by hardware counters is that the sum of L2 accesses is twice the number of misses in L1 which is counterintuitive. The Intel documentation states that L2 accesses include L1 prefetchers which could explain our results with our platform that do not model those. Finally for the L3 cache (Figure 4.7d), we got pretty accurate numbers for loads although it seems the accuracy decrease with bigger dataset. Stores are overestimated but actually, our platform yields

Listing 4.5 Square matrix multiplication using `ikj` loop order

```
for(i = 0; i < N; ++i)
    for(k = 0; k < N; ++k) {
        tmp = A[i][k];
        for(j = 0; j < N; ++j)
            C[i][j] += tmp * B[k][j];
    }
```

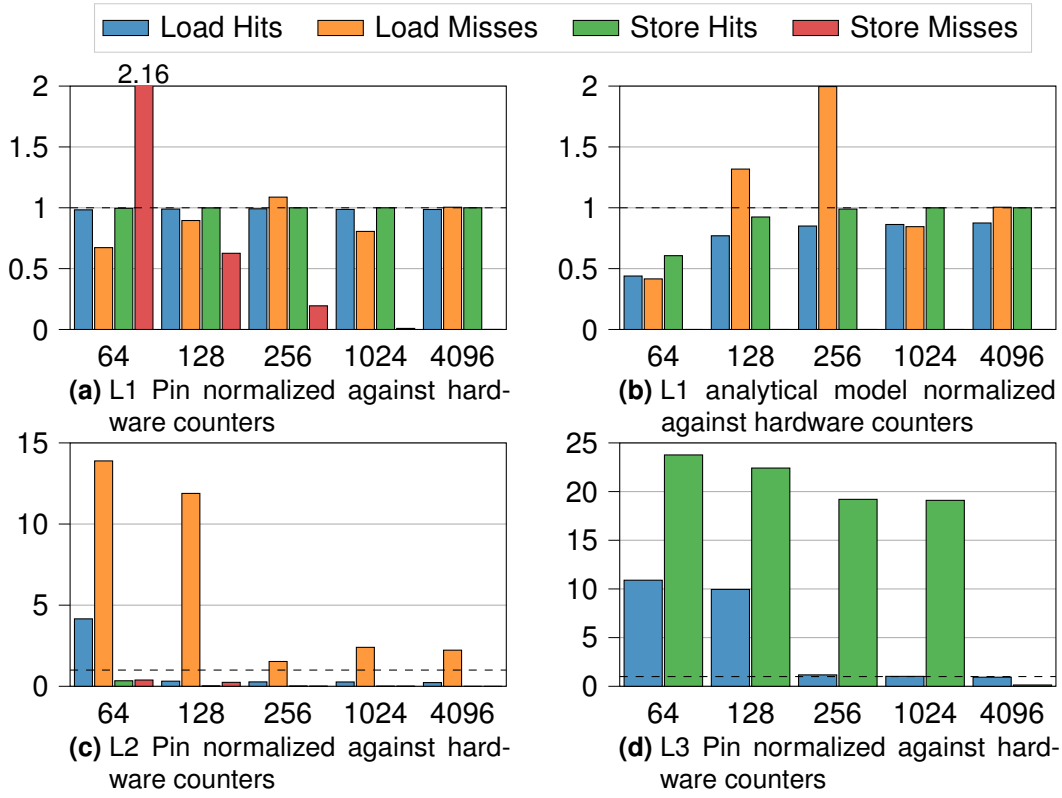


Figure 4.7.: Our platform normalized against hardware counters for cache events. X-axis is matrix size and y-axis is ratio between our own work and hardware counters.

a thousand while `perf` returns only a few tens which explains this gap. This is not really a problem as those few stores are overwhelmingly below the number of loads accurately represented which is around several billions. Timing and energy are thus not impacted by this slight error.

We compared our model with the ground truth provided by hardware counters. We showed that we accurately model cache behavior, especially in regard to the quantity of event. We also saw that hardware counters can have unpredictable behavior that is unsuitable for our research purpose.

4.4.4.2. DRAM tools comparison

We perform a comparison study of the different tools in this part. . We extract two types of traces from our benchmarks, the first one containing all the data accesses performed by the CPU, the second one containing only the accesses going to DRAM including page swapping from disk. Each access is reported in the traces using the current timestamp, the accessed address, the access type (read or write) and the access size in byte (for page miss, this is 4 kB). We then feed these traces to the different tools.

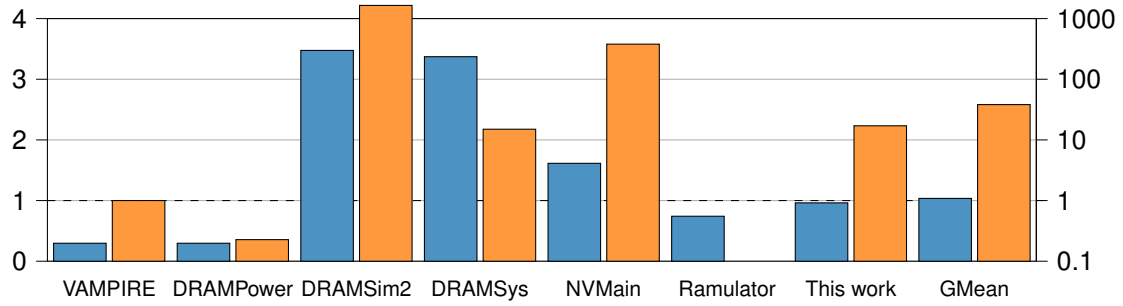


Figure 4.8.: DRAM tools timing (left y-axis) and energy (right y-axis) estimation (geometric mean of all benchmarks). Timing is normalized against benchmark execution time and energy is normalized against VAMPIRE.

For all the tools, we model a 512 MB single channel single rank DDR3 module divided in 8 banks. Each bank is itself subdivided in subarrays of 8192 × 1024 (row-column) with an output width of 8 bits per array. Burst length is set to 8 so that a full burst is equal to 64 bytes, i.e. a full cache line. We use a closed row policy and the frequency is set to 1066 MHz. Most parameters are taken from existing Micron module configuration files provided by the aforementioned simulators. For some simulators, we had to tweak these parameters to allow the simulation to run. Each simulators has its own conception of how subarrays and IO width are linked, so for some we had to multiply the width by 8 to match subarray's width.

All the previous DRAM tools (Section 4.3.2) are fed with traces of the benchmark suite we use and we exploit the timing and energy or power figures these tools output to perform the comparison. To better estimate the differences between these tools, we also measure the real execution time using a similar DDR3 memory but with different frequency that we can then scale accordingly to the frequencies ratio (real and experimental). The real execution time however includes all parts of the benchmarks, including caches and disk accesses, so it is overrated. Nonetheless, it is still a good comparison as this enables us to rule out completely off the mark tools; some of them over evaluate timings by an order of magnitude. The results are averaged and plotted in Figure 4.8. From this, we can see that we are the most accurate, but as the real timings are over rated, we can deduce that **DRAMPower**, **VAMPIRE** and **Ramulator** are more accurate. **DRAMSim2**, **DRAMSys** and **NVMain** are the least accurate of all the tools. Both **DRAMPower** and **VAMPIRE** use the same backend to compute timing hence their equal timings.

If we consider that **VAMPIRE** and **DRAMPower** are the most accurate in terms of timing, we can make the assumption that they are also the most accurate in terms of energy or power. As **VAMPIRE** is the most recent and is based on real measurements from off-the-shelf commercial DRAM components, we consider them as our baseline for power comparison. As shown in Figure 4.8, some tools over evaluate (compared to **VAMPIRE**) by more than 1000× but what is more surprising is the wide distribution spanning 4 orders of magnitude. One criteria of discrimination is to compute the

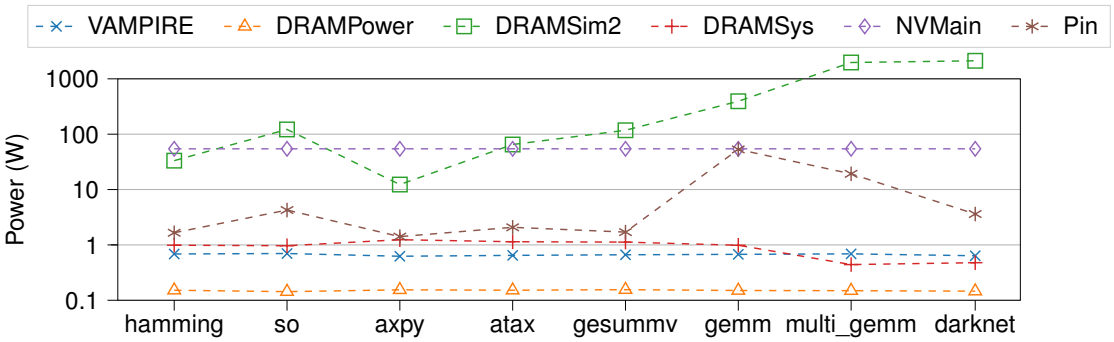


Figure 4.9.: Computed power reported by different tools using our benchmark suite

actual power, i.e. to divide the energy by the timing using both figures reported by the tools; this is exposed in Figure 4.9. This clearly demonstrates that some tools are completely off the grid by reporting power that is over tens of watts for a single DRAM memory stick. Obviously, this is not realistic. Several tools report an almost constant power regardless of the memory activity of the benchmark such as **VAMPIRE**, **DRAMPower** and **NVMain**. Note that none of the tools account for the IO current drawn when serving data. One possible explanation for **DRAMSim2** outstanding reported power might be the input trace fed to it. Indeed, when a page miss occurs, a 4 kB page is fetched from the disk to the DRAM but this is reported as multiple 64 B accesses all happening simultaneously. So it is possible that this specific tool tries to satisfy all the incoming requests at once leading to these huge power outputs. Note that this is clearly not the correct behavior as other tools seem to schedule correctly the DRAM commands from the input trace. **Ramulator** for instance is used to create the DRAM commands trace from our input access traces for the **VAMPIRE** and **DRAMPower** tools. On the other hand, **Ramulator** does not compute power or energy per se, so it cannot be used for our platform.

From this study of different DRAM simulation tools, we observe a lot of variations between tools themselves in both energy and timing which corroborates the fact that DRAM is a complex part to model and accurately represent. The almost constant power output from several tools leads us to use a fixed energy and timing cost per access for simplification.

Conclusion

In this chapter, we presented the motivations behind the design of our platform to simulate our selection of benchmarks at system level. We compared several existing simulation platforms which either lack precision in the memory subsystem or are too slow for our purposes. We decide to design our own simulation platform based on **Pin** [186], an instrumentation tool. Our platform is able to accurately represent the data exchange in the memory hierarchy and can be used to compare existing High Performance Computing (HPC) architecture. It supports system call that meddles

with memory or transfer data to or from disk. To accurately gather energies and timings consumed by each level of the memory hierarchy, we model the latter in specialized hardware tools such as **NVSim** [191] or **VAMPIRE** [200]. We also extend our work presented in Chapter 3 on the C-SRAM to allow wider IO and larger memory sizes. A software interface is designed to communicate between benchmarks at the application level and our C-SRAM. Our final workflow is summarised in Figure 4.10.

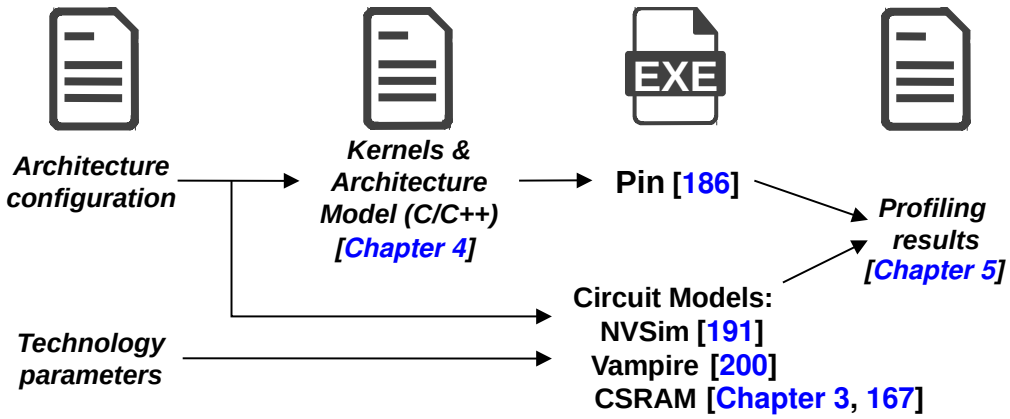


Figure 4.10.: Workflow used in this thesis. We describe an architectural representation in C++ within Pin [186] to model a reference architecture and an IMC one with our C-SRAM. Benchmarks are coded in C with macro based ISA to communicate with the Pin platform. On the hardware side, we retrieve technology parameters for the caches, the PCM and our C-SRAM and simulate the memory hierarchy with appropriate tools: NVSim [191] for the caches and the PCM, VAMPIRE [200] for the DRAM and extend our own work [Chapter 3, 167] for C-SRAM. When combining both, we obtain profiling results (energy and timing) analyzed in Chapter 5.

5. IMC/NMC Computing Architectures

Et toc! Remonte ton slibard, Lothar!

— Perceval IN *Kaamelott* BOOK II, EPISODE 53 « *L'Absent* »

A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away.

— Antoine de Saint-Exupéry

Using the previously defined methodology and the platform we developed in Chapter 4, we study the impact of introducing In-Memory Computing (IMC) with our proposed C-SRAM solution in the memory hierarchy. We consider different placement propositions and different data management policies. We show that our solution greatly reduces energy consumption and outperforms a reference vector Central Processing Unit (CPU) architecture up to 400× energy reduction and 272× speedup for cubic kernels. For linear kernels, we reach a best case of 50× and 38× while quadratic kernels achieve a best case of 25× and 15× respectively. Globally, we also demonstrate that our proposed solution does not increase Non Volatile Memory (NVM) accesses, and especially write accesses.

Contents

Using the platform developed in Chapter 4, we compare the integration of our Computational SRAM (C-SRAM) solution presented in Chapter 3 against a vector architecture using 512-bit Single Instruction Multiple Data (SIMD) instructions. As shown in Chapter 2, state of the art performs memory computing at cache or Dynamic Random Access Memory (DRAM) level but did not study it at the Storage Class Memory (SCM) level where most data lives. Some solutions do exist [103, 108, 125, 130] but are not taking into account memory weariness that is intrinsic to all long term storage memories. Moreover, global data movement through the complete memory hierarchy is also often discarded. The purpose of this chapter is to study the different improvements we can get by computing at different places in the memory hierarchy as shown in Figure 5.1 following different scenarios:

- **Scenario NVM ❶**: An independent C-SRAM is placed between the SCM and the DRAM. Data may come from both memories but rewrites all go through DRAM to prevent wearing out the SCM early. This scenario can have multiple data management policies such as fine grain transfer or page mode transfer that are both analyzed in this work.
- **Scenario NVM ❷**: The C-SRAM is used as a Computing Row Buffer (C-RB) solution in NVM, as already presented in [212] and detailed more precisely in Figure 5.13.
- **Scenario DRAM ❶**: The C-SRAM is now placed between the caches and the DRAM. Once again, data may come from both memories but writes are also bidirectional in this case (i.e. from C-SRAM to both caches and DRAM).
- **Scenario DRAM ❷**: The C-SRAM is used as a C-RB solution in place of the DRAM row-buffer.

Both scenario NVM ❷ and DRAM ❷ aim for a simpler hardware integration in a real system as it is easier to modify an already existing chip than to introduce a completely new one. We remind the benchmarks are introduced in section 4.1.

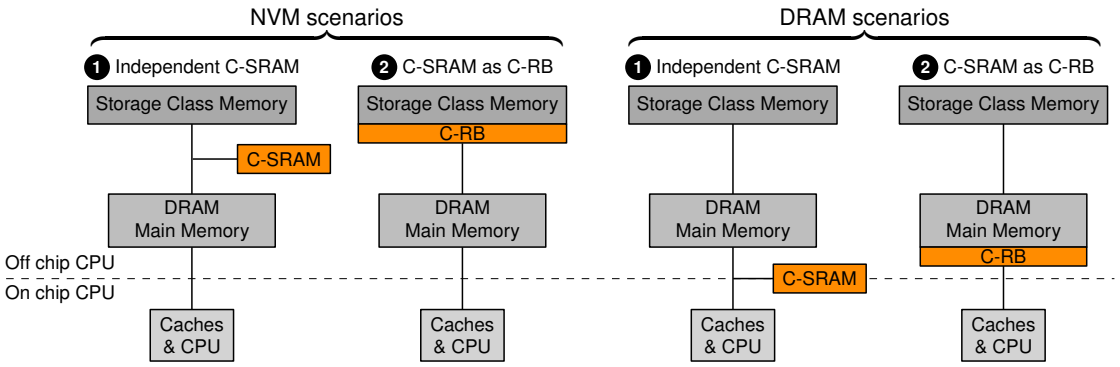
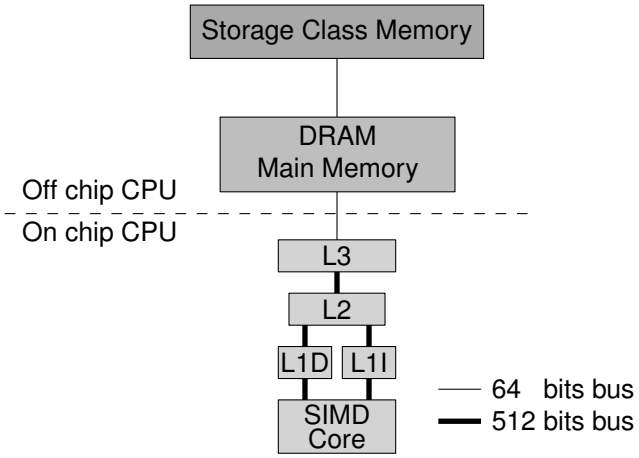


Figure 5.1.: Different integration possibility of the C-SRAM within the memory hierarchy



Memory	Total size & width
SCM (PCM)	4 GB, 4 kB RB
DRAM	512 MB, 64-way 4 kB page
L3 Unified Cache	8 MB, 16-way 64 B block
L2 Unified Cache	256 kB, 4-way 64 B block
L1 I/D Caches	32 kB, 8-way 64 B block

Figure 5.2 & Table 5.1: Reference architecture and memories parameters

5.1. Reference SIMD 512-bit architecture

We present in Figure 5.2 the reference architecture along with the memories parameters. The reference architecture is based on a standard High Performance Computing (HPC) architecture composed of a SCM that serves as a hard disk, a DRAM main memory of 512 MB as well as a 3 levels cache hierarchy. The global memory configuration is given in Table 5.1. Note that memory transfer between different memory levels can either be 64 bits when the transfer occurs off chip or 512 bits between the caches. This is a conventional architecture met in HPC, although there may be a L4 cache in some CPUs. The cache hierarchy is based on the Intel(R) Xeon(R) CPU E3-1240[1] and is composed of two L1 caches, one for instructions (L1I) and one for data (L1D), a L2 unified cache and a L3.

5.2. Computing at the top level in the memory hierarchy

The basic idea of computing close to the SCM is that this is where data is whether the closest and/or originates from. If we take for instance neural networks, all the weights are stored in the SCM, brought into DRAM and then used sparsely by the CPU which moves part of it to caches, back and forth depending on the layer being computed. Images are also stored permanently on non volatile storage but may be brought up through internet connection and thus stored in DRAM as well. By bringing computation directly aside of the data, we minimize its movement and thus the time lost just for the displacement and the energy required to move it. We compare several placement for the C-SRAM close to NVM in scenario NVM ❶ (independent C-SRAM) in Section 5.2.1 and then NVM ❷ (C-SRAM used as C-RB in the NVM) in Section 5.2.2.

[1] ↑This is the CPU of my workstation. I used it to validate the platform with `perf stat` measurements for the caches only. See Section 4.4.4 for more details.

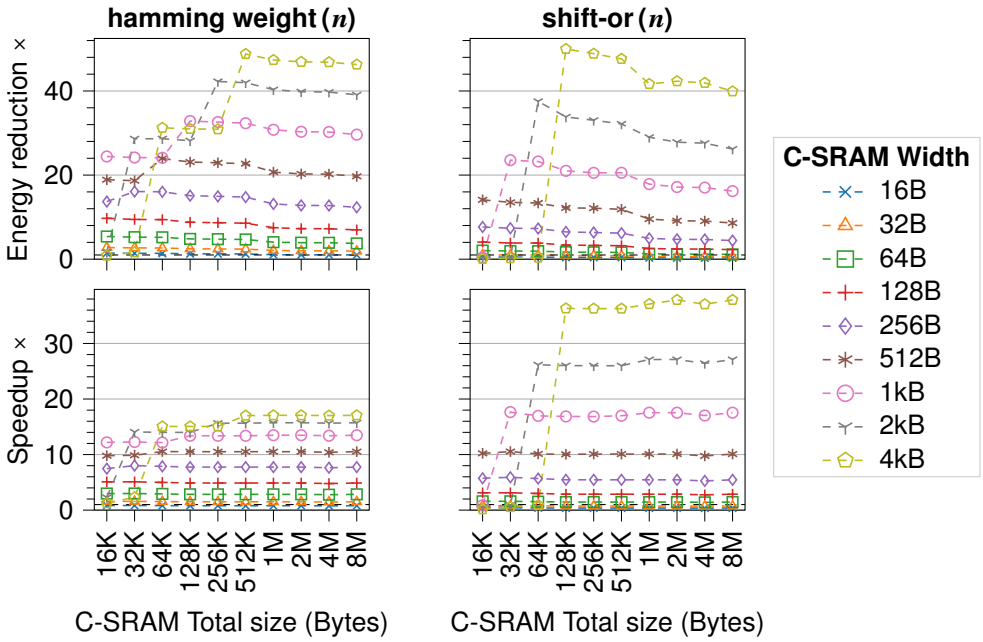


Figure 5.3.: Scenario NVM ❶: Energy reduction and speedup for linear benchmarks normalized against SIMD 512-bit reference (higher is better)

5.2.1. Scenario NVM ❶: Independent C-SRAM between DRAM and NVM

In this scenario, we consider the C-SRAM being somewhere between DRAM and SCM but with no particular coupling and that it has access to the bus linking them. Data transfer can occur between the C-SRAM and both the DRAM and the SCM. DRAM is used as a write buffer from C-SRAM to SCM, i.e. all dirty data from C-SRAM will go through DRAM before being written back to the NVM. We break our analysis on different accesses patterns and benchmarks complexity in terms of computing (see section 4.1). We first analyze the result on a per benchmark basis, then we study the energy and timing distribution to better understand these results. Finally, we show the impact of our solution on the SCM accesses.

5.2.1.1. Linear kernels: *hamming weight*, *shift-or* & *AXPY*

Hamming Weight & Shift-or As shown in Figure 5.3, computing high in the memory hierarchy allows energy reduction up to 50× and speedup up to 38×. We observe that increasing the vector width also increases both the energy reduction and the speedup with a slight saturation starting to appear for *hamming weight*. If we look at the left part of the graphs, especially for the energy reduction, we see steps depending on both vector width and total memory size. This indicates that these benchmarks require a minimal memory size to hold all their frequently accessed variables (constants, array, temporaries, etc.). For instance, if we look at *shift-or* energy

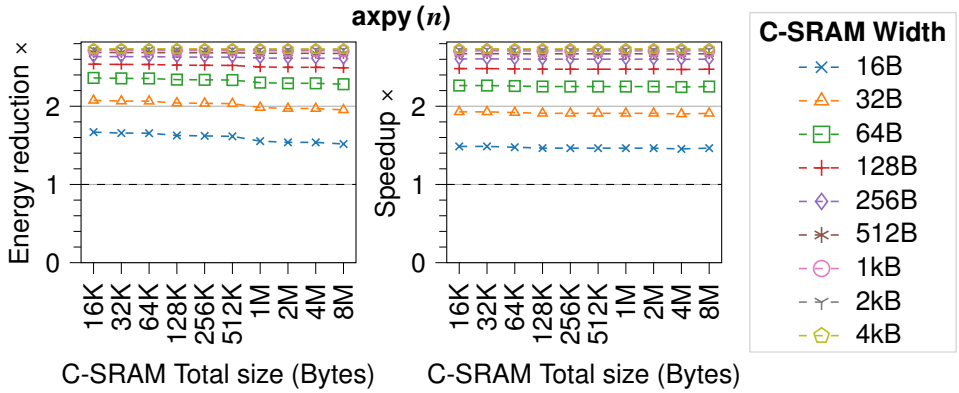


Figure 5.4.: Scenario NVM ❶: Energy reduction and speedup for linear benchmarks with high SCM access rate normalized against SIMD 512-bit reference (higher is better)

reduction, we see that with a 32 kB C-SRAM and a 1 kB vector-width, we get a 23× improvement over SIMD reference. Doubling both these parameters allows a gain up to 37× showing that this benchmark requires between 16 and 32 ($\frac{32\,\text{kB total size}}{1\,\text{kB vector-width}}$) lines to hold its working dataset. The same phenomenon is observed for *hamming weight* with larger steps meaning that this kernel uses some variables only once every n-th iteration. These steps are the reduction operations occurring every 32 iterations for the 8 to 16 bits reduction and every 4096 iterations for the 16 to 32 bits reduction. We also note a slight decrease for both *hamming weight* and *shift-or* in energy reduction after it reached a maximum. Increasing the memory size for both benchmarks is useless and what we observe is just the increased access cost (read/write) along with the increased leak power. For timing, access cost is slightly increased but the larger memory size also permits less writebacks to DRAM and thus a small gain. Both events counteract each other leading to the stagnation observed in both speedup plots. For both benchmarks, increasing the vector width leads to an improvement in both energy reduction and speedup. As these kernels are linear with no data dependency, they can make the best use of a bigger vector. However, for a given vector width, increasing the memory size does not lead to better gains unless there was not enough memory lines available for the variables. For *hamming weight*, the best case is obtained for a 512 kB 4 kB with 50× energy reduction and 17× speedup where as for *shift-or*, the best case is reported at 128 kB 4 kB with also 50× energy reduction and 38× speedup.

AXPY *AXPY* is a linear benchmarks that mainly differs from the previous two in its write intensity. Both *hamming weight* and *shift-or* kernels are completely reducing their input datasets to a scalar or a few numbers of scalar, while *AXPY* writes half of his input dataset back. This is why it has a high part of its energy and timing spent in the NVM hence it can only have a maximum energy reduction and speedup limited to a few units. As it only reads two inputs, multiply one with a scalar and adds them back, it requires only three memory lines, thus the reason there is no stair in this benchmark

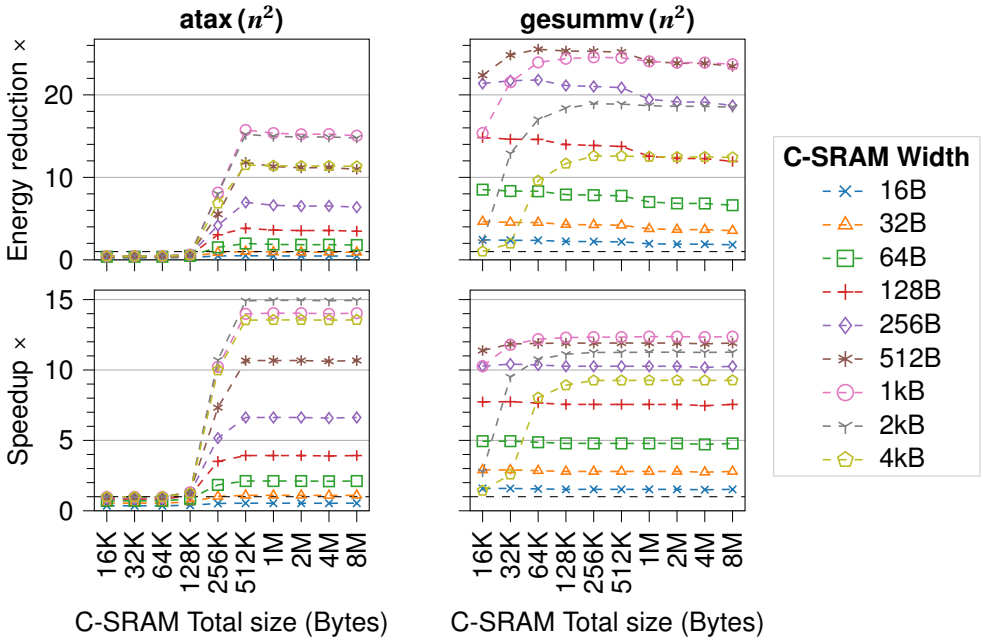


Figure 5.5.: Scenario NVM ❶: Energy reduction and speedup for quadratic benchmarks normalized against SIMD 512-bit reference (higher is better)

compared to other linear benchmarks (Figure 5.3). Although the total memory size has no importance in this case, the vector width allow to lift up a little bit the energy and timing gains until the limit is reached, i.e. when almost all energy and time is spent in the SCM and the C-SRAM. The best energy reduction and speedup of 2.7× is obtained for a 16 kB C-SRAM with a 4 kB vector width.

5.2.1.2. Quadratic kernels: *atax* & *gesummv*

Considering now the quadratic computing complexity benchmarks which involves matrix vector products, we observe quite different tendencies (Figure 5.5). *atax* shows that without a big enough memory size, it even worsens compared to the SIMD 512-bit reference. This benchmark behavior is quite complex as it must stores a temporary vector, performs a reduction loop CPU side and also runs through the same matrix twice. To explain this jump from negative gains to 10× better ratio, we must look at what happens in each memory level. We deduce from Figure 5.6 that this kernels spend most of its time and energy in the DRAM level indicating a back and forth movement of data to and from C-SRAM. This movement then disappear when the C-SRAM memory size is enough to store the working dataset. Also taking into account the access pattern behavior, we note a very poor data reuse in the innermost loop as all accessed variables are used once (temporary vector) before being reused in the second loop. The maximum energy reduction of 16× and speedup of 15× are reached for a 512 kB 1 kB and 2 kB C-SRAM respectively.

If we now consider *gesummv*, we note a different shape where small memories can

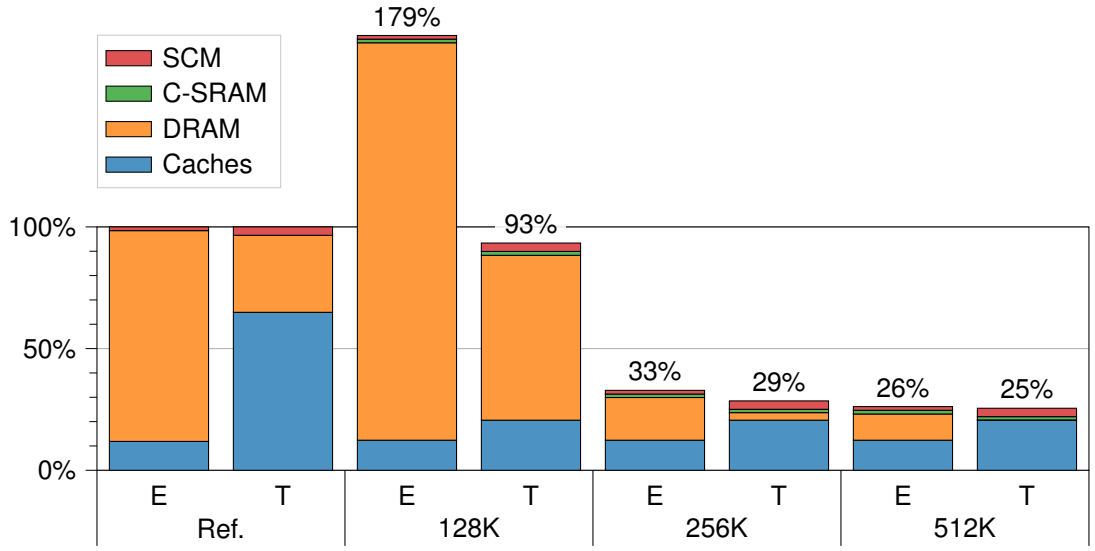


Figure 5.6.: Relative *atax* (to SIMD Reference) Energy and Timing distribution for different sizes and a vector width of 128 bytes (lower is better)

have important gains, but increasing the vector width over a 1 kB leads to a diminution to those gains. Once again, we can use the implementation detail to inspect the access behavior. First, we examine the innermost loop and count only three variables being actively reused while the two input matrices and the input vector are read only once (streaming access pattern). Thus small memories can be enough for this kernel, but increasing the vector width will lead to more conflict for the available space in the C-SRAM. Maximum energy reduction and speedup are respectively 25× for a 64 kB 512 B C-SRAM and 12× for a 256 kB 1 kB C-SRAM.

For both kernels, we now examine the right part of the plots (Figure 5.5) where the memory size is superior to 512 kB. We observe that even if memory size is sufficient, increasing the vector width can reduces both energy and timing improvement. The reduction loop performed along the full vector width by the CPU requires flushing the data back to DRAM then down to the CPU to get the final result. As the vector width increases, this data movement scales up and reduces the performance obtained with smaller vector width. Similarly for linear kernel, we observe a slight decrease in energy reduction when increasing memory size due to more expensive access costs.

5.2.1.3. Cubic benchmarks: *gemm* & *darknet*

We analyze the cubic benchmarks in Figure 5.7 which are mainly matrix-matrix products kernels. To better understands the sudden jump in the *gemm* kernel at 128 kB, we plot, in a similar way to *atax*, the relative energy and timing distribution in Figure 5.8. More than 90 % of the *gemm* kernel reference implementation energy is spent in the DRAM with half of the time spent there as well. As a cubic kernel, some of the input data is read several times, but it is mostly the size of the input matrices that is a problem here. First, the result matrix, which is also an input, is scaled by the β factor in a

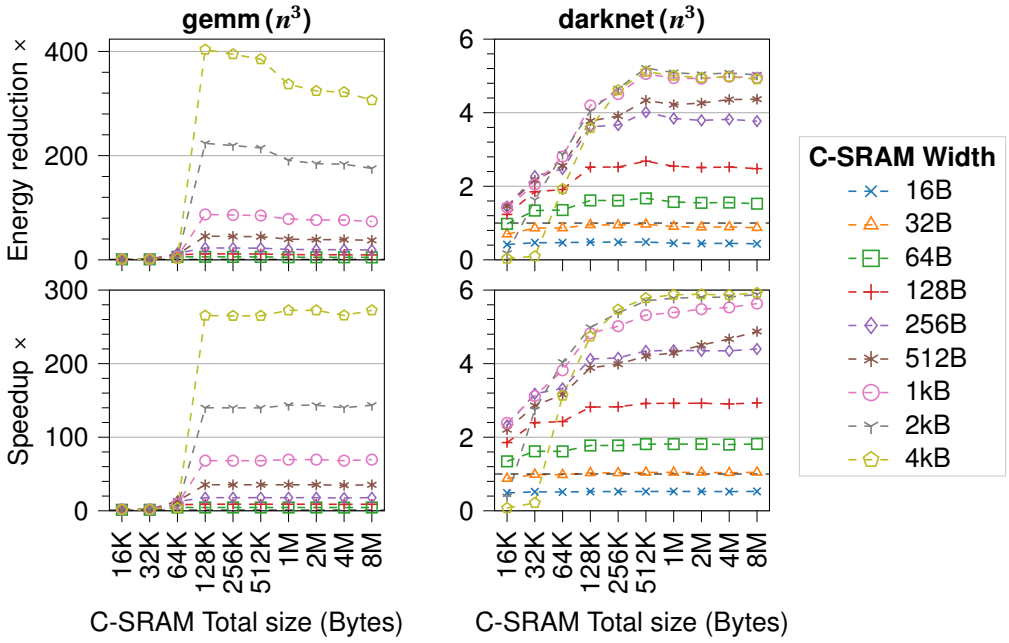


Figure 5.7.: Scenario NVM ❶: Energy reduction and speedup for cubic benchmarks normalized against SIMD 512-bit reference (higher is better)

row major order. Then, the input matrix **B** is also ran through. Both matrices' rows are 32 kB plus the other variables. Thus this kernel requires at least 64 kB of memory size otherwise one of the input matrix will be constantly sent back to DRAM and reloaded at the following iteration. In the C-SRAM architecture, DRAM overwhelms all other costs when memory size is limited, in both energy and timing. For the 32 kB case, DRAM access occurs once every 20 instructions which indicates massive thrashing. After this threshold memory size of 128 kB, we do not observe any more improvement and still detect the slight decrease in energy reduction due to increased access cost and a stagnation for speedup. Increasing the vector width leads to increased energy reduction and speedup with no apparent saturation. There is no reduction loop in the way we wrote this kernel, so it is mainly bound in terms of maximum improvements by the top level memory. We reach the best energy reduction of 403× for a 128 kB C-SRAM with a 4 kB vector width and the fastest speedup of 272× for a 2 MB C-SRAM with also a 4 kB vector width.

Considering that *darknet* is a real case application, we only vectorized functions that are part of its Basic Linear Algebra Subprograms (BLAS) Application Programming Interface (API) (see section 4.1 for details). First of all, we notice that although 80 % of the application is spent doing matrix multiplications (Figure 4.2) similar to all neural networks (Figure 4.1), we are quite far from the gains obtained for *gemm*. Indeed, any computer science student must recall the Amdahl's law which states that, any program or algorithm cannot go faster than its sequential part. As such, we are close to the best speedup we can get by parallelizing the compute intensive part of this application.

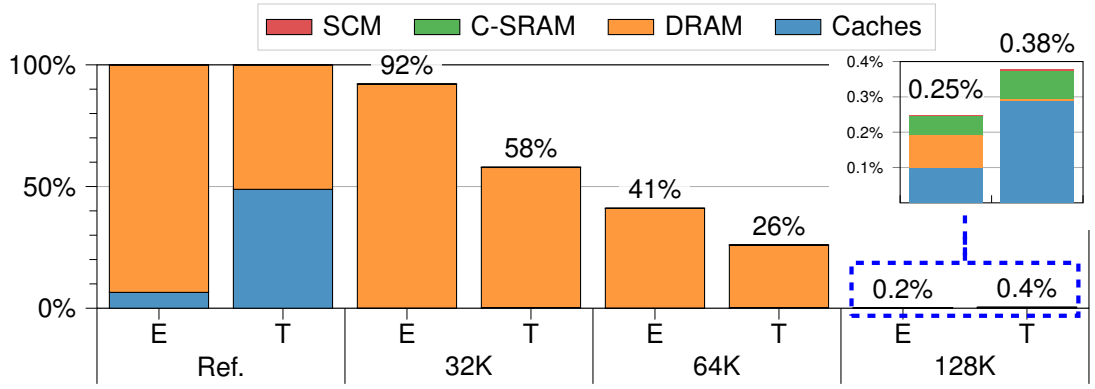


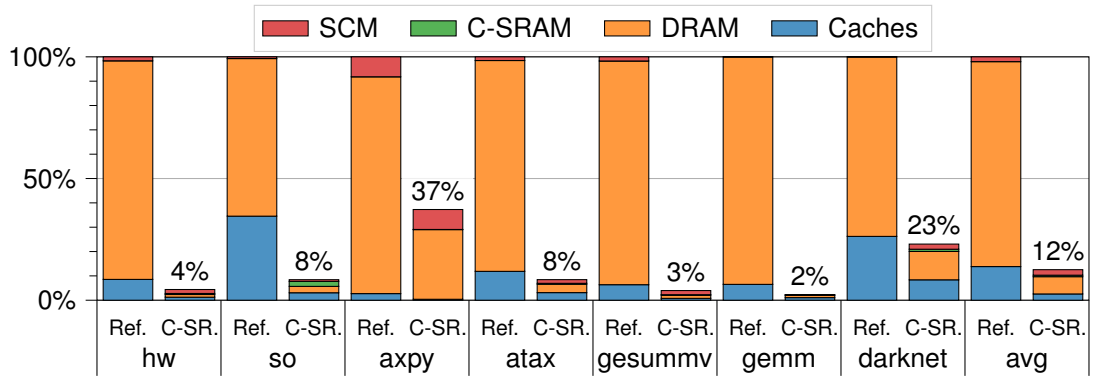
Figure 5.8.: Relative *gemm* (to SIMD Reference) Energy and Timing distribution for different total C-SRAM size and a vector width of 4 kB (lower is better)

We observe that there is no sudden jump and that improvement is more progressive compared to *gemm* with a maximum of 5.2× energy reduction for a C-SRAM of 512 kB 2 kB while speedup is best at 5.9× for a C-SRAM of 8 MB 4 kB.

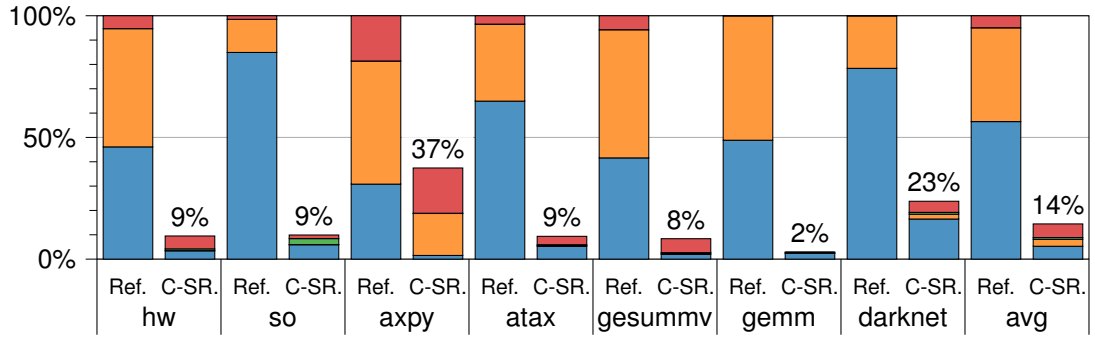
Overall, the tendencies previously extracted from other kernels still stands in our real case application. As such, we do see a saturation when increasing vector size due to the constant exchanges between C-SRAM and CPU. Indeed, only the main kernels function were vectorized, leaving the rest of the application to the CPU including some matrices operations that are branch heavy (`forward_maxpool_layer`), indirect referencing (`softmax`) or includes reduction operations (`forward_avgpool_layer`). Other key functions such as activation are also performed by the CPU. Hence, this saturation is the maximum improvement that we can get by still performing a lot of operations in the CPU. We still note that a 32 B vector size achieve almost as good as the 64 B SIMD reference thanks to reduced data movement.

5.2.1.4. Energy & timing distribution

In the same way as Figure 5.6 and Figure 5.8, we study the relative distribution of both energy and timing for all benchmarks for the best average case which is a C-SRAM of 512 kB with a vector size of 512 B in Figure 5.9. Looking specifically at the energy, we note that DRAM takes a huge part of it in the reference architecture, representing in average 84 %. As we perform computing above DRAM, the read only data are only brought into C-SRAM without going down to the DRAM allowing our solution to tackle most of this energy granting huge improvements in energy reduction, up to 8× in average. The rest of the energy is split between the SCM and the caches. The former being only 2 % of the total while the latter represents around 14 % of the total energy of the reference. The caches energy is also lowered by a factor 5 compared to the reference for the same reason as DRAM. For the timing, the distribution is more even between the DRAM and the caches with the latter representing in average 56 % while the former is up to 39 % leaving 5 % of the total timing in the SCM. The average caches timing is reduced by a factor 10 which is better than caches energy reduction. For



(a) Energy distribution

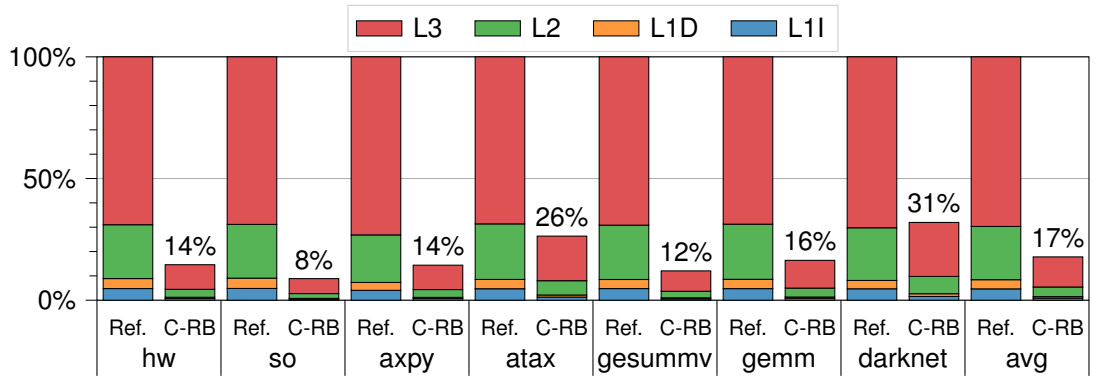


(b) Timing distribution

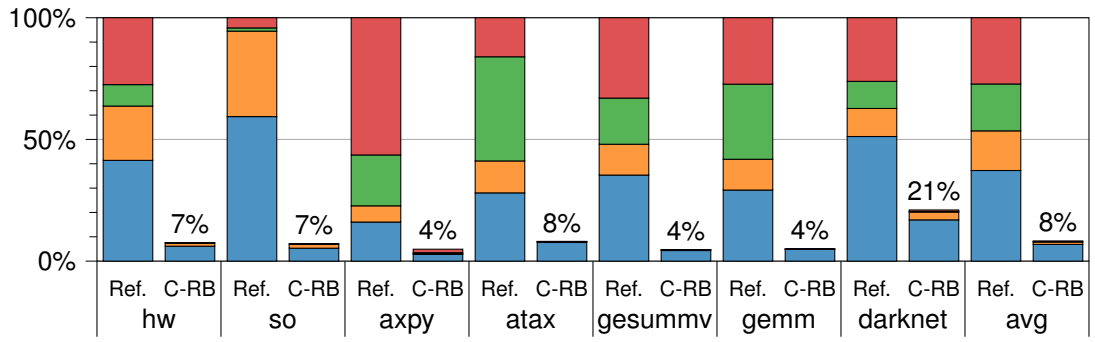
Figure 5.9.: Scenario NVM ❶: Energy and timing distribution for a C-SRAM of 512 kB and vector size of 512 B normalized to SIMD 512-bit reference architecture (lower is better)

both energy and timing distribution, we note that for every benchmark, the SCM part stays constant in our solution. Indeed, as all data originates from this memory, it must still be read at least the same number of times to load the input datasets. It must also be written at least once for the final result.

We give a more detailed insight for the different caches level in Figure 5.10. First we note that L3 makes up the majority of the energy in both the reference and our solution. In the reference case, all the caches are intensively used as all the computation is done by the CPU. But in our solution, only the L1 caches are used as most of the computation is done in the C-SRAM. It is mainly due to the instruction cache and a tiny portion is from the stack variables that reside in the data cache. The energy of both L3 and L2 caches in our solution is largely due to their leakage although they are still a little bit used for kernels with a reduction loop (*atax*, *gesummv*) or complex computing (*darknet*). If we look at the timing distribution (Figure 5.10b), in the reference case, 37 % is due to the instruction cache while it represents 83 % in our solution. As previously said, our solution makes little use of the cache hierarchy, but it still uses the instruction caches as the CPU drives the C-SRAM. We also see a small part of L1 data cache being used in our solution. C-SRAM instructions are computed



(a) Energy distribution



(b) Timing distribution

Figure 5.10.: Scenario NVM ❶: Caches energy and timing distribution for a C-SRAM of 512 kB and vector size of 512 B normalized to SIMD 512-bit reference architecture (lower is better)

on the fly by the CPU and the compiler often optimizes it to store precomputed part of those in the stack, hence the use of the L1 data cache.

5.2.1.5. SCM accesses

Finally, we need to look at the SCM accesses in Figure 5.11, and especially the write accesses as they determine the lifetime of the system. For the reads accesses, there is almost no change as only the input datasets are read. Only for *gemm* we can observe a slight increase of +2 % due to the reuse of the input matrix that does not fit neither in DRAM nor C-SRAM. On the other hands, writes accesses seems to be clearly reduced, but a closer analysis shows that those are not "*real*" writes. The reference case use the DRAM as its main live storage leading to a lot of stale data to stay there unused, including the input datasets. When memory space starts missing, dirty data must be kept and are thus written back to the permanent storage (swapping). These data are often some initialisation data that are used only in the start phase of the benchmark but, as they are the oldest, they got evinced first. In our solution, these dirty stale data compete way less against the input datasets as the read only data stay in the

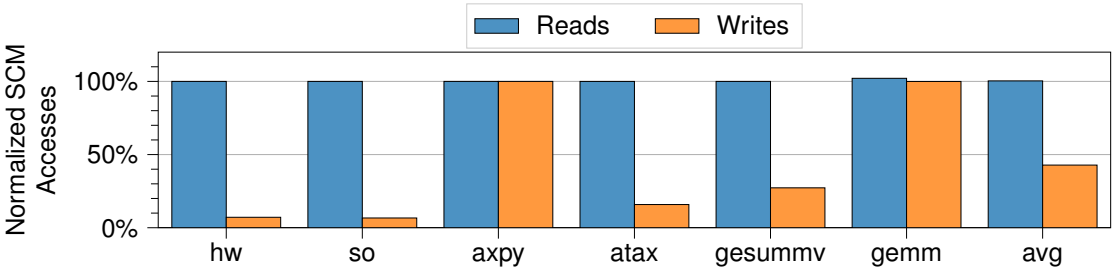


Figure 5.11.: SCM memory accesses for a C-SRAM of 512 kB and vector size of 512 B normalized to SIMD 512-bit reference architecture (lower is better)

C-SRAM and are not written back to the SCM because they are not dirty. To put some actual number on this, considering *hamming weight*, then we only write the final result but 13 pages are written back to SCM. If we take into account only the write that are wanted by the programmer, i.e. actual write to files, then we consider only *AXPY* and *gemm* which are the only write intensive benchmarks. For these two benchmarks, the results are over 100′s MB and are thus more precise for variation in both cases. For both benchmarks, there are almost no variation ($<5 \cdot 10^{-4}$) so we can conclude that our solution does not affect in neither positive nor negative way the endurance and thus the lifetime of the system.

With the current approach, we show that computing close to the NVM offers great improvements in terms of energy reduction and speedup, up to 50× for linear kernels, 25× for quadratic kernels and up to 403× for cubic kernels. On a real case application, although limited by the Amdahl's law, we still reach gains up to 5.2×. For all benchmarks, we demonstrate that our solution does not incur more write accesses to the NVM which is one of the limiting factor in a system lifetime. Figure 5.12 shows the range (minimum and maximum) of achieved values and the average of *all* cases, including the "*bad*" ones. Averaging all benchmarks, we reach a 17.4× energy reduction and a 12.9× speedup compared to SIMD reference.

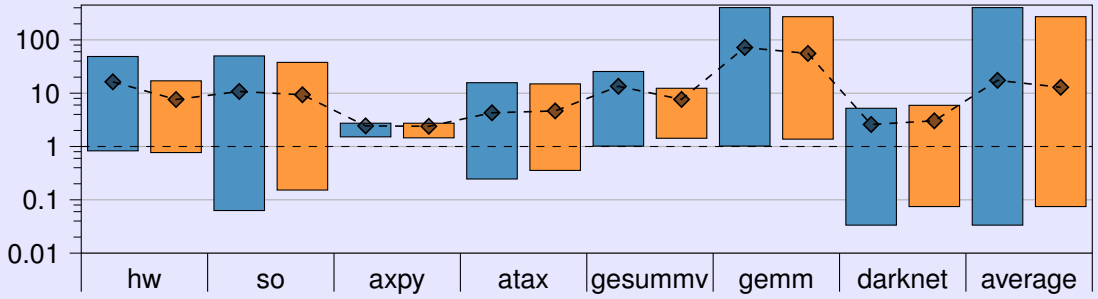


Figure 5.12.: Scenario NVM ❶: Best, worst and average (◆) of *all* cases for both energy reduction ( ) and speedup ( ) normalized against SIMD reference (higher is better)

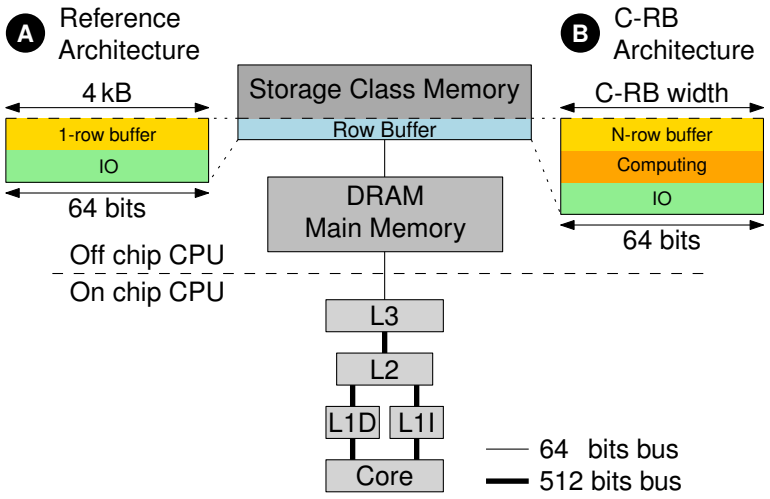


Figure 5.13.: Detailed memory hierarchy. A is the reference architecture where the row buffer acts as a temporary storage for read data. B is the C-RB architecture where the row buffer is turned into a computing element.

5.2.2. Scenario NVM 2: C-SRAM as Computing row buffer

In this scenario, the C-SRAM is tightly coupled to the SCM where it replaces the row-buffer and acts as a multi-line row-buffer (Figure 5.13). Hence, in this scenario, the C-SRAM is labelled Computing Row Buffer (C-RB). The difference in data movement is that all data must go through the row-buffer, even if it is not used by the C-RB. Another key difference is that the NVM width is scaled to the C-RB one, i.e. both have the same width. This means that to load 4 kB to DRAM using a 1 kB C-RB, SCM will perform 4 loads. Energy cost and timing for these cases are discussed in Section 4.3.3.

The results are really close to the previous scenario with the same graph shape when normalized against SIMD 512-bit reference. However, when we plot the normalisation of this scenario versus the former one, we observe that this normalisation is constant for a given vector size so we only plot the average for every vector sizes. The results are shown in Figure 5.14. All benchmarks except the cubic ones present an increase in energy consumption and timing for small vector sizes showing that small vector sizes are slower and require more energy in this scenario. This effect is due to the position of the C-RB as all data must go through it. Moreover, the scaled and matched width between C-RB and NVM creates more accesses to load or write the same amount of data. This effect is clearly visible (bottom row) for the NVM access intensive *AXPY* benchmark (□), where vector width smaller than 512 B are up to 10× less efficient. For the cubic benchmarks, we see no change at all (16 B and 32 B *gemm* case are missing) as given the kernel behavior, it generates a lot of swapping between C-SRAM, DRAM and NVM independent of vector size. Hence the global cost in both energy and timing in this case remains constant. The log plot (top row) clearly shows a saturation in gain for linear and quadratic kernels as described earlier. Only *gemm* does not exhibit this saturation which means that it could have better improvements using wider

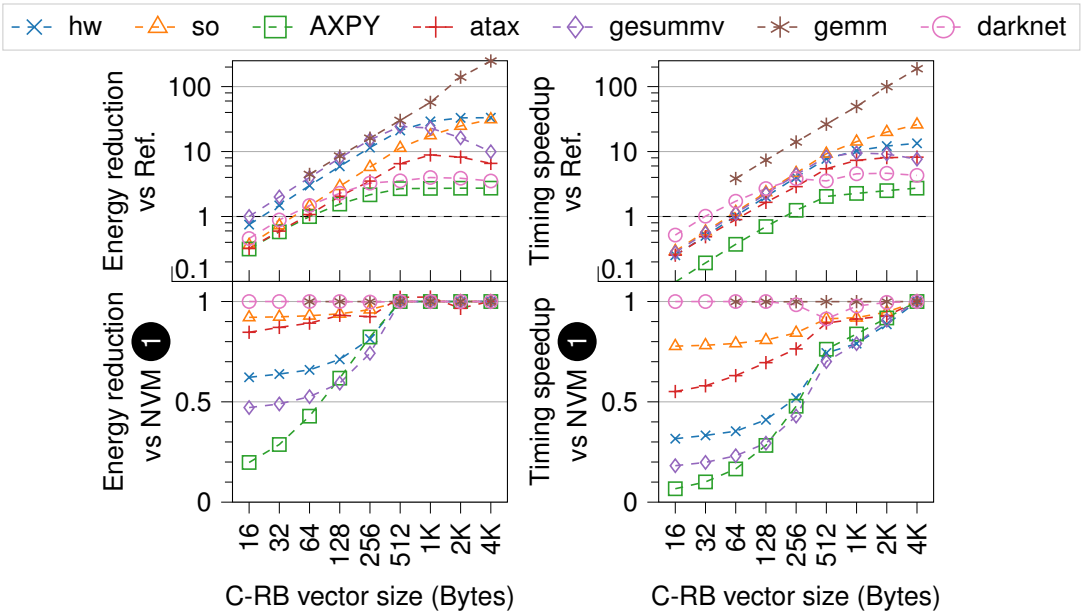


Figure 5.14.: Energy reduction and speedup of NVM row buffer scenario ❷ normalized against SIMD reference (top row), against independent C-SRAM (bottom row) and averaged for one vector size through all total sizes (higher is better)

vector. Nonetheless, our kernel must be representative and wider vector means larger matrix size which are more rare. For all benchmarks, energy and timing distribution is sensibly the same with an increase in the NVM energy and timing due to its smaller width in the general case. Similarly, the NVM accesses are constants compared to the first scenario.

Although there are some losses compared to the previous scenario, this one provides easier integration in a real system as it repurposes an already existing part, the row-buffer, into a computing row-buffer. In the best cases, there is no loss in energy and timing, and for a 256 B vector width, speedup can be reduced by half compared to an independent C-SRAM, but still improved by 5× compared to the SIMD reference. This scenario was published in DATE 2021 [212] and patented [213].

5.2.3. Scenario NVM ❶ with page transfer

In this scenario, the C-SRAM is independent from the DRAM and the SCM but all data transfer from NVM to the C-SRAM are of system page size, i.e. 4 kB, to ease coherency management. Once again, the results are very similar to the first scenario, so we normalize them against the latter. Second normalisation is through all vector sizes because vector width does not matter in the transfer type and only the total memory size has an influence. The results are shown in Figure 5.15.

As expected, page transfer only impacts small C-SRAMs size as big enough C-SRAMs

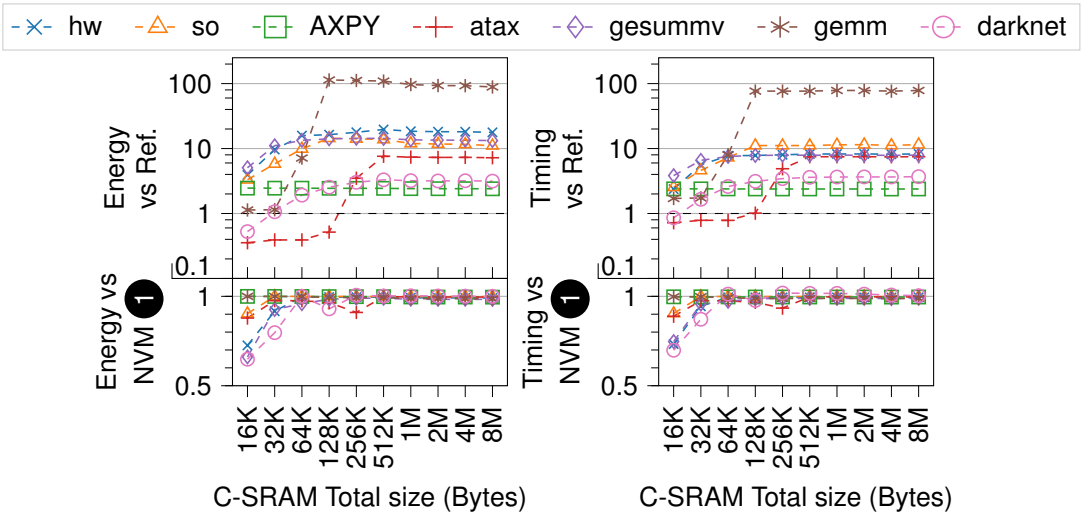


Figure 5.15.: Energy reduction and speedup of page transfer NVM scenario ❶ normalized against SIMD reference (top row), against independent C-SRAM (bottom row) and averaged for one total size through all vector sizes (higher is better)

have sufficient available memory space to manage those extra data. Moreover, most benchmarks have a good data locality which means that loading more data is equivalent to prefetching data that will be used in a few iterations without using any storage that is required by the C-SRAM for previous iterations. On the worst case for a 16 kB C-SRAM, we have a 27 % reduction compared to what we got from the scenario NVM ❶, in both timing and energy. For all total sizes above 512 kB, there is no impact on the improvements. For *AXPY*, there is no change observed as it always has sufficient space due to using only 3 C-SRAM lines.

5.2.4. Impact of the reduction loop

Several benchmarks (*atax*, *gesummv* and *gemm*) include a reduction loop or a memory broadcast that is performed by the CPU. The reduction loop transforms a vector into a scalar by the addition (add-reduction) and the memory broadcast simply set all elements of a vector to a given value from the memory. To analyze the impact of these two operations that force data to be sent back to the CPU, we add a couple of opcodes to our C-SRAM:

- `mbcast` for memory broadcast which allows a scalar data to be broadcast over an entire C-SRAM line using only its address;
- `scopy` for scalar copy which copies a scalar data into another scalar data without moving any of the two to the CPU.

We do not add a special add-reduction operator because this would severely impact silicon area, although not evaluated. Multiple add operators would need to be instan-

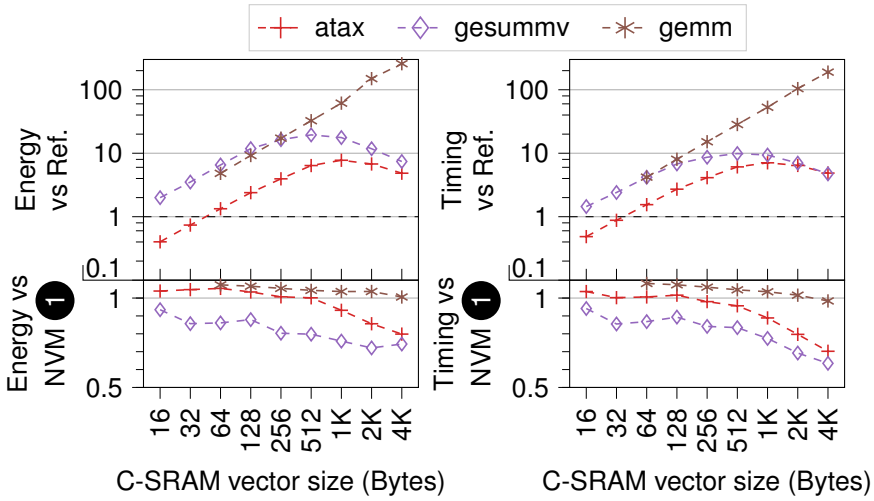


Figure 5.16.: Energy reduction and speedup when performing reduction loop and memory broadcast inside C-SRAM compared against SIMD reference (top row), against independent C-SRAM scenario NVM ❶ (bottom row) and averaged for one vector size through all total sizes (higher is better)

tiated instead of only a single already existing one. Moreover, it would need to be specifically designed for a given vector size instead of being generic.

First, we start by reducing all the vectors using `hswap` operators which allows us to go from 128 bits of data down to 8 bits. Then, we iterate over the reduced data using `scopy`, move them to a vector aligned address and add the results. This is used in *atax* and *gesummv*. *gemm* is written without a reduce loop, but can use `mbcast` to initialize a line to the scalar value of the matrix **A**. *atax* also uses `mbcast` right after the reduction. The results are normalized against the independent C-SRAM scenario NVM ❶ and presented in Figure 5.16. As one could expect, the bigger the vector size, the lesser the improvements as the reduction loop is longer and the C-SRAM is not very effective for this kind of operation. *gesummv* shows negative improvements for all the vector sizes whereas *atax* presents small gains, +5.4 % for energy and +3.6 % for speedup, for small vector sizes. After 256 B, *atax* also has negative improvements. Only *gemm* shows improvements for all the vector sizes, except 4 kB in timing, but the bigger vector size the better for this benchmark (Figure 5.7).

The introduction of new operators to perform the reduction loop with scalar operations demonstrates that it is of small interest because the C-SRAM still needs to activate full memory lines. It can however help for matrix kernels to further reduce data movement to the CPU.

5.3. Computing near DRAM

5.3.1. Scenario DRAM ❶: Independent C-SRAM between DRAM and L3 cache

We perform the same experiments by moving the C-SRAM between the DRAM and the caches. We expect to get way less improvements compared to previous solutions because the DRAM will be widely used. One main difference with the reference SIMD scenario is that data transfer from DRAM to C-SRAM is of vector size bytes while transfer from DRAM to L3 is always 64 B. As we use a closed row policy, this means that we do not need to open and close DRAM's rows as often, and thus can spare some time and energy. We perform the same analysis as in Section 5.2.1.

5.3.1.1. Linear benchmarks: *hamming weight*, *shift-or* & *AXPY*

For the linear benchmarks, we tend to the same observations as in scenario NVM ❶ but with way smaller improvements for both energy and timing. We reach 1.3× energy reduction and 1.8× speedup for *hamming weight* for the best case with a 512 kB 4 kB C-SRAM. For *shift-or*, we achieve a 2.8× energy reduction and up to 6.1× speedup for a 128 kB 4 kB C-SRAM. We can also observe the same line that defines the optimum ratio of memory lines versus memory size, especially for *shift-or*. Likewise, we observe a clear increase in energy reduction and speedup with a broader vector size, although a saturation is clearly visible for all 3 benchmarks and both metrics. Besides, we also note the same slight decrease of improvements when using greater memory sizes. A key difference is that the 64 B vector roughly corresponds to a unitary ratio with the SIMD 512-bits reference for both benchmarks. This is in line with the major part of the energy and timing spent in the DRAM that was short-circuited in previous scenarios. Hence, using the same vector width as the reference, we got equivalent performances results.

5.3.1.2. Quadratic kernels: *atax* & *gesummv*

Correspondingly for quadratic kernels, the same observations as before are still valid (Figure 5.18). We note for *atax* that a minimum working memory size is required to jump from negative improvements to small positive ones due to the complex behavior of this kernel. For *gesummv*, there is also a jump from 64 kB to 128 kB C-SRAM but otherwise, improvements are independent of the C-SRAM size. Increasing the vector width still has a negative impact on the improvements for both benchmarks. The best cases are a 1.3× better energy reduction and 2.6× speedup for *atax* with a 512 kB 1 kB C-SRAM while its capped at 1.2× better energy reduction and 1.6× speedup for *gesummv* using a 128 kB 1 kB C-SRAM.

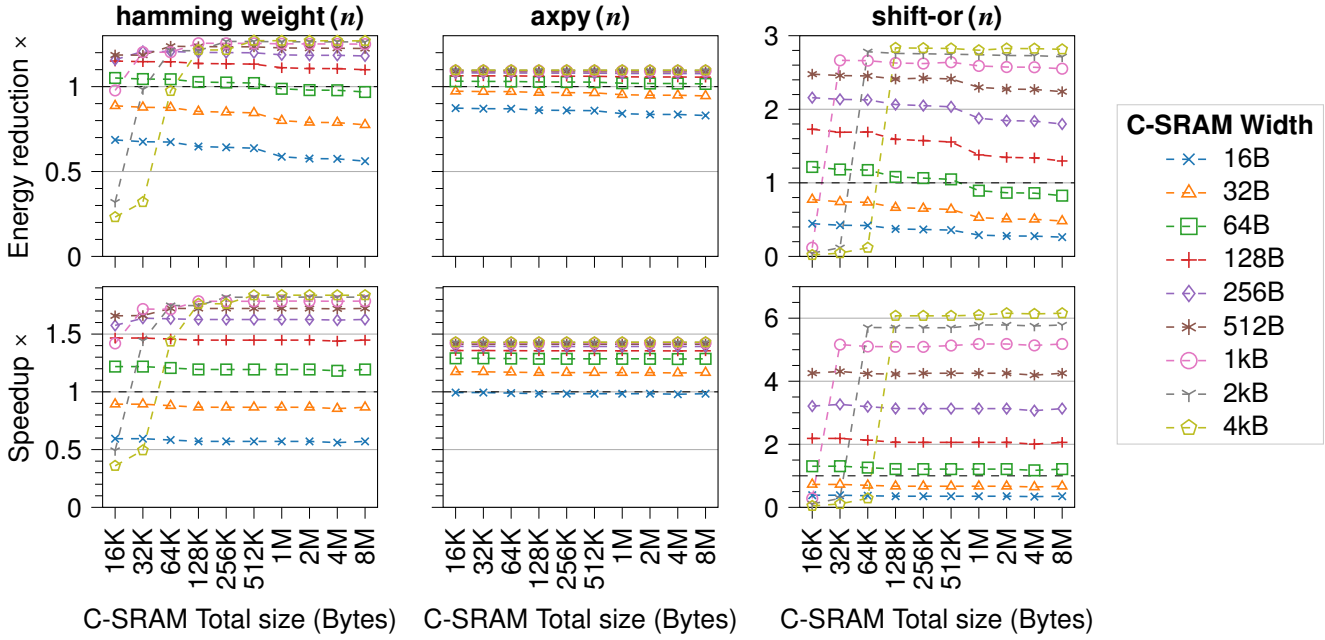


Figure 5.17.: Scenario DRAM ❶: Energy reduction and speedup for linear benchmarks normalized against SIMD 512-bit reference (higher is better)

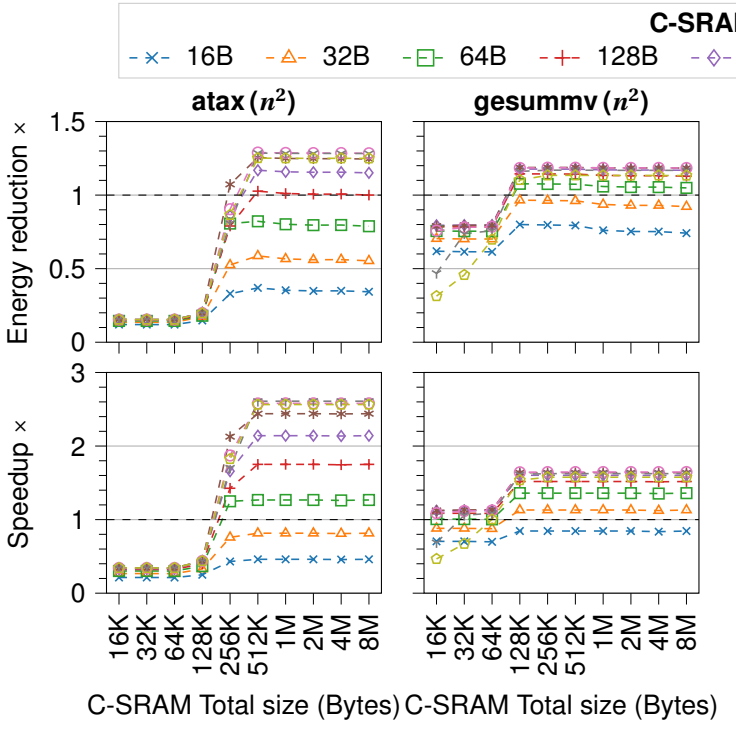


Figure 5.18.: Scenario DRAM ❶: Energy reduction and speedup for quadratic benchmarks normalized against SIMD 512-bit reference (higher is better)

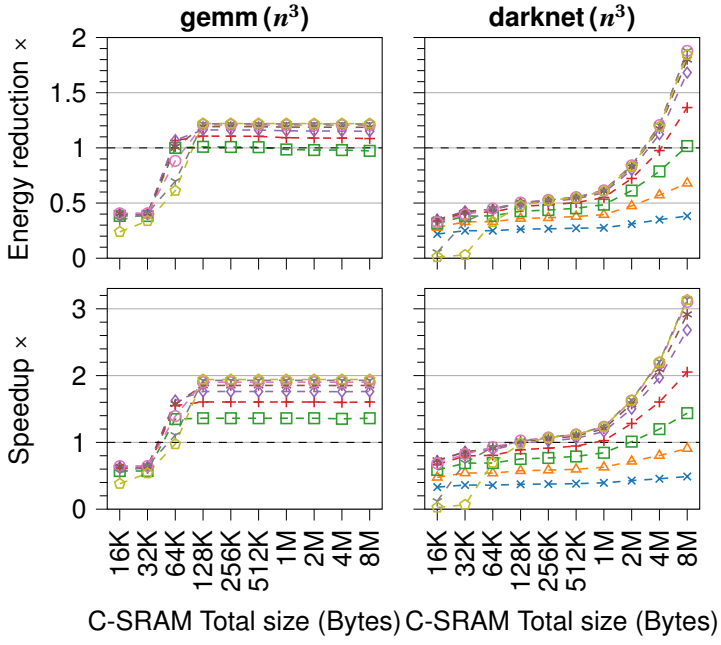


Figure 5.19.: Scenario DRAM ❶: Energy reduction and speedup for cubic benchmarks normalized against SIMD 512-bit reference (higher is better)

5.3.1.3. Cubic benchmarks: *gemm* & *darknet*

Moving on to the cubic kernels in Figure 5.19, the previous observations in Section 5.2.1 stand. There is still a thrashing behavior using small C-SRAMs with *gemm* as this does not depend on the C-SRAM location but only on memory size. Once again, we observe that the 64 B vector width shows no energy reduction compared to the reference, which is in line with the distribution shown in Figure 5.9a while timing is slightly reduced as DRAM's timing is about 50 % (Figure 5.9b). We achieve only a 1.3× energy reduction and a 1.9× speedup for a 128 kB 4 kB C-SRAM.

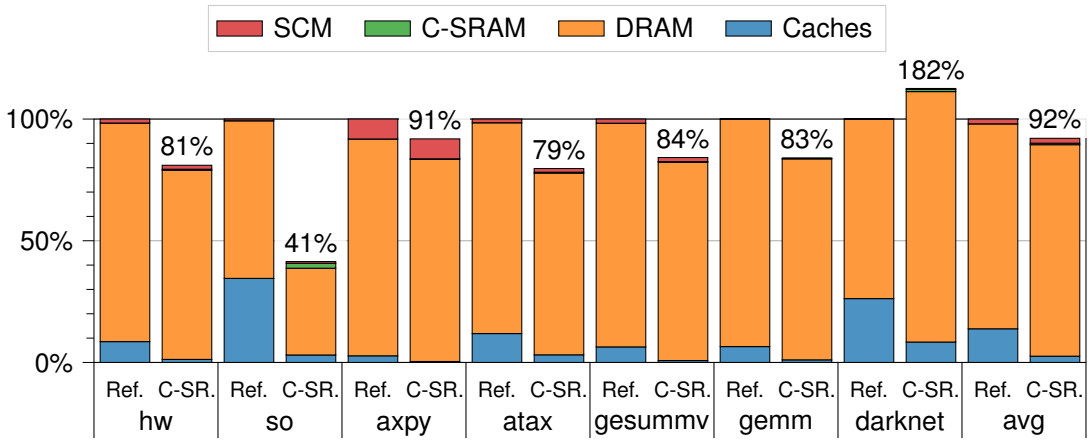
Looking at *darknet* benchmarks, which is our real case application, computing below DRAM memory level worsens both energy consumption and timing spent for most cases. We get positive improvements only with 4 MB C-SRAM and with vector width wider than 128 B for energy while timing gets faster from 128 kB C-SRAM and 256 B vector width. But at the best case using a 8 MB 4 kB C-SRAM, we still reach a 1.9× energy reduction and a 3.1× speedup compared to the SIMD reference.

5.3.1.4. Energy & timing distribution

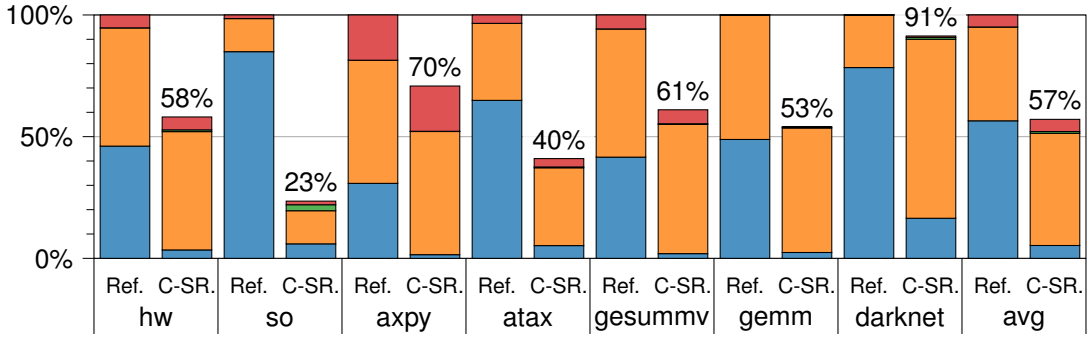
When looking at the energy distribution in Figure 5.20a, we see that C-SRAM computing at the DRAM level does reduce the energy consumption but by less than 10 % in average, while we had a 88 % reduction in our first proposed solution. As already showed in Figure 5.19, energy consumption is almost doubled for *darknet* due to thrashing. In all cases, caches energy is reduced and NVM energy remains constant similar to previous scenarios. For the timing distribution in Figure 5.20b, we have similar observations. Caches timings are drastically reduced by 92 % in average while DRAM usage remains constant, except for *darknet* where it is more than tripled. As stated in the start of this section, the main difference in this scenario versus the reference is that data transfer from DRAM to C-SRAM are based on C-SRAM vector width. To transfer n bytes from DRAM to L3, we would need to activate DRAM $\frac{n}{64\,\mathrm{B}}$ times where 64 B is the L3's block size. Each DRAM activation has a fixed cost (activation and precharge) before transfer of data can occur. With a larger data transfer, naturally less activation occurs hence the actual tiny improvement (<0.1 %) in timing. The caches energy and timing distribution are identical to those of Figure 5.10. NVM accesses are also unchanged compared to Figure 5.11.

5.3.1.5. Impact of the reduction loop

Contrary to the previous scenarios where the reduction loop and memory broadcast had more impact on both energy reduction and speedup, we see in Figure 5.21 that the results are more mitigated in this case. Indeed in the former scenarios, we had to move data through DRAM down to the CPU while now data just has to go through caches. Thus the deltas in both energy and timing we can spare is smaller, hence minor improvements when performing the reduction loop and memory broadcast in the C-SRAM. Similarly, as this delta is smaller, the losses that we observed for *gesummv* are also reduced in this case with the plots showing the same shapes. As we have



(a) Energy distribution



(b) Timing distribution

Figure 5.20.: Scenario DRAM 1: Energy and timing distribution for a C-SRAM of 512 kB and vector size of 512 B normalized to SIMD 512-bit reference architecture (lower is better)

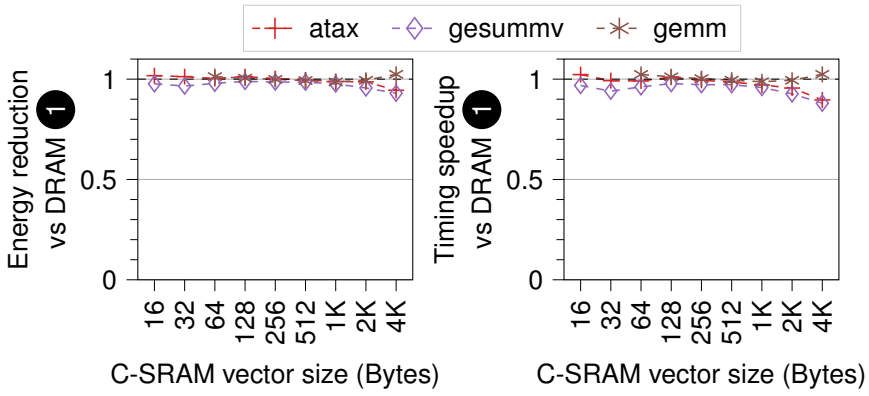


Figure 5.21.: Energy reduction and speedup when performing reduction loop and memory broadcast inside C-SRAM normalized against independent C-SRAM at DRAM level (scenario DRAM 1) and averaged for one total size through all vector sizes (higher is better)

positive improvements only for small vector sizes that are worse than the reference in this scenario, this solution is not to be used. Only *gemm* has a better outcome and only for the last case using a 4 kB wide vector getting a +2.4 % in both energy reduction and speedup while *atax* and *gesummv* have a −7 % worse energy reduction and are 11 % slower.

5.3.2. Scenario DRAM ❷: C-SRAM as Computing DRAM row buffer

The results of this scenario are really close to those of scenario DRAM ❶ (Section 5.3.1). The differences are inferior to 0.1 % for both timing and energy. Thus we do not plot the results as the figures are identical to Figure 5.17 for linear kernels, Figure 5.18 for quadratic kernels and Figure 5.19 for cubic benchmarks. The same analysis stands also for the energy and timing distribution plotted in Figure 5.20. The small differences come from the DRAM being activated less thanks to C-RB and allowing larger block of data to be transferred in a single row activation. However, this was also the case in the previous scenario as most of the data was transferred from DRAM to the C-SRAM. Now, only the data used by the CPU benefits from this small improvement.

Compared to the NVM scenarios, computing close to DRAM offers greatly reduced, but still positive, improvements for both energy reduction and speedup, up to 6.1× for linear kernels, 2.6× for quadratic kernels and 3.3× for *darknet*. Averaging all benchmarks, we get a −2 % in energy reduction but a +51 % speedup compared to SIMD reference (Figure 5.22). This considerable difference in energy reduction and speedup improvements is due to greatly reducing the caches usage which grants some speedup but very few energy reduction (Figure 5.10). Overall, this is mainly a result of having a shorter program execution time from the greater vectorisation.

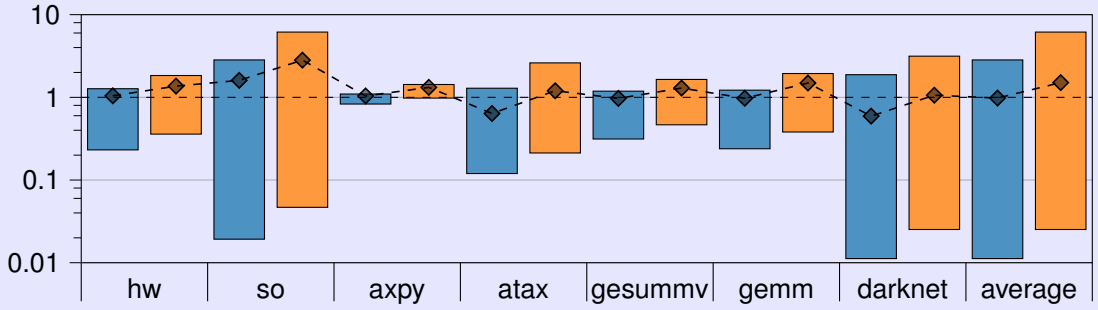


Figure 5.22.: Scenario DRAM ❶: Best, worst and average (◆) of *all* cases for both energy reduction ( ) and speedup ( ) normalized against SIMD reference (higher is better)

5.4. Conclusion

We have studied in the previous sections different plausible scenarios where the integration and data management of our proposed solution vary. Using several benchmarks representative of usual accesses patterns in data intensive applications, we show that computing close to the SCM is the most interesting solution concerning energy reduction and speedup, and that computing near the DRAM offers little to no improvements for these two metrics. The former yields an **energy reduction and a speedup, in average, of 17.4× and 12.9× respectively** while the latter concedes a 2 % increase in energy consumption but grants a 50 % speedup in average. Small adaptations such as using a page transfer mode instead of a fine grain transfer mode do not change the results when using a large enough C-SRAM. This demonstrates that **data management policy is not significant** and that **only the computing location in the memory hierarchy is relevant**. Performing the reduction loop inside the C-SRAM leads to lesser improvements as our solution was not designed for this kind of operation. Our solution also allows to **save the NVM's endurance** by not writing to it after each operation. We provide a full comparison of all the scenarios in Figure 5.23 as well as a best case selection in Table 5.2.

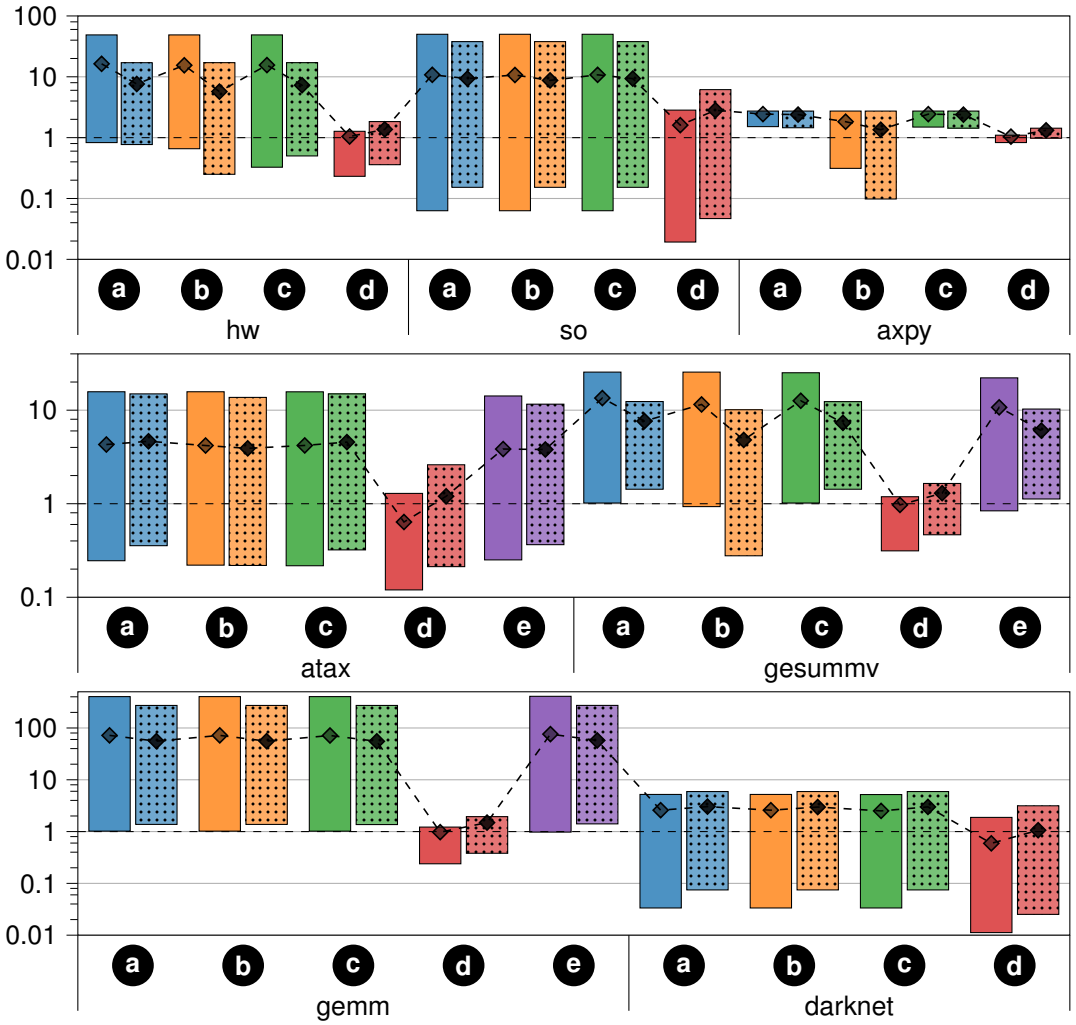


Figure 5.23.: Minimum, maximum and average (◆) for each benchmark and tested scenario. Left (plain color) is energy and right (lighter color and dots) is timing. **a** corresponds to the NVM's independent C-SRAM (NVM **1**); **b** is NVM C-RB (NVM **2**); **c** is NVM's independent C-SRAM with page transfer (NVM **1**); **d** is DRAM's independent C-SRAM (DRAM **1**); **e** is NVM's independent C-SRAM with reduction loop performed in the C-SRAM (NVM **1**). All graph higher is better

Table 5.2.: Best total and vector sizes for energy reduction (top), speedup (middle) and energy-delay product (bottom)

Benchmark	Max Gain	Size	Vector Size	Scenario
Hamming Weight	48.80	512.0K	4.0K	Independent C-SRAM (NVM ❶)
	17.07	1.0M	4.0K	Page Transfer (NVM ❶)
	830.23	512.0K	4.0K	Independent C-SRAM (NVM ❶)
shift-or	49.99	128.0K	4.0K	Page Transfer (NVM ❶)
	37.84	2.0M	4.0K	Row buffer (NVM ❷)
	1815.41	128.0K	4.0K	Page Transfer (NVM ❶)
AXPY	2.73	16.0K	4.0K	Page Transfer (NVM ❶)
	2.73	32.0K	4.0K	Row buffer (NVM ❷)
	7.48	16.0K	4.0K	Page Transfer (NVM ❶)
atax	15.74	512.0K	1.0K	Row buffer (NVM ❶)
	14.94	1.0M	2.0K	Independent C-SRAM (NVM ❶)
	226.77	512.0K	2.0K	Independent C-SRAM (NVM ❶)
gesummv	25.53	64.0K	512B	Row buffer (NVM ❷)
	12.37	1.0M	1.0K	Independent C-SRAM (NVM ❶)
	303.53	64.0K	512B	Independent C-SRAM (NVM ❶)
gemm	408.79	128.0K	4.0K	reduce Independent C-SRAM (NVM ❶)
	273.00	8.0M	4.0K	reduce Row buffer (NVM ❷)
	109457.78	128.0K	4.0K	reduce Independent C-SRAM (NVM ❶)
darknet	5.21	512.0K	2.0K	Row buffer (NVM ❷)
	5.92	8.0M	4.0K	Page Transfer (NVM ❶)
	29.75	512.0K	2.0K	Independent C-SRAM (NVM ❶)

Conclusion

Ah ça y est. Je viens de comprendre à quoi ça sert la canne. En fait ça sert à rien [...] Du coup, ça nous renvoie à notre propre utilité : l'Homme, face à l'absurde.

— Perceval IN *Kaamelott* BOOK IV, EPISODE 95, « *L'Inspiration* »

We are all apprentices in a craft where no one ever becomes a master.

— Ernest Hemingway

We have shown that current technology trends are facing a soon to come dead end with the end of technology scaling for conventional CMOS. Several other problems arise. First, power density keeps increasing since the end of Dennard's law which cause issues with heat dissipation. It also gave birth to dark silicon phenomenon where a circuit cannot be fully powered permanently. One of the most visible consequence is the halt of clock increase in 2005 as circuits could not be cooled down enough. Secondly, the increasing demand for more computing power has led to several hardware innovations. When clock could not be further accelerated, multicore Central Processing Unit (CPU) architecture were introduced and today's chip can reach tens of cores in a single CPU. Single Instruction Multiple Data (SIMD) have followed to answer the rising need for processing performance with up to 512 bits architecture for faster vector processing. Branch predictors, memory prefetching and speculative execution are all hardware improvements to fasten processing speed. However, this perpetual race for performances lead to a diminution of hardware security with several hardware vulnerability such as Spectre and Meltdown. Finally, memory performances, mainly Dynamic Random Access Memory (DRAM), disks and Flash, did not scale proportionately. While CPU performances were multiplied by 1000, DRAM speed was only increased by a factor 10. This stretch of performance is called the memory wall as it is a performance wall that no CPU can go beyond, regardless of the number of cores, prefetchers or other methods to speed up processing. A linked issue is the von Neumann bottleneck as all the data needs to go from a wide row in memory into scalar registers in CPU. To reduce the performance gap between memory and processors, caches were extended and deepened, going from a single cache to 3 levels of cache memory in modern CPUs. They are based on spatial and temporal locality to *cache* neighbouring data that is likely to be used or reused soon. But the downside is that data now needs to traverse several cache levels before arriving to the processing unit and likewise for being written back into memory. This is worsened by coarse data transfer in caches, typically 64 bytes are transferred, even if only a single byte is used. As a consequence, data access is the leading cost in CPUs, being hundreds times more expensive than computation itself. Similarly, time needed for these data movement are rising up. It should be noted that, on average, memory represents between 50 % and 80 % of a system's overall energy budget. Although memory wall has

existed since 40 years, new memory technologies have been forecast to reduce this gap. Hybrid Memory Cube (HMC) and High Bandwidth Memory (HBM) are recent DRAM technologies that manifolded bandwidth by using larger bus width and exploiting 3D stacking. Despite largely improving bandwidth, von Neumann bottleneck still remains as the architecture stayed identical and being compute centric rather than data centric. Furthermore, the dynamic nature of DRAM is not eliminated and is one of the major energy consumer. New emerging Non Volatile Memory (NVM) technologies can partially solve this problem but a global solution demands a paradigm shift to data centric architecture. To this end, In-Memory Computing (IMC) is a promising solution as it can, if not completely, greatly reduce data movement in the architecture, and if used with NVMs, it will also cut down energy consumption. Moreover, NVMs have promising performances in both energy and timing access costs, intermediate to Static Random Access Memory (SRAM) and DRAM which may enable us to remove some memories from the memory hierarchy, including DRAM and perhaps level 3 cache at least.

State of the art study reveal plethora of methods and technics to implement IMC based solution in all memory technologies and at every level of the memory hierarchy. We can classify them in different categories depending on where the computing takes place, whether inside the bitcell array at the analog level, or after the Sense Amplifiers (SAs) in the digital domain or a mix of both. SRAM based solutions rely on both type of memory computing: IMC and Near-Memory Computing (NMC). We make the distinction between standard 6T bitcell that is widely used in the industry and modified bitcells (8T, 9T, 10T, etc.) that are specifically designed for IMC. Proposed solutions are mostly targeted at specific applications, especially Artificial Intelligence (AI) based applications (deep learning), with only one general purpose solution. Yet by using SRAM memories, these solutions places themselves at the bottom of the memory hierarchy, i.e. close to the CPU, or target edge devices with flat memory hierarchy. As such, they do not address the data movements through the memory hierarchy and also expose low improvements against their baseline. Moving on to DRAM memory, we present multiple solutions ranging from *true* IMC to Processing In Memory (PIM) where computing is completely outside the memory circuit. Nonetheless, most papers are simulation only and as far as I know, there are only 3 demonstrated solutions including an already commercialized one. As DRAM uses a different foundry process than conventional CMOS, it is harder to integrate potential solutions. Moreover, DRAM read is destructive which incurs additional data copy before performing operation. Yet, DRAM is one of the biggest energy consumer in data centers and consumer devices. Computing inside this technology only solve partially the von Neumann bottleneck, as the data is often not originating from this memory.

Getting to NAND Flash which is the most common purely electrical NVM, i.e. with no moving part contrary to Hard Disk Drives (HDDs) and tapes. Most designs focus on Matrix Vector Multiplication (MVM) using current summation as analog mean of computing which is too restrictive. Moreover, the limited NAND Flash endurance is not regarded as a problem. However, its high density can make NAND Flash memory

computing an interesting solution for edge devices using large neural networks. Resistive Random Access Memory (RRAM) offers the most promising integration along with IMC as bitcells are disposed appropriately to create logic functions. Crossbar array structure also is the most dense memory implementation (excluding 3D technologies). As all resistive memory technology, numerous solutions use MVM current summation implementation. Despite having some of the most impressive metrics such as energy efficiency up to 700 TOPS/W, it is limited to low precision, typically binary and up to 8 bits, and suffers from analog computing with intrinsic device variability. The same observation applies for Phase Change Memory (PCM) and Magnetic Random Access Memory (MRAM) except that IMC integration is complicated by incompatible foundry process to make these memories. PCM has the highest on-off ratio of NVMs and easily supports multilevel bitcells while MRAM, due to its binary orientation nature, is limited to single level cell.

Finally, we studied some generic works that apply to all kinds of memory using only a common property, namely resistive memories. Other works tried to combine multiple memories to get the best of both worlds. We also made a small overview of programming models, Instruction Set Architecture (ISA) and listed the limitations and constraints still faced by IMC: complex hardware implementation, analog computing with low precision, limited endurance of emerging NVMs, application specific (mostly deep learning) solution lacking generality and few architectural studies. That is why we believe a digital wrapper around any NVM will achieve the best performances and save endurance by using small SRAM buffers to perform computation and serve as a write buffer. Digital wrapper grants more flexibility and uses conventional CMOS operators which are less limited in terms of precision. It also offers the generality required for computing systems and is not subject to analog noise.

As presented in Chapter 2, analog computing is not general purpose and is subject to noise from device variations and thermal effects. The use of modified bitcell in SRAM to provide isolation between bitlines and bitcells when computing is counterproductive. First, the *precomputed* operations can easily be done with a couple of transistors in the digital domain, and the cost of multiple wordlines activation requires row decoder modification whose area cost exceeds the few spared gates. Second, non standard bitcells are designed with logic rules, i.e. with less strict design rules leading to less dense memory. Design validation costs must also be taken into account to make sure the designed bitcell works properly in a wide range of situation (low/high temperature, low/high voltage, etc.). Computing in the NVM would wear it down in a few days due to its limited endurance. Moreover, it may have *fast* memory access, but its energy access cost still surpasses SRAM's one. Thus, we decide to design a SRAM based digital wrapper to be placed around NVM to kill two birds with one stone. On one hand, we get the benefits of high density NVM and non volatility. On the other hand, we have unlimited SRAM endurance and fast memory access for enhanced computing performances.

Our wrapper is designed as a vector computation unit with its own pipeline and decode unit. It receives instruction from the CPU and executes them on the fly. To

reduce hardware complexity, we do not handle pipeline hazards such as Read After Write (RAW) or Write After Write (WAW) and leave it to the compiler or the developer. The Arithmetic & Logical Unit (ALU) supports bitwise logic operations, shifts and arithmetic operations: addition, subtraction, multiplication and comparison. The shifts, addition and subtraction can be of any 8, 16 or 32 bits while the multiplication is only 8 bits in our proof of concept. Only integer arithmetic is supported, but fixed point is easily implemented using shift operations. In order to minimize silicon area, operators are muxed so that only a 64-bit adder is instantiated and bit tricks are used to mimic narrower width. An ISA is designed inspired by RISC-V and bit placements are chosen to minimize muxes in decode stage. The wrapper receives instruction on both address and data buses and the address MSB determines if it is a normal memory operation or an IMC operation (memory mapped peripheral).

I carry an exploration work using standard design and simulation tools on multiple SRAM types with different number of ports: 1RW, 1R1W, 1R1RW and 2RW. We use a 22 nm node from GlobalFoundries. I show that the area overhead of our solution is limited to 5 % for 256 kB memory which is acceptable while the power overhead is about 20 % for the same memory; it is more expensive but adding functionality, in this case computing, always comes at a cost. I demonstrate that 1R1W memory types have the best EDP, being at least twice better than 2RW and 4× better than 1RW memories. In terms of energy efficiency, our design achieves 2 TOPS/W with a 2 kB 1R1W memory which is lower than state of the art solutions. However, this result must be tempered as state of the art is often measured on MAC operation, where the multiplication is single bit while ours is 8-bit. Using logic decomposition of 8-bit multiplication into 1-bit operation, we obtain a factor of 400 which puts us at the top. Operation density is around 100 GOPS/mm^2 which is the average in the state of the art, and once again set our design above most state of the art solutions with the correction factor. This proves that our proposed solution offers diversity in terms of memory types and sizes, while providing efficient and general purpose designs. Our digital wrapper, that we name C-SRAM, can be furthermore customized by easily removing or adding operators as our RTL workflow provides fast prototyping.

In Chapter 4, I present the benchmarks we use and the platform I develop to explore various architectures with our digital wrapper. We target *big data* applications as those are the one pressurising the memory systems and demanding the most performances out of computers. We consider several linear benchmarks, namely *hamming weight* used in information theory, *shift-or* used in bioengineering for protein pattern matching and *AXPY*, a classic Basic Linear Algebra Subprograms (BLAS) kernel. The first two are compute bound, i.e. they are limited by the computing speed of the CPU while the last one is memory bound, i.e. limited by the memory bandwidth. Next, we have two quadratic benchmarks which are, *atax* used in linear solvers and *gesummv* a general case of MVM used in image processing and deep learning. Finally, we pick a cubic benchmark, *gemm*, a matrix multiplication kernel that is widely spread in numerous applications: AI with deep learning and neural networks, image processing, scientific simulation, etc. We also select a real case application, *darknet* a neural

network implementation using 3 of the previous kernels: *gemm*, *gesummv* and *AXPY*.

Before exploring the integration of our digital wrapper in a complete architecture, I develop a simulation platform suited to our needs. First, I explain why I do not use state of the art proposed simulators such as gem5. Indeed, it is one of the most used system simulator in system architecture research yet suffers from numerous flaws. We are especially interested in the memory subsystem but it is known to be inaccurate. As we are prototyping new architectures, we cannot use performance counters available in most CPUs as they cannot count not yet existing events and are inaccurate for memory events beyond L2. Then, I present hardware model tools that I use to simulate the different stages of the memory hierarchy and retrieve accurate energy and timing measurements. I choose to use NVSim to model the NVM but also the cache hierarchy as it is based on Cacti which is the reference on that matter. It also gives less absurd results than the latter. We decide to use PCM as our Storage Class Memory (SCM) since it is the most mature technology in that matter with already existing commercial devices. Technology parameters for the PCM are extracted from state of the art papers to be as realistic as possible. The DRAM is modelled using VAMPIRE, a tool that is calibrated on real measurements of commercial devices. I also considered other tools but picked up this one as it is the most recent of all and its methodology seemed the most robust. Finally, C-SRAM parameters are taken from my previous study in Chapter 3 and are extended for bigger and wider memories using tiling pattern calibrated from a team previous work.

To develop our exploration platform, I use Pin, a tool that allows to instrument any CPU instruction and in our case, every memory access. This includes all instructions as each instruction is loaded from memory. As an instrumentation tool, it has a low overhead penalty, i.e. around 100× compared to full simulators or emulators which can be a million times slower than real time execution. Since we target big data applications with huge datasets, simulation speed is an important metric in our choice. A software interface is designed to be compatible with our presented ISA in Chapter 3. This ISA is however extended to support 64 bits address mode instead of 32 bits. Then, I model a 3 levels cache hierarchy, a DRAM, our C-SRAM behavior and a top level SCM. The first version of the platform models an enforced coherency, where computing in the C-SRAM updates the value in the caches and reciprocally, but that cost is not accounted for in the memory statistics. Moreover, all SCM accesses are not counted as swapping is not tracked. Memory initialization is always coming from the SCM which is not the case with growing stack or heap. This first version led to a publication in DATE21 [212]. A second version refines these aspects to be more accurate. It handles system call that accesses file system to monitor those and count them in the global energy and timing bill. Memory allocation is correctly traced so that useless data copy from SCM to DRAM is removed. Pages are now tracked to account for swapping when DRAM is full, saved into the SCM and loaded back when data is needed. Finally, I switch from an *hardware* enforced coherency to a software coherency approach and adds adequate memory management functions to our ISA.

I then perform a validation step to ensure that our model accurately represents the reality. I compare the number of cache accesses of our platforms with performance

counter for caches hierarchy on a real architecture with the same parameters. Our modelling is accurate and the bigger the dataset, the more accurate we are. Some events seem as incorrectly modelled but it is because of their scarcity so that a small variation leads to huge ratio. Nonetheless, these rare events are hidden by the billions of other correctly counted events. I confront different DRAM simulators' results and show that the power and timings each span over 5 orders of magnitude. This proves how hard it is to accurately model DRAM. Using a *fixed* access cost, we got the most accurate timing compared to other simulators using real time execution as reference. Nevertheless, energy estimation shows more variations and is not referenced with a real measurement as it requires complex set up. To better assess, I plot DRAM power computed as energy divided by time and show that some simulators give a power superior to 1 kW which is ridiculous. Our platform yields power in the range 1–10 W but do have some outliers with an estimated power of 50 W.

Now that we have designed our digital wrapper and developed a simulation platform, we can put them both together to start exploring new computing architectures. As we have shown in the state of the art, computing close to the CPU yields low ameliorations while computing higher in the memory hierarchy, like DRAM or NVM has much more room for improvements. Using the listed benchmarks in Chapter 4 and the defined methodology, we explore four different architectures where computing takes place either in the repurposed row-buffer of the memory or our C-SRAM is placed in between the lower rank memory (DRAM or L3 cache) and the targeted memory (respectively SCM or DRAM) as an independent circuit. The reference baseline is a 512-bit SIMD CPU with a 3 levels cache hierarchy which is the de facto standard in High Performance Computing (HPC) nowadays. Follow a small DRAM that is voluntarily of reduced size so that datasets do not hold all together in it, similar to modern big data applications. At the top is a SCM of type PCM that contains all the data required for the application to run correctly. The basic idea of computing close to the SCM is that this is where data is, whether the closest or originates from or both. Indeed, massive datasets such as neural networks weights and images databases are saved on permanent storage, so the cost of loading these data from SCM must be taken into account.

In the first two architectures where the C-SRAM is either a C-RB in the SCM or between the DRAM and the SCM, we severely reduce both energy consumption and execution time, up to in average 17.4× and 12.9× respectively. First, we note that all benchmarks have a significant energy reduction and speedup in our exploration space, and that a C-SRAM between 128 kB and 512 kB is the optimal size. Secondly, we distinguish different improvement patterns based on the benchmarks behaviors. We make the following observations:

- Linear benchmarks benefits from wider computing vectors as there are no dependencies between loop iterations. We achieve a 50× energy reduction and speedup up to 38× for *shift-or* and 17× for *hamming weight*. For *AXPY*, we remark that the improvements are much lower, between 2× and 3× for both energy reduction and speedup. This is due to write intensive SCM accesses which limit the maximum improvements.

- Quadratic benchmarks show less impressive improvements but still interesting with energy reduction and speedup around 15× for *atax*, and 25× and 12× respectively for *gesummv*. More complex access behaviors and data reuse explain the differences with linear benchmarks. Moreover, bigger vector does not enhance improvements due to data reduction operations that cannot be performed by the C-SRAM. A variation where this reduction is done in place shows no significant improvements (<1 %).

- Cubic benchmarks such as *gemm* have the best improvements of all, with energy reduction toping up at 403× and speedup reaching 272×. However, this is just a kernel and a real life application such as *darknet* shows way less enhancement, around 5× energy reduction and 6× speedup due to Amdahl's law.

Finally, we show that SCM accesses remains constants for both reads and writes, compared to the reference so that our solution does not reduce SCM lifetime.

In the second cases where computing takes place near DRAM, the gains are much more limited or even negative as DRAM accesses are not reduced at all and still represents up to 80 % of overall energy consumption, yielding in consequence very little energy reduction and speedup.

Perspectives and future works

I have shown that computing at the top of the memory hierarchy is the best place to integrate memory computing, whatever memory technology is used for SCM. IMC solutions family can yield large improvements in a wide range of applications including linear algebra, deep learning, databases, etc. Nonetheless, some works still need to be carried on. First of all, a real life demonstration would confirm our results but that would require huge work in both hardware and software level. Our designed ISA would need hardware implementation in a RISC-V CPU, but the toughest part is probably the bus integration above DRAM to communicate with the C-SRAM. Although I have demonstrated reduced energy consumption, I have not considered instant power which is probably spiking when performing operation on large vectors. This in turns may pose problems for the underlying memory if our C-SRAM heats too much. At the software level, programming model still needs to change paradigm from compute centric to data centric. Special instructions for finer data movement in a distributed computing architecture can also be explored. Enhanced security claims are still to be verified as well.

On a final note, I would like to warn about the undesired effects of our proposed C-SRAM solution. Reducing energy consumption is of course a noble cause. But it will probably not reduce the energy consumption of data centers or devices as long as the energy budget remains stable. As such, all the saved energy will be spent in hundreds or thousands of more computing devices to keep the growth of computing performances. This is the rebound effect.

Bibliography

[1] Gordon E Moore. “Cramming more components onto integrated circuits”. In: *Electronics* 38.8 (1965), p. 4 (cit. on p. 3).

[2] Jan Rabaey. *Low Power Design Essentials*. Integrated Circuits and Systems. Boston, MA: Springer US, 2009. ISBN: 978-0-387-71712-8. DOI: 10.1007/978-0-387-71713-5 (cit. on p. 4).

[3] Nam Sung Kim et al. “Leakage current: Moore’s law meets static power”. In: *Computer* 36.12 (2003-12), pp. 68–75. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1250885 (cit. on p. 4).

[4] Stephen W. Keckler et al. “GPUs and the Future of Parallel Computing”. In: *IEEE Micro* 31.5 (2011-09), pp. 7–17. ISSN: 0272-1732. DOI: 10.1109/MM.2011.89 (cit. on p. 4).

[5] R.H. Dennard et al. “Design of ion-implanted MOSFET’s with very small physical dimensions”. In: *IEEE Journal of Solid-State Circuits* 9.5 (1974-10), pp. 256–268. ISSN: 1558-173X. DOI: 10.1109/JSSC.1974.1050511 (cit. on p. 4).

[6] Anja Bog. “The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software”. en. In: *Dr. Dobb’s Journal* (2005-03), p. 8 (cit. on p. 5).

[7] Andrew Danowitz et al. “CPU DB: recording microprocessor history”. In: *Communications of the ACM* 55.4 (2012-04-01), pp. 55–63. ISSN: 0001-0782. DOI: 10.1145/2133806.2133822. URL: http://cpudb.stanford.edu (cit. on p. 5).

[8] *AMD Ryzen™ Threadripper™ 3990X Processor*. URL: https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-3990x (visited on 2022-07-13) (cit. on p. 5).

[9] H. Esmaeilzadeh et al. “Dark silicon and the end of multicore scaling”. In: *2011 38th Annual International Symposium on Computer Architecture (ISCA)*. 2011-06, pp. 365–376 (cit. on p. 5).

[10] *Xeon Silver 4116 - Intel - WikiChip*. URL: https://en.wikichip.org/wiki/intel/xeon_silver/4116 (visited on 2022-07-05) (cit. on p. 6).

[11] Chris Gregg and Kim Hazelwood. “Where is the data? Why you cannot debate CPU vs. GPU performance without the answer”. In: *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*. 2011-04, pp. 134–144. DOI: 10.1109/ISPASS.2011.5762730 (cit. on p. 6).

[12] *June 2022 | TOP500*. URL: https://www.top500.org/lists/top500/2022/06/ (visited on 2022-07-13) (cit. on pp. 6, 8).

[13] Bill Dally. “To exascale and Beyond”. In: Supercomputing. 2010. URL: https://www.nvidia.com/content/PDF/sc_2010/theater/Dally_SC10.pdf (visited on 2021-11-11) (cit. on p. 6).

[14] John L Hennessy. *Computer Architecture: A Quantitative Approach* (cit. on p. 7).

[15] Paul Kocher et al. "Spectre attacks: exploiting speculative execution". In: *Communications of the ACM* 63.7 (2020-06), pp. 93–101. ISSN: 0001-0782. DOI: 10.1145/3399742 (cit. on p. 7).

[16] Hadi Esmaeilzadeh et al. "Looking back on the language and hardware revolutions: measured power, performance, and scaling". In: *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*. ASPLOS XVI. New York, NY, USA: Association for Computing Machinery, 2011-03-05, pp. 319–332. ISBN: 978-1-4503-0266-1. DOI: 10.1145/1950365.1950402 (cit. on p. 7).

[17] M. Horowitz. "1.1 Computing's energy problem (and what we can do about it)". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 2014-02, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323 (cit. on p. 8).

[18] Aurélie Villard, Alan Lelah, and Daniel Brissaud. "Drawing a chip environmental profile: environmental indicators for the semiconductor industry". en. In: *Journal of Cleaner Production* 86 (2015-01), pp. 98–109. ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2014.08.061. (Visited on 2022-10-07) (cit. on p. 8).

[19] Valeria Bertacco. "Re-Imagining Scalable System Design". In: *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC). 2018-10, pp. ix–xiii. DOI: 10.1109/VLSI-SoC.2018.8644750 (cit. on p. 9).

[20] Hoan Nguyen et al. "A 7NM Double-Pumped 6R6W Register File for Machine Learning Memory". In: *2018 IEEE Symposium on VLSI Circuits*. 2018 IEEE Symposium on VLSI Circuits. 2018-06, pp. 1–2. DOI: 10.1109/VLSIC.2018.8502393 (cit. on pp. 10, 49).

[21] Yoongu Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. ISSN: 1063-6897. 2014-06, pp. 361–372. DOI: 10.1109/ISCA.2014.6853210 (cit. on p. 12).

[22] Micron. *176-Layer NAND*. en. 2022. URL: https://www.micron.com/products/nand-flash/176-layer-nand (visited on 2022-07-13) (cit. on pp. 13, 35).

[23] *Lecture 21: Storage*. URL: https://www.cs.utexas.edu/users/mckinley/352/lectures/21.pdf (visited on 2022-07-13) (cit. on p. 15).

[24] *IBM Makes Tape Storage Better Than Ever*. IEEE Spectrum. 2020-12-17 (cit. on p. 15).

[25] M. M. Sabry Aly et al. "Energy-Efficient Abundant-Data Computing: The N3XT 1,000x". In: *Computer* 48.12 (2015-12), pp. 24–33. ISSN: 0018-9162. DOI: 10.1109/MC.2015.376 (cit. on pp. 17, 27).

[26] SHARP KABUSHIKI KAISHA. *EUIPO - eSearch*. 2003. URL: https://euipo.europa.eu/eSearch/#details/trademarks/003062791 (visited on 2022-09-06) (cit. on p. 18).

[27] Alexander Hankin et al. "Evaluation of Non-Volatile Memory Based Last Level Cache Given Modern Use Case Behavior". In: *2019 IEEE International Symposium on Workload Characterization (IISWC)*. 2019-11, pp. 143–154. DOI: 10.1109/IISWC47752.2019.9042051 (cit. on p. 18).

[28] H.-S. Philip Wong et al. "Metal–Oxide RRAM". In: *Proceedings of the IEEE* 100.6 (2012-06), pp. 1951–1970. ISSN: 1558-2256. DOI: 10.1109/JPROC.2012.2190369 (cit. on pp. 19, 22).

[29] Yangyin Chen. "ReRAM: History, Status, and Future". In: *IEEE Transactions on Electron Devices* 67.4 (2020-04), pp. 1420–1433. ISSN: 0018-9383, 1557-9646. DOI: 10.1109/TED.2019.2961505 (cit. on p. 19).

[30] Shimeng Yu and Pai-Yu Chen. "Emerging Memory Technologies: Recent Trends and Prospects". In: *IEEE Solid-State Circuits Magazine* 8.2 (2016), pp. 43–56. ISSN: 1943-0590. DOI: 10.1109/MSSC.2016.2546199 (cit. on pp. 19–22).

[31] M. Ezzadeen et al. "Low-Overhead Implementation of Binarized Neural Networks Employing Robust 2T2R Resistive RAM Bridges". In: *ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC)*. 2021-09, pp. 83–86. DOI: 10.1109/ESSCIRC53450.2021.9567742 (cit. on p. 19).

[32] Young-Bae Kim et al. "Bi-layered RRAM with unlimited endurance and extremely uniform switching". In: *2011 Symposium on VLSI Technology - Digest of Technical Papers*. 2011-06, pp. 52–53 (cit. on p. 19).

[33] Y. S. Chen et al. "Highly scalable hafnium oxide memory with improvements of resistive distribution and read disturb immunity". In: *2009 IEEE International Electron Devices Meeting (IEDM)*. 2009-12, pp. 1–4. DOI: 10.1109/IEDM.2009.5424411 (cit. on p. 19).

[34] H.-S. Philip Wong et al. "Phase Change Memory". In: *Proceedings of the IEEE* 98.12 (2010-12), pp. 2201–2227. ISSN: 1558-2256. DOI: 10.1109/JPROC.2010.2070050 (cit. on pp. 20, 22).

[35] Moinuddin K. Qureshi et al. "Enhancing lifetime and security of PCM-based Main Memory with Start-Gap Wear Leveling". In: *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2009-12, pp. 14–23. DOI: 10.1145/1669112.1669117 (cit. on pp. 20, 21).

[36] Andre Seznec. "A Phase Change Memory as a Secure Main Memory". In: *IEEE Computer Architecture Letters* 9.1 (2010-01), pp. 5–8. ISSN: 2473-2575. DOI: 10.1109/L-CA.2010.2 (cit. on pp. 20, 21).

[37] T. Nirschl et al. "Write Strategies for 2 and 4-bit Multi-Level Phase-Change Memory". In: *2007 IEEE International Electron Devices Meeting*. 2007-12, pp. 461–464. DOI: 10.1109/IEDM.2007.4418973 (cit. on p. 20).

[38] Benjamin C. Lee et al. “Phase-Change Technology and the Future of Main Memory”. In: *IEEE Micro* 30.1 (2010-01), pp. 143–143. ISSN: 1937-4143. DOI: 10.1109/MM.2010.24 (cit. on p. 21).

[39] Jaehyun Park, Donghwa Shin, and Hyung Gyu Lee. “Design space exploration of row buffer architecture for phase change memory with LPDDR2-NVM interface”. In: *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. 2015-10, pp. 104–109. DOI: 10.1109/VLSI-SoC.2015.7314400 (cit. on p. 21).

[40] Benjamin C. Lee et al. “Architecting phase change memory as a scalable dram alternative”. In: *Proceedings of the 36th annual international symposium on Computer architecture*. ISCA ’09. New York, NY, USA: Association for Computing Machinery, 2009-06, pp. 2–13. ISBN: 978-1-60558-526-0. DOI: 10.1145/1555754.1555758 (cit. on p. 21).

[41] Yiming Huai et al. “High Density 3D Cross-Point STT-MRAM”. In: *2018 IEEE International Memory Workshop (IMW)*. 2018-05, pp. 1–4. DOI: 10.1109/IMW.2018.8388833 (cit. on pp. 21, 22).

[42] An Chen. “A review of emerging non-volatile memory (NVM) technologies and applications”. en. In: *Solid-State Electronics*. Extended papers selected from ESSDERC 2015 125 (2016-11), pp. 25–38. ISSN: 0038-1101. DOI: 10.1016/j.sse.2016.07.006 (cit. on pp. 21, 22).

[43] Sanjay Prajapati and Brajesh Kumar Kaushik. “Area and Energy Efficient Series Multilevel Cell STT-MRAMs for Optimized Read–Write Operations”. In: *IEEE Transactions on Magnetics* 55.1 (2019-01). Conference Name: IEEE Transactions on Magnetics, pp. 1–10. ISSN: 1941-0069. DOI: 10.1109/TMAG.2018.2875885 (cit. on p. 22).

[44] Sparsh Mittal, Jeffrey S. Vetter, and Dong Li. “A Survey Of Architectural Approaches for Managing Embedded DRAM and Non-Volatile On-Chip Caches”. In: *IEEE Transactions on Parallel and Distributed Systems* 26.6 (2015-06). ISSN: 2161-9883. DOI: 10.1109/TPDS.2014.2324563 (cit. on p. 22).

[45] Jalil Boukhobza et al. “Emerging NVM: A Survey on Architectural Integration and Research Challenges”. In: *ACM Transactions on Design Automation of Electronic Systems* 23 (2018-01). DOI: 10.1145/3131848 (cit. on p. 22).

[46] Gianlucca O. Puglia et al. “Non-Volatile Memory File Systems: A Survey”. In: *IEEE Access* 7 (2019), pp. 25836–25871. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2899463 (cit. on p. 22).

[47] Martin Hilbert and Priscila López. “The World’s Technological Capacity to Store, Communicate, and Compute Information”. en. In: *Science* 332.6025 (2011-04), pp. 60–65. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1200970 (cit. on p. 23).

[48] *Data Durability, and Back-up at scale: A tale of "the Tape"*. en. URL: https://community.ibm.com/community/user/storage/blogs/shawn-brume1/2020/07/14/data-durability-and-back-up-at-scale-a-tale-of-the (visited on 2022-07-13) (cit. on pp. 23, 24).

[49] Jingcheng Wang et al. "A 28-nm Compute SRAM With Bit-Serial Logic/Arithmetic Operations for Programmable In-Memory Vector Computing". In: *IEEE Journal of Solid-State Circuits* (2019-11), pp. 1–11. ISSN: 1558-173X. DOI: 10.1109/JSSC.2019.2939682 (cit. on pp. 25, 31).

[50] W. H. Kautz. "Cellular Logic-in-Memory Arrays". In: *IEEE Transactions on Computers* C-18.8 (1969-08), pp. 719–727. ISSN: 0018-9340. DOI: 10.1109/TC.1969.222754 (cit. on p. 25).

[51] H. S. Stone. "A Logic-in-Memory Computer". In: *IEEE Transactions on Computers* C-19.1 (1970-01), pp. 73–78. ISSN: 0018-9340. DOI: 10.1109/TC.1970.5008902 (cit. on p. 25).

[52] D. G. Elliott, W. M. Snelgrove, and M. Stumm. "Computational Ram: A Memory-simd Hybrid And Its Application To Dsp". In: *1992 Proceedings of the IEEE Custom Integrated Circuits Conference*. 1992 Proceedings of the IEEE Custom Integrated Circuits Conference. 1992-05, pp. 30.6.1–30.6.4. DOI: 10.1109/CICC.1992.591879 (cit. on p. 25).

[53] Wm. A. Wulf and Sally A. McKee. "Hitting the memory wall: implications of the obvious". In: *ACM SIGARCH Computer Architecture News* 23.1 (1995-03), pp. 20–24 (cit. on p. 25).

[54] M. Gokhale, B. Holmes, and K. Iobst. "Processing in memory: the Terasys massively parallel PIM array". In: *Computer* 28.4 (1995-04), pp. 23–31. ISSN: 0018-9162. DOI: 10.1109/2.375174 (cit. on p. 26).

[55] D. Patterson et al. "A case for intelligent RAM". In: *IEEE Micro* 17.2 (1997-03), pp. 34–44. ISSN: 0272-1732. DOI: 10.1109/40.592312 (cit. on p. 26).

[56] Yiran Chen. "Reshaping Future Computing Systems With Emerging Nonvolatile Memory Technologies". In: *IEEE Micro* 39.1 (2019-01), pp. 54–57. ISSN: 1937-4143. DOI: 10.1109/MM.2018.2885588 (cit. on p. 26).

[57] Roman Gauchi. "Exploration of Reconfigurable Tiles of Computing-in-Memory Architecture for Data-intensive Applications". en. PhD thesis. Université Grenoble Alpes: Université Grenoble Alpes, 2021-03. URL: https://tel.archives-ouvertes.fr/tel-03281795 (cit. on pp. 29, 52, 54, 72, 73, 79).

[58] K. C. Akyel et al. "DRC2: Dynamically Reconfigurable Computing Circuit based on memory architecture". In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 2016-10, pp. 1–8. DOI: 10.1109/ICRC.2016.7738698 (cit. on p. 31).

[59] S. Aga et al. "Compute Caches". In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2017-02, pp. 481–492. DOI: 10.1109/HPCA.2017.21 (cit. on pp. 31–33, 39, 41, 44).

[60] Jianmin Zeng et al. “DM-IMCA: A dual-mode in-memory computing architecture for general purpose processing”. In: *IEICE Electronics Express* 17.4 (2020), pp. 20200005–20200005. DOI: 10.1587/elex.17.20200005 (cit. on pp. 30, 44).

[61] R. Khaddam-Aljameh et al. “An SRAM-Based Multibit In-Memory Matrix-Vector Multiplier With a Precision That Scales Linearly in Area, Time, and Power”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2020), pp. 1–14. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2020.3037871 (cit. on p. 31).

[62] H. Chen et al. “Configurable 8T SRAM for Enbling in-Memory Computing”. In: *2019 2nd International Conference on Communication Engineering and Technology (ICCET)*. 2019-04, pp. 139–142. DOI: 10.1109/ICCET.2019.8726871 (cit. on p. 31).

[63] Zhiting Lin et al. “In-Memory Computing With Double Word Lines and Three Read Ports for Four Operands”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.5 (2020-05), pp. 1316–1320. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2020.2976099 (cit. on p. 31).

[64] A. Agrawal et al. “X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.12 (2018-12), pp. 4219–4232. ISSN: 1549-8328. DOI: 10.1109/TCSI.2018.2848999 (cit. on p. 31).

[65] C. Eckert et al. “Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks”. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 2018-06, pp. 383–396. DOI: 10.1109/ISCA.2018.00040 (cit. on p. 31).

[66] Y. Zhang et al. “Recryptor: A reconfigurable in-memory cryptographic Cortex-M0 processor for IoT”. In: *2017 Symposium on VLSI Circuits*. 2017-06, pp. C264–C265. DOI: 10.23919/VLSIC.2017.8008501 (cit. on pp. 31, 42).

[67] Z. Jiang et al. “XNOR-SRAM: In-Memory Computing SRAM Macro for Binary/Ternary Deep Neural Networks”. In: *2018 IEEE Symposium on VLSI Technology*. 2018-06, pp. 173–174. DOI: 10.1109/VLSIT.2018.8510687 (cit. on pp. 31, 46, 60, 61).

[68] J. Wang et al. “A Compute SRAM with Bit-Serial Integer/Floating-Point Operations for Programmable In-Memory Vector Acceleration”. In: *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*. San Francisco, USA, 2019-02, pp. 224–226. DOI: 10.1109/ISSCC.2019.8662419 (cit. on p. 31).

[69] D. Jeon et al. “A 23-mW Face Recognition Processor with Mostly-Read 5T Memory in 40-nm CMOS”. In: *IEEE Journal of Solid-State Circuits* 52.6 (2017-06), pp. 1628–1642. ISSN: 0018-9200. DOI: 10.1109/JSSC.2017.2661838 (cit. on p. 32).

[70] L. Fick et al. “Analog in-memory subthreshold deep neural network accelerator”. In: *2017 IEEE Custom Integrated Circuits Conference (CICC)*. 2017-04, pp. 1–4. DOI: 10.1109/CICC.2017.7993629 (cit. on p. 32).

[71] H. E. Sumbul et al. “A 2.9–33.0 TOPS/W Reconfigurable 1-D/2-D Compute-Near-Memory Inference Accelerator in 10-nm FinFET CMOS”. In: *IEEE Solid-State Circuits Letters* 3 (2020), pp. 118–121. ISSN: 2573-9603. DOI: 10.1109/LSSC.2020.3007185 (cit. on p. 32).

[72] A. Biswas and A. P. Chandrakasan. “Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications”. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. 2018-02, pp. 488–490. DOI: 10.1109/ISSCC.2018.8310397 (cit. on pp. 32, 46, 60, 61).

[73] J. Saikia et al. “K-Nearest Neighbor Hardware Accelerator Using In-Memory Computing SRAM”. In: *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 2019-07, pp. 1–6. DOI: 10.1109/ISLPED.2019.8824822 (cit. on p. 32).

[74] William Simon et al. “A Fast, Reliable and Wide-Voltage-Range In-Memory Computing Architecture”. In: ACM, 2019-02, p. 83. ISBN: 978-1-4503-6725-7. DOI: 10.1145/3316781.3317741 (cit. on p. 32).

[75] S. Srinivasa et al. “ROBIN: Monolithic-3D SRAM for Enhanced Robustness with In-Memory Computation Support”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.7 (2019-07), pp. 2533–2545. ISSN: 1549-8328. DOI: 10.1109/TCSI.2019.2897497 (cit. on p. 32).

[76] Onur Mutlu et al. “Processing data where it makes sense: Enabling in-memory computation”. In: *Microprocessors and Microsystems* 67 (2019-06), pp. 28–41. ISSN: 0141-9331. DOI: 10.1016/j.micpro.2019.01.009 (cit. on p. 32).

[77] S. Jeloka et al. “A 28 nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T Bit Cell Enabling Logic-in-Memory”. In: *IEEE Journal of Solid-State Circuits* 51.4 (2016-04), pp. 1009–1021. ISSN: 0018-9200. DOI: 10.1109/JSSC.2016.2515510 (cit. on p. 32).

[78] W. Khwa et al. “A 65nm 4Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3ns and 55.8TOPS/W fully parallel product-sum operation for binary DNN edge processors”. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. 2018-02, pp. 496–498. DOI: 10.1109/ISSCC.2018.8310401 (cit. on pp. 32, 60, 61).

[79] Jinseok Kim et al. “Area-Efficient and Variation-Tolerant In-Memory BNN Computing using 6T SRAM Array”. In: *2019 Symposium on VLSI Circuits*. ISSN: 2158-5636. 2019-06, pp. C118–C119. DOI: 10.23919/VLSIC.2019.8778160 (cit. on p. 32).

[80] J. Zhang, Z. Wang, and N. Verma. "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array". In: *IEEE Journal of Solid-State Circuits* 52.4 (2017-04), pp. 915–924. ISSN: 0018-9200. DOI: 10.1109/JSSC.2016.2642198 (cit. on pp. 33, 60, 61).

[81] K. Ando et al. "BRein memory: A 13-layer 4.2 K neuron/0.8 M synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm CMOS". In: *2017 Symposium on VLSI Circuits*. 2017-06, pp. C24–C25. DOI: 10.23919/VLSIC.2017.8008533 (cit. on pp. 33, 60, 61).

[82] M. Kang, S. K. Gonugondla, and N. R. Shanbhag. "A 19.4 nJ/decision 364K decisions/s in-memory random forest classifier in 6T SRAM array". In: *ESSCIRC 2017 - 43rd IEEE European Solid State Circuits Conference*. 2017-09, pp. 263–266. DOI: 10.1109/ESSCIRC.2017.8094576 (cit. on p. 33).

[83] Jian-Wei Su et al. "15.2 A 28nm 64Kb Inference-Training Two-Way Transpose Multibit 6T SRAM Compute-in-Memory Macro for AI Edge Chips". In: *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. 2020-02, pp. 240–242. DOI: 10.1109/ISSCC19947.2020.9062949 (cit. on p. 33).

[84] Vivek Seshadri et al. "RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization". In: *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-46. New York, NY, USA: Association for Computing Machinery, 2013-12, pp. 185–197. ISBN: 978-1-4503-2638-4. DOI: 10.1145/2540708.2540725 (cit. on pp. 33, 76).

[85] Dongping Zhang et al. "TOP-PIM: Throughput-oriented Programmable Processing in Memory". In: *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*. HPDC '14. New York, NY, USA: ACM, 2014, pp. 85–98. ISBN: 978-1-4503-2749-7. DOI: 10.1145/2600212.2600213 (cit. on p. 33).

[86] Vivek Seshadri et al. "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology". In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 273–287. ISBN: 978-1-4503-4952-9. DOI: 10.1145/3123939.3124544 (cit. on pp. 33, 34).

[87] Amirali Boroumand et al. "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks". en. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '18*. Williamsburg, VA, USA: ACM Press, 2018, pp. 316–331. ISBN: 978-1-4503-4911-6. DOI: 10.1145/3173162.3173177 (cit. on pp. 33, 35).

[88] Thomas Vogelsang. "Understanding the Energy Consumption of Dynamic Random Access Memories". In: *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. 2010-12, pp. 363–374. DOI: 10.1109/MICRO.2010.42 (cit. on pp. 33, 35).

[89] Vasileios Zois et al. “Massively Parallel Skyline Computation for Processing-in-memory Architectures”. In: *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. PACT ’18. New York, NY, USA: ACM, 2018, 1:1–1:12. ISBN: 978-1-4503-5986-3. DOI: 10.1145/3243176.3243187 (cit. on p. 34).

[90] Dominique Lavenier et al. *BLAST on UPMEM*. Research Report RR-8878. INRIA Rennes - Bretagne Atlantique, 2016-03, p. 20. URL: https://hal.archives-ouvertes.fr/hal-01294345 (visited on 2019-02-08) (cit. on p. 34).

[91] Mingxuan He et al. “Newton: A DRAM-maker’s Accelerator-in-Memory (AiM) Architecture for Machine Learning”. In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2020-10, pp. 372–385. DOI: 10.1109/MICRO50266.2020.00040 (cit. on p. 34).

[92] Shaahin Angizi and Deliang Fan. “GraphiDe: A Graph Processing Accelerator leveraging In-DRAM-Computing”. In: *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. GLSVLSI ’19. New York, NY, USA: Association for Computing Machinery, 2019-05, pp. 45–50. ISBN: 978-1-4503-6252-8. DOI: 10.1145/3299874.3317984 (cit. on p. 34).

[93] Shuangchen Li et al. “SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator”. In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2018-10, pp. 696–709. DOI: 10.1109/MICRO.2018.00062 (cit. on p. 34).

[94] *UPMEM*. 2017. URL: http://www.upmem.com/ (visited on 2017-10-23) (cit. on p. 34).

[95] Fabrice Devaux and Jean-François Roy. “Memory circuit with integrated processor”. en. US10324870B2. 2019-06 (cit. on p. 34).

[96] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. “ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs”. In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO ’52. New York, NY, USA: Association for Computing Machinery, 2019-10, pp. 100–113. ISBN: 978-1-4503-6938-1. DOI: 10.1145/3352460.3358260 (cit. on p. 34).

[97] Shanshan Xie et al. “16.2 eDRAM-CIM: Compute-In-Memory Design with Reconfigurable Embedded-Dynamic-Memory Array Realizing Adaptive Data Converters and Charge-Domain Computing”. In: *2021 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 64. 2021-02, pp. 248–250. DOI: 10.1109/ISSCC42613.2021.9365932 (cit. on p. 34).

[98] Samsung. *PIM | Technology*. en. 2021. URL: https://semiconductor.samsung.com/content/semiconductor/global/insights/technology/pim.html (visited on 2022-08-06) (cit. on p. 35).

[99] Samsung. *Samsung Brings In-Memory Processing Power to Wider Range of Applications*. en. 2021. URL: https://news.samsung.com/global/samsung-brings-in-memory-processing-power-to-wider-range-of-applications (visited on 2022-08-06) (cit. on p. 35).

[100] Peng Li, Kevin Gomez, and David J. Lilja. "Exploiting free silicon for energy-efficient computing directly in NAND flash-based solid-state storage systems". In: *2013 IEEE High Performance Extreme Computing Conference (HPEC)*. 2013-09, pp. 1–6. DOI: 10.1109/HPEC.2013.6670317 (cit. on pp. 35, 36).

[101] Panni Wang et al. "Three-Dimensional nand Flash for Vector–Matrix Multiplication". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.4 (2019-04), pp. 988–991. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2018.2882194 (cit. on p. 35).

[102] Hang-Ting Lue et al. "Optimal Design Methods to Transform 3D NAND Flash into a High-Density, High-Bandwidth and Low-Power Nonvolatile Computing in Memory (nvCIM) Accelerator for Deep-Learning Neural Networks (DNN)". In: *2019 IEEE International Electron Devices Meeting (IEDM)*. 2019-12. DOI: 10.1109/IEDM19573.2019.8993652 (cit. on p. 35).

[103] Wonbo Shim and Shimeng Yu. "Technological Design of 3D NAND-Based Compute-in-Memory Architecture for GB-Scale Deep Neural Network". In: *IEEE Electron Device Letters* 42.2 (2021-02), pp. 160–163. ISSN: 1558-0563. DOI: 10.1109/LED.2020.3048101 (cit. on pp. 35, 92).

[104] Won Ho Choi et al. "An In-Flash Binary Neural Network Accelerator with SLC NAND Flash Array". In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2020-10, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9180920 (cit. on p. 36).

[105] Wen Zhou et al. "Temporal Correlation Detection Based on 3D NAND Flash In-Memory Computing". In: *IEEE Electron Device Letters* 43.6 (2022-06), pp. 874–877. ISSN: 1558-0563. DOI: 10.1109/LED.2022.3170593 (cit. on p. 36).

[106] Minsu Kim et al. "An Embedded nand Flash-Based Compute-In-Memory Array Demonstrated in a Standard Logic Process". In: *IEEE Journal of Solid-State Circuits* 57.2 (2022-02), pp. 625–638. ISSN: 1558-173X. DOI: 10.1109/JSSC.2021.3098671 (cit. on p. 36).

[107] Samsung. *Smart SSD | SSD Card*. en. 2020. URL: https://semiconductor.samsung.com/content/semiconductor/global/ssd/smart-ssd.html (visited on 2022-08-09) (cit. on p. 36).

[108] S. Kvatinsky et al. "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.10 (2014-10), pp. 2054–2066. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2013.2282132 (cit. on pp. 36, 37, 92).

[109] S. Kvatinsky et al. “MAGIC—Memristor-Aided Logic”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 61.11 (2014-11), pp. 895–899. ISSN: 1549-7747. DOI: 10.1109/TCSII.2014.2357292 (cit. on pp. 36, 37).

[110] N. Talati et al. “Practical challenges in delivering the promises of real processing-in-memory machines”. In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018-03, pp. 1628–1633. DOI: 10.23919/DATE.2018.8342275 (cit. on pp. 37, 43).

[111] A. Haj-Ali et al. “Not in Name Alone: A Memristive Memory Processing Unit for Real In-Memory Processing”. In: *IEEE Micro* 38.5 (2018-09), pp. 13–21. ISSN: 0272-1732. DOI: 10.1109/MM.2018.053631137 (cit. on p. 37).

[112] A. Haj-Ali et al. “Efficient Algorithms for In-Memory Fixed Point Multiplication Using MAGIC”. In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2018-05, pp. 1–5. DOI: 10.1109/ISCAS.2018.8351561 (cit. on p. 37).

[113] R. Ben Hur et al. “Simple magic: Synthesis and in-memory Mapping of logic execution for memristor-aided logic”. In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2017-11, pp. 225–232. DOI: 10.1109/ICCAD.2017.8203782 (cit. on p. 37).

[114] P. Gaillardon et al. “The Programmable Logic-in-Memory (PLiM) computer”. In: *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2016-03, pp. 427–432. ISBN: 978-3-9815-3707-9 (cit. on p. 37).

[115] M. Abu Lebdeh et al. “An Efficient Heterogeneous Memristive xnor for In-Memory Computing”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 64.9 (2017-09), pp. 2427–2437. ISSN: 1549-8328. DOI: 10.1109/TCSI.2017.2706299 (cit. on p. 37).

[116] João Vieira et al. “A Product Engine for Energy-Efficient Execution of Binary Neural Networks Using Resistive Memories”. In: *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. 2019-10, pp. 160–165. DOI: 10.1109/VLSI-SoC.2019.8920343 (cit. on p. 37).

[117] Tianqi Tang et al. “Binary convolutional neural network on RRAM”. In: *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2017-01, pp. 782–787. DOI: 10.1109/ASPDAC.2017.7858419 (cit. on p. 38).

[118] Mahdi Nazm Bojnordi and Engin Ipek. “Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning”. In: *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2016-03, pp. 1–13. DOI: 10.1109/HPCA.2016.7446049 (cit. on pp. 38, 42).

[119] Jaesung Park et al. “TiOx-Based RRAM Synapse With 64-Levels of Conductance and Symmetric Conductance Change by Adopting a Hybrid Pulse Scheme for Neuromorphic Computing”. In: *IEEE Electron Device Letters* 37.12 (2016-12), pp. 1559–1562. ISSN: 1558-0563. DOI: 10.1109/LED.2016.2622716 (cit. on p. 38).

[120] F. Su et al. “A 462GOPs/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory”. In: *2017 Symposium on VLSI Circuits*. 2017-06, pp. C260–C261. DOI: 10.23919/VLSIC.2017.8008585 (cit. on p. 38).

[121] W. Chen et al. “A 16Mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme”. In: *2017 IEEE International Electron Devices Meeting (IEDM)*. 2017-12. DOI: 10.1109/IEDM.2017.8268468 (cit. on p. 38).

[122] W. Chen et al. “A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors”. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. 2018-02, pp. 494–496. DOI: 10.1109/ISSCC.2018.8310400 (cit. on p. 38).

[123] Qi Liu et al. “33.2 A Fully Integrated Analog ReRAM Based 78.4TOPS/W Compute-In-Memory Chip with Fully Parallel MAC Computing”. In: *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. 2020-02, pp. 500–502. DOI: 10.1109/ISSCC19947.2020.9062953 (cit. on p. 38).

[124] Zhuo-Rui Wang et al. “Efficient Implementation of Boolean and Full-Adder Functions With 1T1R RRAMs for Beyond Von Neumann In-Memory Computing”. In: *IEEE Transactions on Electron Devices* 65.10 (2018-10), pp. 4659–4666. ISSN: 1557-9646. DOI: 10.1109/TED.2018.2866048 (cit. on pp. 39, 44).

[125] A. Sebastian et al. “Computational memory-based inference and training of deep neural networks”. In: *2019 Symposium on VLSI Technology*. ISSN: 2158-9682. 2019-06, T168–T169. DOI: 10.23919/VLSIT.2019.8776518 (cit. on pp. 39, 92).

[126] Jing Li et al. “1 Mb 0.41 µm² 2T-2R Cell Nonvolatile TCAM With Two-Bit Encoding and Clocked Self-Referenced Sensing”. en. In: *IEEE Journal of Solid-State Circuits* 49.4 (2014-04), pp. 896–907. ISSN: 0018-9200, 1558-173X. DOI: 10.1109/JSSC.2013.2292055 (cit. on p. 39).

[127] P. Narayanan et al. “Fully On-Chip MAC at 14 nm Enabled by Accurate Row-Wise Programming of PCM-Based Weights and Parallel Vector-Transport in Duration-Format”. In: *IEEE Transactions on Electron Devices* 68.12 (2021-12), pp. 6629–6636. ISSN: 1557-9646. DOI: 10.1109/TED.2021.3115993 (cit. on p. 39).

[128] Riduan Khaddam-Aljameh et al. “HERMES-Core—A 1.59-TOPS/mm2 PCM on 14-nm CMOS In-Memory Compute Core Using 300-ps/LSB Linearized CCO-Based ADCs”. In: *IEEE Journal of Solid-State Circuits* 57.4 (2022-04), pp. 1027–1038. ISSN: 1558-173X. DOI: 10.1109/JSSC.2022.3140414 (cit. on p. 39).

[129] Wang Kang et al. “In-Memory Processing Paradigm for Bitwise Logic Operations in STT–MRAM”. In: *IEEE Transactions on Magnetics* 53.11 (2017-11), pp. 1–4. ISSN: 1941-0069. DOI: 10.1109/TMAG.2017.2703863 (cit. on p. 39).

[130] Shubham Jain et al. “Computing in Memory With Spin-Transfer Torque Magnetic RAM”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.3 (2018-03), pp. 470–483. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2017.2776954 (cit. on pp. 39, 92).

[131] Masoud Zabihi et al. “In-Memory Processing on the Spintronic CRAM: From Hardware Design to Application Mapping”. In: *IEEE Transactions on Computers* 68.8 (2019-08), pp. 1159–1173. ISSN: 1557-9956. DOI: 10.1109/TC.2018.2858251 (cit. on p. 39).

[132] Tifenn Hirtzlin et al. “Stochastic Computing for Hardware Implementation of Binarized Neural Networks”. In: *IEEE Access* 7 (2019), pp. 76394–76403. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2921104 (cit. on p. 39).

[133] Tung-Cheng Chang et al. “13.4 A 22nm 1Mb 1024b-Read and Near-Memory-Computing Dual-Mode STT-MRAM Macro with 42.6GB/s Read Bandwidth for Security-Aware Mobile Devices”. In: *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. 2020-02, pp. 224–226. DOI: 10.1109/ISSCC19947.2020.9063072 (cit. on p. 40).

[134] Peter Deaville et al. “A Maximally Row-Parallel MRAM In-Memory-Computing Macro Addressing Readout Circuit Sensitivity and Area”. In: *ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC)*. 2021-09, pp. 75–78. DOI: 10.1109/ESSCIRC53450.2021.9567807 (cit. on p. 40).

[135] Shuangchen Li et al. “Pinatubo: A Processing-in-memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories”. In: *Proceedings of the 53rd Annual Design Automation Conference*. DAC ’16. New York, NY, USA: ACM, 2016, 173:1–173:6. ISBN: 978-1-4503-4236-0. DOI: 10.1145/2897937.2898064 (cit. on p. 40).

[136] M. Imani, S. Gupta, and T. Rosing. “GenPIM: Generalized processing in-memory to accelerate data intensive applications”. In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018-03, pp. 1155–1158. DOI: 10.23919/DATE.2018.8342186 (cit. on p. 40).

[137] Shun Okamoto et al. “Application Driven SCM and NAND Flash Hybrid SSD Design for Data-Centric Computing System”. In: *2015 IEEE International Memory Workshop (IMW)*. 2015-05, pp. 1–4. DOI: 10.1109/IMW.2015.7150277 (cit. on p. 40).

[138] Ken Takeuchi. "Data-aware NAND flash memory for intelligent computing with deep neural network". In: *2017 IEEE International Electron Devices Meeting (IEDM)*. 2017-12, pp. 28.4.1–28.4.4. DOI: 10.1109/IEDM.2017.8268470 (cit. on p. 40).

[139] Linbin Chen et al. "CCE: A Combined SRAM and Non Volatile Cache for Endurance of Next Generation Multilevel Non Volatile Memories in Embedded Systems". In: *Proceedings of the 14th IEEE/ACM International Symposium on Nanoscale Architectures*. NANOARCH '18. New York, NY, USA: ACM, 2018, pp. 58–64. ISBN: 978-1-4503-5815-6. DOI: 10.1145/3232195.3232196 (cit. on p. 40).

[140] Xueyong Zhang, Vivek Mohan, and Arindam Basu. "CRAM: Collocated SRAM and DRAM With In-Memory Computing-Based Denoising and Filling for Neuromorphic Vision Sensors in 65 nm CMOS". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 67.5 (2020-05), pp. 816–820. ISSN: 1558-3791. DOI: 10.1109/TCSII.2020.2980125 (cit. on p. 40).

[141] Marco Rios et al. "Running Efficiently CNNs on the Edge Thanks to Hybrid SRAM-RRAM In-Memory Computing". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2021-02, pp. 1881–1886. DOI: 10.23919/DATE51398.2021.9474233 (cit. on p. 40).

[142] Hwajung Kim, Heon Y. Yeom, and Hanul Sung. "Understanding the Performance Characteristics of Computational Storage Drives: A Case Study with SmartSSD". en. In: *Electronics* 10.21 (2021-10), p. 2617. ISSN: 2079-9292. DOI: 10.3390/electronics10212617 (cit. on p. 40).

[143] Di Gao et al. "Eva-CiM: A System-Level Performance and Energy Evaluation Framework for Computing-in-Memory Architectures". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020), pp. 1–1. ISSN: 1937-4151. DOI: 10.1109/TCAD.2020.2966484 (cit. on pp. 40, 44).

[144] R. Gauchi et al. "Reconfigurable tiles of computing-in-memory SRAM architecture for scalable vectorization". In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. ISLPED '20. New York, NY, USA: Association for Computing Machinery, 2020-08, pp. 121–126. ISBN: 978-1-4503-7053-0. DOI: 10.1145/3370748.3406550 (cit. on pp. 41, 79).

[145] Junwhan Ahn et al. "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture". In: *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 2015-06, pp. 336–348. DOI: 10.1145/2749469.2750385 (cit. on p. 41).

[146] Maciej Besta et al. "SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems". In: *arXiv:2104.07582 [cs]* (2021-04). URL: http://arxiv.org/abs/2104.07582 (visited on 2021-04-26) (cit. on p. 41).

[147] J. v Lunteren et al. “Coherently Attached Programmable Near-Memory Acceleration Platform and its application to Stencil Processing”. In: *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2019-03, pp. 668–673. DOI: 10.23919/DATE.2019.8715088 (cit. on p. 41).

[148] Liang Chang et al. “DASM: Data-Streaming-Based Computing in Nonvolatile Memory Architecture for Embedded System”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.9 (2019-09), pp. 2046–2059. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2019.2912941 (cit. on p. 41).

[149] N. Verma et al. “In-Memory Computing: Advances and Prospects”. In: *IEEE Solid-State Circuits Magazine* 11.3 (2019), pp. 43–55. ISSN: 1943-0590. DOI: 10.1109/MSSC.2019.2922889 (cit. on pp. 41, 42, 59–61).

[150] Chuan-Jia Jhang et al. “Challenges and Trends of SRAM-Based Computing-In-Memory for AI Edge Devices”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.5 (2021-05), pp. 1773–1786. ISSN: 1558-0806. DOI: 10.1109/TCSI.2021.3064189 (cit. on p. 42).

[151] Abu Sebastian et al. “Memory devices and applications for in-memory computing”. en. In: *Nature Nanotechnology* 15.7 (2020-07), pp. 529–544. ISSN: 1748-3395. DOI: 10.1038/s41565-020-0655-z (cit. on p. 42).

[152] M. Aamir, Somya Sharma, and Anuj Grover. “ChaCha20-in-Memory for Side-Channel Resistance in IoT Edge-Node Devices”. In: *IEEE Open Journal of Circuits and Systems* 2 (2021), pp. 833–842. ISSN: 2644-1225. DOI: 10.1109/OJCAS.2021.3127273 (cit. on p. 42).

[153] I. Giannopoulos et al. “8-bit Precision In-Memory Multiplication with Projected Phase-Change Memory”. In: *2018 IEEE International Electron Devices Meeting (IEDM)*. 2018-12, pp. 27.7.1–27.7.4. DOI: 10.1109/IEDM.2018.8614558 (cit. on p. 44).

[154] T. B. Preußer et al. “Inference of quantized neural networks on heterogeneous all-programmable devices”. In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018-03, pp. 833–838. DOI: 10.23919/DATE.2018.8342121 (cit. on p. 51).

[155] M. Kooli et al. “Smart instruction codes for in-memory computing architectures compatible with standard SRAM interfaces”. In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018-03, pp. 1634–1639. DOI: 10.23919/DATE.2018.8342276 (cit. on pp. 52, 79).

[156] Henry S. Warren. *Hacker’s delight*. en. 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2013. ISBN: 978-0-321-84268-8 (cit. on p. 53).

[157] Andrew Waterman and Krste Asanovic. *RISC-V Unprivileged ISA specification*. en. Tech. rep. 1. RISC-V Organisation, 2019, p. 238. URL: https://riscv.org/technical/specifications/ (visited on 2022-09-26) (cit. on p. 53).

[158] Yu-Hsin Chen et al. “14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks”. In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. ISSN: 2376-8606. 2016-01, pp. 262–263. DOI: 10.1109/ISSCC.2016.7418007 (cit. on pp. 60, 61).

[159] Zhe Yuan et al. “Sticker: A 0.41-62.1 TOPS/W 8Bit Neural Network Processor with Multi-Sparsity Compatible Convolution Arrays and Online Tuning Acceleration for Fully Connected Layers”. In: *2018 IEEE Symposium on VLSI Circuits*. ISSN: 2158-5601. 2018-06, pp. 33–34. DOI: 10.1109/VLSIC.2018.8502404 (cit. on pp. 60, 61).

[160] Shouyi Yin et al. “A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications”. In: *2017 Symposium on VLSI Circuits*. 2017-06, pp. C26–C27. DOI: 10.23919/VLSIC.2017.8008534 (cit. on pp. 60, 61).

[161] Dongjoo Shin et al. “14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks”. In: *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. ISSN: 2376-8606. 2017-02, pp. 240–241. DOI: 10.1109/ISSCC.2017.7870350 (cit. on pp. 60, 61).

[162] B. Moons et al. “14.5 Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI”. In: *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. 2017-02, pp. 246–247. DOI: 10.1109/ISSCC.2017.7870353 (cit. on pp. 60, 61).

[163] Jinmook Lee et al. “UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision”. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. ISSN: 2376-8606. 2018-02, pp. 218–220. DOI: 10.1109/ISSCC.2018.8310262 (cit. on pp. 60, 61).

[164] Daniel Bankman et al. “An always-on 3.8μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28nm CMOS”. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. ISSN: 2376-8606. 2018-02, pp. 222–224. DOI: 10.1109/ISSCC.2018.8310264 (cit. on pp. 60, 61).

[165] Hossein Valavi et al. “A Mixed-Signal Binarized Convolutional-Neural-Network Accelerator Integrating Dense Weight Storage and Multiplication for Reduced Data Movement”. In: *2018 IEEE Symposium on VLSI Circuits*. 2018 IEEE Symposium on VLSI Circuits. 2018-06, pp. 141–142. DOI: 10.1109/VLSIC.2018.8502421 (cit. on pp. 60, 61).

[166] Jean-Philippe Noel et al. “Method and device for designing a computational memory circuit”. 2021156420A1. 2021-08 (cit. on p. 62).

[167] J.-P. Noel et al. “Computational SRAM Design Automation using Pushed-Rule Bitcells for Energy-Efficient Vector Processing”. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. ISSN: 1558-1101. Grenoble, France, 2020-03, pp. 1187–1192. DOI: 10.23919/DATE48585.2020.9116506 (cit. on pp. 62, 90).

[168] Bruce Fleischer and Sunil Shukla. *Approximate Computing for On-Chip AI Acceleration: IBM Research at VLSI*. en-US. 2018-06. URL: https://www.ibm.com/blogs/research/2018/06/approximate-computing-ai-acceleration/ (visited on 2022-08-31) (cit. on p. 64).

[169] Louis-Noel Pouchet and Tomofumi Yuki. *PolyBench/C – The polyhedral benchmark suite*. 2015. URL: https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/ (visited on 2022-08-23) (cit. on p. 64).

[170] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. 2013. URL: https://pjreddie.com/darknet/ (visited on 2022-08-23) (cit. on pp. 65, 67).

[171] Wikipedia. *Bitap algorithm*. en. Page Version ID: 1063296095. 2022-01. URL: https://en.wikipedia.org/w/index.php?title=Bitap_algorithm&oldid=1063296095 (visited on 2022-08-31) (cit. on p. 65).

[172] *BLAS (Basic Linear Algebra Subprograms)*. 2021. URL: https://netlib.org/blas/ (visited on 2022-08-23) (cit. on p. 65).

[173] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009-06, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on p. 67).

[174] Anant Agarwal, John Hennessy, and Mark Horowitz. “Analytical cache model”. In: *ACM Transactions on Computer Systems (TOCS)* 7 (1989-05), pp. 184–215. DOI: 10.1145/63404.63407 (cit. on p. 68).

[175] Fei Guo and Yan Solihin. “An analytical model for cache replacement policy performance”. In: vol. 34. 2006-06, pp. 228–239. DOI: 10.1145/1140103.1140304 (cit. on p. 68).

[176] B.L. Jacob et al. “An analytical model for designing memory hierarchies”. In: *IEEE Transactions on Computers* 45.10 (1996-10), pp. 1180–1194. ISSN: 00189340. DOI: 10.1109/12.543711 (cit. on p. 68).

[177] Wikichip. *Skylake (client) - Microarchitectures - Intel - WikiChip*. en. 2015. URL: https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client) (visited on 2022-09-01) (cit. on p. 69).

[178] Linux Kernel Developers. *Perf Wiki*. 2022. URL: https://perf.wiki.kernel.org/index.php/Main_Page (visited on 2022-08-24) (cit. on p. 68).

[179] *Performance Application Programming Interface*. 2022. URL: https://icl.utk.edu/papi/software/ (visited on 2022-08-24) (cit. on p. 68).

[180] *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*. Developer's manual Volume 3B, Part 2. 2021, p. 582 (cit. on pp. 68, 69, 86).

[181] John McCalpin. *John McCalpin's blog " Blog Archive " Notes on the mystery of hardware cache performance counters*. URL: https://sites.utexas.edu/jdm4372/2013/07/14/notes-on-the-mystery-of-hardware-cache-performance-counters/ (visited on 2021-07-05) (cit. on p. 69).

[182] Fabrice Bellard. "QEMU, a Fast and Portable Dynamic Translator". en. In: *USENIX Annual Technical Conference* (2005), p. 6 (cit. on pp. 71, 73).

[183] Nathan Binkert et al. "The gem5 simulator". en. In: *ACM SIGARCH Computer Architecture News* 39.2 (2011-05), pp. 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718 (cit. on pp. 72, 73).

[184] Anastasiia Butko et al. "Accuracy evaluation of GEM5 simulator system". In: *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC). 2012-07, pp. 1–7. DOI: 10.1109/ReCoSoC.2012.6322869 (cit. on p. 72).

[185] Daniel Sanchez and Christos Kozyrakis. "ZSim: fast and accurate microarchitectural simulation of thousand-core systems". en. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. Tel-Aviv Israel: ACM, 2013-06, pp. 475–486. ISBN: 978-1-4503-2079-5. DOI: 10.1145/2485922.2485963 (cit. on pp. 72, 73).

[186] Chi-Keung Luk et al. "Pin: building customized program analysis tools with dynamic instrumentation". In: *ACM SIGPLAN Notices* 40.6 (2005-06), pp. 190–200. ISSN: 0362-1340. DOI: 10.1145/1064978.1065034 (cit. on pp. 72, 73, 79, 85, 89, 90).

[187] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation". en. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*. Seattle, Washington: ACM Press, 2011, p. 1. ISBN: 978-1-4503-0771-0. DOI: 10.1145/2063384.2063454 (cit. on pp. 72, 73).

[188] C. Lattner and V. Adve. "LLVM: A compilation framework for lifelong program analysis & transformation". en. In: *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* San Jose, CA, USA: IEEE, 2004, pp. 75–86. ISBN: 978-0-7695-2102-2. DOI: 10.1109/CGO.2004.1281665 (cit. on pp. 72, 73).

[189] M. Kooli et al. "Software Platform Dedicated for In-Memory Computing Circuit Evaluation". In: *2017 International Symposium on Rapid System Prototyping (RSP)*. 2017-10, pp. 43–49. ISBN: 978-1-4503-5418-9 (cit. on p. 72).

[190] Intel. *Pin – A Dynamic Binary Instrumentation Tool*. URL: https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html (visited on 2022-04-25) (cit. on pp. 72, 79).

[191] Xiangyu Dong et al. "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 (2012-07), pp. 994–1007. ISSN: 1937-4151. DOI: 10.1109/TCAD.2012.2185930 (cit. on pp. 73–75, 78, 90).

[192] S.J.E. Wilton and N.P. Jouppi. "CACTI: an enhanced cache access and cycle time model". In: *IEEE Journal of Solid-State Circuits* 31.5 (1996-05), pp. 677–688. ISSN: 1558-173X. DOI: 10.1109/4.509850 (cit. on pp. 73, 74).

[193] ITRS. *ITRS Models and Papers*. 2015. URL: http://www.itrs2.net/itrs-models-and-papers.html (visited on 2022-09-03) (cit. on p. 74).

[194] Youngdon Choi et al. "A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth". In: *2012 IEEE International Solid-State Circuits Conference*. 2012-02, pp. 46–48. DOI: 10.1109/ISSCC.2012.6176872 (cit. on pp. 74, 75).

[195] R. Gauchi et al. "Memory Sizing of a Scalable SRAM In-Memory Computing Tile Based Architecture". In: *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. 2019-10, pp. 166–171. DOI: 10.1109/VLSI-SoC.2019.8920373 (cit. on pp. 75, 77).

[196] Jinsong Ji, Chao Wang, and Xuehai Zhou. "System-Level Early Power Estimation for Memory Subsystem in Embedded Systems". In: 2008-11-08, pp. 370–375. DOI: 10.1109/SEC.2008.48 (cit. on p. 76).

[197] Yoongu Kim, Weikun Yang, and Onur Mutlu. "Ramulator: A Fast and Extensible DRAM Simulator". In: *IEEE Computer Architecture Letters* 15.1 (2016-01), pp. 45–49. ISSN: 1556-6064. DOI: 10.1109/LCA.2015.2414456 (cit. on pp. 76, 77).

[198] *Ramulator: A DRAM Simulator*. 2022-03-01. URL: https://github.com/CMU-SAFARI/ramulator (visited on 2022-03-04) (cit. on pp. 76, 77).

[199] *VAMPIRE*. 2022-02-11. URL: https://github.com/CMU-SAFARI/VAMPIRE (visited on 2022-03-04) (cit. on pp. 76, 77).

[200] Saugata Ghose et al. "What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study". en. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2.3 (2018-12), pp. 1–41. ISSN: 2476-1249, 2476-1249. DOI: 10.1145/3224419 (cit. on pp. 76, 77, 90).

[201] Karthik Chandrasekar et al. *DRAMPower*. URL: http://www.drampower.info (visited on 2022-03-04) (cit. on pp. 76, 77).

[202] *DRAM Power Model (DRAMPower)*. 2022-02-11. URL: https://github.com/tukl-msd/DRAMPower (visited on 2022-03-04) (cit. on pp. 76, 77).

[203] Matt Poremba and Yuan Xie. "NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories". In: *2012 IEEE Computer Society Annual Symposium on VLSI*. 2012 IEEE Computer Society Annual Symposium on VLSI. 2012-08, pp. 392–397. DOI: 10.1109/ISVLSI.2012.82 (cit. on pp. 76, 77).

[204] Matthew Poremba, Tao Zhang, and Yuan Xie. "NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems". In: *IEEE Computer Architecture Letters* 14.2 (2015-07), pp. 140–143. ISSN: 1556-6064. DOI: 10.1109/LCA.2015.2402435 (cit. on p. 76).

[205] P Rosenfeld, E Cooper-Balis, and B Jacob. "DRAMSim2: A Cycle Accurate Memory System Simulator". In: *IEEE Computer Architecture Letters* 10.1 (2011-01), pp. 16–19. ISSN: 1556-6056. DOI: 10.1109/L-CA.2011.4 (cit. on pp. 76, 77).

[206] *umd-memsys/DRAMSim2*. 2022-03-04. URL: https://github.com/umd-memsys/DRAMSim2 (visited on 2022-03-04) (cit. on pp. 76, 77).

[207] Matthias Jung, Christian Weis, and Norbert Wehn. "DRAMSys: A Flexible DRAM Subsystem Design Space Exploration Framework". In: *IPSJ Transactions on System LSI Design Methodology* 8 (2015), pp. 63–74. DOI: 10.2197/ipsjtsldm.8.63 (cit. on pp. 76, 77).

[208] *DRAMSys - Fraunhofer IESE*. Fraunhofer Institute for Experimental Software Engineering IESE. URL: https://www.iese.fraunhofer.de/en/innovation_trends/autonomous-systems/memtonomy/DRAMSys.html (visited on 2022-03-12) (cit. on pp. 76, 77).

[209] Lukas Steiner et al. "DRAMSys4.0: A Fast and Cycle-Accurate SystemC/TLM-Based DRAM Simulator". In: *Embedded Computer Systems: Architectures, Modeling, and Simulation*. Ed. by Alex Orailoglu, Matthias Jung, and Marc Reichenbach. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 110–126. ISBN: 978-3-030-60939-9. DOI: 10.1007/978-3-030-60939-9_8 (cit. on p. 77).

[210] *tukl-msd/DRAMSys*. 2022-03-08. URL: https://github.com/tukl-msd/DRAMSys (visited on 2022-03-12) (cit. on p. 77).

[211] *SEAL-UCSB/NVmain*. 2022-01-31. URL: https://github.com/SEAL-UCSB/NVmain (visited on 2022-03-04) (cit. on p. 77).

[212] Valentin Egloff et al. "Storage Class Memory with Computing Row Buffer: A Design Space Exploration". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2021-02, pp. 1–6. DOI: 10.23919/DATE51398.2021.9473992 (cit. on pp. 85, 92, 104, 119).

[213] Valentin EGLOFF, Jean-Philippe Noel, and Jean-Michel PORTAL. "Device comprising a non-volatile memory circuit". 4036916A1. 2022-08 (cit. on p. 104).

[214] Maha Kooli et al. “Towards a Truly Integrated Vector Processing Unit for Memory-bound Applications Based on a Cost-competitive Computational SRAM Design Solution”. In: *ACM Journal on Emerging Technologies in Computing Systems* 18.2 (2022-04), 40:1–40:26. ISSN: 1550-4832. DOI: 10.1145/3485823. (Visited on 2022-10-04).

Appendices

Contents

Appendix A.

Résumé long

Cette thèse est divisée en cinq chapitres qui vous présenteront *pourquoi* je fais cette thèse : le contexte global de l'industrie semiconducteur soulève des questions sur la durabilité du modèle actuel ainsi que sur les performances et l'efficacité énergétique du calcul en général (chapitre 1). Ceci appelle une solution innovante et prometteuse telle que peut l'être le calcul en mémoire. Le calcul en mémoire est une solution, qui plus est, compatible avec les mémoires non volatiles émergentes qui apportent des améliorations technologiques dans le monde de l'informatique. Nous étudions l'état de l'art dans le chapitre 2 et montrons comment la plupart des papiers passent à côté de deux points clés sur l'endurance des mémoires non volatiles et sur où se trouve physiquement la donnée dans le système. Nous proposons notre solution, une enveloppe numérique autour d'une mémoire statique à accès aléatoire (SRAM) que nous appellons C-SRAM (chapitre 3). Notre C-SRAM peut ensuite être étroitement couplée à une mémoire non volatile ou une mémoire de stockage de masse. Pour réaliser une évaluation architecturale, nous développons une plateforme de simulation nourrie avec des paramètres technoloqiques tirés de l'état de l'art et de nos propres travaux (chapitre 4). En combinant nos différents travaux de design et de simulation numériques, nous démontrons que calculer au sommet de la hiérarchie mémoire, c'est à dire proche du stockage de masse permanent, donne les meilleurs gains concernant le temps d'exécution et l'énergie consommée (chapitre 5).

Chapitre 1 : Sur l'industrie des semiconducteurs

Nous montrons à partir d'une analyse de l'état de l'art que la tendance de la technologie CMOS actuelle arrive à un cul de sac dû à la fin de la réduction possible de la taille du nœud. Plusieurs problèmes en découlent. Premièrement, la densité de puissance continue d'augmenter depuis la fin de la loi de Dennard ce qui pose des problèmes de dissipation de chaleur. Pour éviter la surchauffe, certaines parties d'un circuit ne peuvent fonctionner ensemble, ce que l'on appelle le *silicium noir*. L'un des effets les plus visibles est la fin de l'augmentation de la fréquence des processeurs en 2005 car les circuits ne pouvaient plus être refroidis suffisamment. Deuxièmement, l'accélération de la demande pour plus de puissance de calcul a mené à de nombreuses percées architecturales. Lorsque les fréquences n'ont pu être augmentées, les processeurs multicœurs furent introduits et les puces modernes peuvent atteindre plusieurs dizaines de cœurs. Les jeux d'instructions Une Instruction Plusieurs Données (SIMD) ont suivis pour répondre à une demande toujours plus pressante de calcul avec des largeures de vecteur atteignant aujourd'hui 512 bits. Les prédicteurs de branchement, l'accès à la

mémoire par anticipation et l'exécution spéculative sont toutes des améliorations matérielles pour augmenter la vitesse de traitement. Cependant, cette course perpétuelle mena à une diminution de la sécurité matérielle avec plusieurs vulnérabilités telles que Spectre et Meltdown. Finalement, les performances mémoires, principalement celles de la mémoire dynamique (DRAM), disques durs et Flash, n'ont pas évoluées parallèlement. Alors que les performances des processeurs ont été multipliées par 1000 depuis 1980, celles de la DRAM ne l'ont été que d'un facteur 10. Cet écart de performance est appelé le mur mémoire car il s'agit d'un mur de performance non dépassable par aucun processeur, indépendemment du nombre de cœurs et de toute autre amélioration matérielle. Un problème parallèle est le goulot d'étranglement de von Neumann car toutes les données doivent traverser la hierarchie mémoire en partant de larges lignes mémoire vers des registres processeur scalaires. Pour réduire l'écart de performances entre la mémoire et le processeur, les caches ont été étendus en taille mémoire et en profondeur, allant aujourd'hui jusqu'à 4 niveaux de cache. Ils sont basés sur les principes de localité spatiale et temporelle pour *cacher* les données voisines qui sont plus susceptibles d'être utilisées ou les mêmes données qui peuvent être réutilisées juste après. La contrepartie étant que, désormais, les données doivent traverser tous les caches avant de pouvoir être utilisées par le processeur et réciproquement pour la réécriture en mémoire du résultat. Cet effet est empiré par les transferts gros grains, typiquement 64 octets sont transferrés au niveau des caches, même si seulement un seul octet est utilisé. En conséquence, les accès aux données sont, en terme de coûts énergétiques et de temps d'exécution, les principaux coupables, étant des centaines de fois plus coûteux que le calcul en lui-même. Il est à noter que l'état de l'art mentionne que ces accès mémoire représentent en moyenne entre 50 % et 80 % du budget énergétique total du système. Bien que le mur mémoire existe depuis plus de 40 ans, les nouvelles technologies mémoire sont prévues pour réduire cet écart. Le cube mémoire hybride (HMC) et la mémoire haute bande passante (HBM) sont de récentes évolutions des mémoires dynamiques qui démultiplient la bande passante en utilisant des bus plus larges et exploitant l'empilement 3D. Malgré une bande passante largement augmentée, le goulot d'étranglement de von Neumann demeure car l'architecture reste identique et est centrée sur le calcul plutôt que sur la donnée. De plus, la nature dynamique de la DRAM n'est pas éliminée et est toujours l'un des gouffres énergétiques. Les mémoires non volatiles (NVM) émergentes peuvent partiellement résoudre ces problèmes, mais une solution globale requiert un changement de paradigme vers des architectures centrées sur les données. À cette fin, le calcul en mémoire est une solution prometteuse car il peut, si non complètement, au moins largement réduire les mouvements de données dans l'architecture. Si utilisé avec des NVMs, il peut également diminuer drastiquement la consommation globale des mémoires. De plus, les mémoires non volatiles ont des performances prometteuses, tant en énergie qu'en temps d'accès, intermédiaires entre la SRAM et la DRAM ce qui pourra permettre de supprimer certaines mémoires de la hiérarchie, incluant la DRAM et éventuellement le cache de niveau 3.

Chapitre 2 : État de l'art

L'état de l'art révèle pléthore de techniques et méthodes pour implémenter le calcul en mémoire (IMC) dans toutes les technologies mémoires et à tous les niveaux de la hiérarchie. On peut les classifier dans différentes catégories selon où le calcul prend place, que cela soit dans le tableau mémoire au niveau analogique, ou après les amplificateurs de lecture dans le domaine numérique ou bien un mélange des deux. Les solutions à base de SRAM reposent sur les deux formes de calcul en mémoire : IMC et calcul proche mémoire. Nous faisons la distinction entre la cellule 6 transistors (6T) standard largement utilisée dans l'industrie et les cellules modifiées (8T, 9T, 10T, etc.) qui sont spécifiquement designées pour l'IMC. La cible des solutions proposées est principalement sur des applications spécifiques, notamment l'intelligence artificielle (IA) dont l'apprentissage profond est une seule solution générique. Cependant, en utilisant uniquement de la SRAM, ces solutions se placent d'elles-même tout en bas de la hiérarchie mémoire, c'est à dire proche du processeur, ou ne ciblent que des dispositifs périphériques avec une hiérarchie mémoire plate. En tant que telles, elles n'addressent pas les mouvements de données à travers la hiérarchie mémoire et ne font preuves que de faibles améliorations par rapport à leurs références.

Nous passons maintenant à la mémoire dynamique (DRAM) et présentons de multiples solutions allant du *vrai* IMC au traitement en mémoire (PIM) où le calcul est complètement en dehors du circuit mémoire. Néanmoins, la majorité des articles sont purement simulations et, pour autant que je sache, il n'y a que 3 solutions démontrées incluant une solution déjà commercialisée. Comme la DRAM utilise un procédé de fabrication différent des CMOS conventionnels, il est plus compliqué d'y intégrer les solutions éventuelles. De plus, en DRAM la lecture est destructive ce qui entraîne des accès supplémentaires pour copier les données avant d'y appliquer des traitements. Enfin, la DRAM est l'une des plus grosses consommations d'énergie dans les centres de données ou les appareils grand public. Exécuter le calcul dans cette technologie mémoire ne résoud que partiellement le goulot d'étranglement de von Neumann car les données ne sont pas originaires de cette mémoire.

Concernant la NAND Flash, qui est une mémoire purement électronique (sans partie mécanique) contrairement aux disques durs, la plupart des designs se concentrent sur l'opération de multiplication matrice-vecteur (MVM). Ils utilisent pour cela la sommation des courants comme moyen de calcul analogique ce qui est trop restrictif. De plus, l'endurance limitée de la Flash n'est pas considérée comme un problème. Cependant, sa haute densité permet à l'IMC en mémoire Flash d'être une solution pertinente pour les dispositifs périphériques utilisant de grands réseaux de neurones. Les mémoires résistives (RRAM) offrent les meilleurs promesses d'intégration vis-à-vis de l'IMC car les cellules mémoires sont disposées de manière appropriée pour créer des fonctions logiques. Les tableaux mémoires en grille connectée (*crossbar*) sont l'implémentation mémoire la plus dense, en excluant les technologies 3D. Comme toutes les technologies de mémoire résistive, de nombreuses solutions utilisent la sommation des courants pour implémenter les produits matrice-vecteur. Malgré des résultats impressionnants avec une efficacité énergétique allant jusqu'à 700 TOPS/W,

cette solution est limitée en précision, usuellement binaire mais parfois jusqu'à 8 bits. Elle souffre également du calcul analogique sensible au bruit et à la variabilité intrinsèque des cellules mémoires. La même observation s'applique pour les mémoires à changement de phase (PCM) et les mémoires magnétiques (MRAM), excepté que l'intégration de l'IMC est compliqué par l'incompatibilité des processus de fabrication. La PCM a le meilleur ratio de valeurs de résistances état 1/état 0 des NVMs et supporte facilement plusieurs niveaux par cellule. D'un autre côté, la MRAM, à cause de son orientation naturelle binaire, est limitée à une cellule à un seul niveau.

Enfin, nous étudions quelques travaux génériques qui s'appliquent à toutes sortes de mémoire utilisant uniquement une propriété commune, en l'occurrence la résistivité des mémoires. D'autres travaux ont essayé de combiner plusieurs mémoires pour obtenir le meilleur des deux mondes. Nous effectuons également un rapide survol des modèles de programmation, des jeux d'instructions et listons les limitations et contraintes auxquelles fait toujours face l'IMC : implémentation matérielle complexe, calcul analogique à faible précision, endurance limitée des NVMs émergentes, applications spécifiques (principalement apprentissage profond) qui manquent de généralité et peu d'études architecturales. C'est pourquoi nous pensons qu'une enveloppe numérique autour de n'importe quelle NVM atteindra les meilleures performances et économisera l'endurance en utilisant un tampon SRAM pour effectuer les opérations de calcul et servir de tampon d'écriture. Cette enveloppe numérique fournit plus de flexibilité et utilise les opérateurs CMOS conventionnels qui sont moins limités en terme de précision numérique. Elle offre également la généralité requise pour les systèmes de calcul et n'est pas sujet au bruit analogique.

Chapitre 3 : Design numérique de la C-SRAM

Comme présenté au chapitre précédent, le calcul analogique n'est pas un outil répandu et générique et est sujet au bruit dû aux variations internes des cellules mémoires et au bruit thermique. L'utilisation de cellules SRAM modifiées pour apporter une isolation entre les lignes de bit et les cellules lorsqu'un calcul active deux lignes simultanément est contre productive. Premièrement, l'opération *précalculée* peut facilement être réalisée avec quelques transistors dans le domaine numérique. L'activation de plusieurs lignes de mot requiert la modification du décodeur de ligne dont l'augmentation de la surface dépasse les peu de portes logiques épargnées. Deuxièmement, les cellules non standard sont dessinées avec les règles logiques, c'est à dire avec des règles moins strictes amenant à une mémoire moins dense. Les coûts de validation du design doivent être pris en compte pour vérifier que la cellule dessinée fonctionne correctement dans un large panel de situation : haute et basse températures, haute et basse tensions, etc. Le calcul dans des NVMs les useraient en quelques jours à cause de leur endurance limitée. De plus, l'accès à la mémoire est peut-être *rapide* (par rapport à la DRAM ou à un disque dur), mais le coût énergétique surpasse encore celui de la SRAM. Ainsi, nous décidons de designer une enveloppe numérique à base de SRAM pour être intégrée autour d'une NVM pour faire d'une pierre, deux coups.

D'un côté, nous bénéficions de la haute densité de la NVM et de sa non volatilité. De l'autre, nous avons l'endurance virtuellement illimitée de la SRAM et sa grande vitesse d'accès pour de meilleures performances de calcul.

Notre enveloppe est dessinée comme une unité de calcul vectorielle avec son propre *pipeline*[1] et son unité de décodage. Elle reçoit les instructions calculées à la volée par le processeur et les exécute. Pour limiter la complexité matérielle, nous ne gérons pas les aléas de *pipeline* tels que lecture après écriture (RAW) ou écriture après écriture (WAW), et les laissons au développeur ou au compilateur. L'unité arithmétique et logique supporte les opérations logiques bit à bit, les décalages et les opérations arithmétiques : addition, soustraction, multiplication et comparaison. Les décalages, addition et soustraction peuvent être de n'importe quelle taille parmi 8, 16 ou 32 bits tandis que la multiplication est uniquement sur 8 bits dans notre étude. Seule l'arithmétique entière est implémentée, mais les calculs en virgule fixe sont facilement réalisables en utilisant les opérations de décalage. Pour minimiser la surface silicium, les opérateurs sont multiplexés de sorte qu'un seul et unique additionneur 64 bits soit instancié et des opérations logiques matérielles sont utilisées pour mimer le comportement des opérateurs plus petits. Un jeu d'instructions est concu en s'inspirant de RISC-V et le placement des bits est choisi pour réduire le nombre de multiplexeurs dans l'étage de décodage. L'enveloppe numérique reçoit les instructions sur les bus d'addresse et de données et le bit d'addresse le plus significatif indique s'il s'agit d'un accès mémoire classique ou d'une opération IMC (périphérique à correspondance mémoire).

Je réalise un travail d'exploration de notre design en utilisant le flot de simulation et conception standard sur de multiples types de SRAM avec différents nombres de port : 1LE (Lecture/Écriture), 1L1E, 1L1LE, 2LE. Nous utilisons le nœud 22 nm de GlobalFoundries avec des mémoires Invecas. Je montre que le surcoût surfacique de notre solution est limité à 5 % pour une mémoire de 256 ko ce qui est acceptable alors que l'augmentation de la puissance est aux alentours de 20 % pour la même mémoire ; cela est plus coûteux mais l'ajout de fonctionnalités, dans ce cas du calcul, a toujours un prix. Je démontre que les mémoires 1L1E ont le meilleur produit énergie-temps, celui-ci étant au moins 2× supérieur par rapport aux mémoires 2LE et 4× meilleur que les mémoires 1LE. En terme d'efficacité énergétique, notre design attteint 2 TOPS/W avec une mémoire 1L1E de 2 ko ce qui est inférieur aux solutions de l'état de l'art. Cependant, ce résultat doit être nuancé car l'état de l'art est souvent mesuré sur une operation MAC où la multiplication est binaire alors que la nôtre est sur 8 bits. En utilisant la décompositions en fonctions logiques binaires de la multiplication 8 bits, nous obtenons un facteur de réduction de l'état de l'art de 400, ce qui nous place au dessus. La densité d'opération est d'environ 100 GOPS/mm^2 ce qui est dans la moyenne de l'état de l'art et, une fois de plus, nous place au dessus en considérant le facteur de correction. Ceci prouve que notre solution offre une diversité en terme de types de mémoires et de tailles, tout en fournissant un design énergétiquement efficient et générique. Notre enveloppe numérique, que nous appelons C-SRAM, peut

[1] ↑Bitoduc ou infoduc en français

de plus être personnalisée en retirant ou ajoutant facilement des opérateurs car notre flot de travail RTL permet un prototypage rapide.

Chapitre 4 : Outils et plateforme de simulation

Dans ce chapitre, je présente les applications au banc d'essai que nous utilisons ainsi que la plateforme que je développe pour explorer diverses architectures utilisant notre enveloppe numérique. Nous visons les applications à gros jeux de données (*big data*) car ce sont ces applications qui sont les plus limitées par le mur mémoire et demandent également le plus de performances. Nous considérons plusieurs bancs d'essai linéaires, nommément *poids de hamming* utilisé en théorie de l'information, *shift-or* utilisé en bioingénierie pour comparaison de motifs de protéines et *AXPY*, un noyau classique de la bibliothèque algèbre linéaire basique (BLAS). Les deux premiers sont limités par la vitesse de calcul tandis que le dernier est limité par la bande passante de la mémoire ou des interfaces processeur/mémoire. Ensuite, nous avons deux bancs d'essai quadratiques qui sont *atax*, utilisé dans les solveurs linéaires et *gesummv*, un cas général du produit matrice-vecteur, que l'on retrouve en traitement d'image et apprentissage profond. Enfin, nous choisissons un banc d'essai cubique, *gemm*, un noyau de multiplication de matrices largement répandu dans de nombreuses applications : AI avec apprentissage profond et réseaux de neurones y compris convolutionnels, traitement d'image, simulation scientifique, etc. Nous sélectionnons aussi une application cas réel, *darknet*, une implémentation de réseaux de neurones utilisant trois des précédents bancs d'essai : *gemm*, *gesummv* et *AXPY*.

Avant d'explorer l'intégration de notre enveloppe numérique dans une architecture complète, je développe une plateforme de simulation adaptée à nos besoins. Premièrement, j'explique pourquoi je n'utilise pas les solutions proposées par l'état de l'art telles que gem5. En effet, gem5 est l'un des simulateurs système les plus utilisé dans le monde de la recherche en architecture des systèmes. Pourtant, il souffre de nombreux défauts. Nous sommes particulièrement intéressés par le sous système mémoire mais il est reconnu comme étant peu précis. Comme nous prototypons de nouvelles architectures, nous ne pouvons pas utiliser les compteurs de performances matériels disponibles dans la plupart des processeurs, puisqu'ils ne peuvent pas compter des évènements encore inexistants et sont imprécis pour les évènements mémoires au delà du cache de niveau 2. Ensuite, je présente les outils de modélisation matérielle que j'utilise pour simuler les différents étages de la hiérarchie mémoire et récupérer des estimations précises des coûts d'accès en temps et en énergie. Je choisi d'utiliser NVSim pour modéliser la NVM mais également la hiérarchie de cache car ce dernier est basé sur Cacti qui est la référence en la matière. Il donne également des résultats moins absurdes que ce dernier. Nous décidons d'utiliser une mémoire à changement de phase pour notre stockage de masse puisque c'est la technologie mémoire émergente la plus mature à ce jour, avec des produits commerciaux déjà disponibles (Intel Optane). Les paramètres technologiques de la PCM sont extraits de l'état de l'art pour être le plus réaliste possible. La DRAM est modélisée en utilisant VAMPIRE, un

outil étalonné à partir de mesures sur des barettes mémoires commerciales. J'ai aussi considéré d'autres outils mais ai choisi celui-ci car c'est le plus récent de tous et sa méthodologie me paraît la plus robuste. Finalement, les paramètres de notre C-SRAM, tirés du chapitre 3, sont étendus pour des mémoires plus grandes et plus larges en utilisant un motif de tuilage calibré à partir d'un travail précédent interne à l'équipe.

Pour développer notre plateforme d'exploration, j'utilise Pin, un outil qui permet d'instrumenter n'importe quelle instruction et en l'occurrence, tous les accès mémoires. Cela inclue toutes les instructions car chaque instruction est chargée depuis la mémoire. En tant qu'outil d'instrumentation, il bénéficie d'une faible pénalité de surcharge, aux alentours de 100× à comparer aux émulateurs et simulateurs complets qui peuvent être un million de fois plus lent que le temps réel d'exécution. Comme nous visons les applications à gros jeux de données, la vitesse de simulation est primordiale dans notre choix. Une interface logicielle est conçue tout en étant compatible avec notre jeu d'instruction présenté dans le chapitre 3. Ce jeu d'instructions est cependant étendu pour supporter un mode d'addressage 64 bits au lieu de 32 bits. Ensuite, je modélise une hiérarchie de cache à 3 niveaux, une DRAM, notre C-SRAM et son comportement associé et au sommet, une mémoire de masse. La première version de notre plateforme modélise une hiérarchie avec une cohérence forcée, où calculer en C-SRAM met à jour la valeur dans les caches et vice-versa. Cependant ce coût de cohérence n'est pas compté dans les statistiques mémoires. De plus, les accès à la mémoire de masse sont comptés incorrectement car le phénomène d'échange (*swapping*) n'est pas monitoré. L'initialisation de la mémoire vient toujours de la SCM ce qui n'est pas réellement le cas avec la pile et le tas. Une seconde version raffine ces aspects pour être plus fidèle à la réalité. Elle gère les appels systèmes qui accèdent au système de fichiers pour les compter dans la facture énergétique et temporelle globale. L'allocation mémoire est correctement suivie de sorte que les copies de données inutiles de la SCM vers la DRAM soient supprimées. Les pages sont désormais suivies pour prendre en compte le phénomène d'échange lorsque la DRAM est pleine, sauvegardée dans la SCM et rechargée quand les données sont demandées à nouveau. Finalement, je passe d'une version à cohérence forcée à une version à cohérence logicielle et ajoute les fonctions adéquates de gestion de la mémoire à notre jeu d'instructions.

Enfin, je réalise une étape de validation pour s'assurer que notre modèle représente fidèlement la réalité. Je compare les nombres d'accès aux caches de notre plateforme avec les compteurs matériels sur une véritable architecture avec les mêmes paramètres. Notre modèle est correct et plus le jeu de données est gros, plus notre modèle est précis (relativement parlant). Quelques évènements apparaissent comme modélisés incorrectement, mais ce sont des évènements rare et la moindre variation amène à de grosses différences de ratios. Néanmoins, ces rares évènements sont cachés par les milliards d'autres correctement modélisés. Je confronte différents simulateurs DRAM et montre que la puissance et les temps d'exécution reportés s'étalent sur 5 ordres de grandeur. Cela prouve à quel point il est difficile de modéliser précisément la DRAM. En utilisant un coût d'accès fixe, nous obtenons les temps les plus précis par rapport aux autres simulateurs en se basant sur le temps d'exécution réel pour référence. Cependant, les estimations d'énergie montrent plus de variations et ne bénéficient

pas d'une véritable référence indiscutable car cela nécessiterait une mise en place lourde. Pour mieux s'assurer des résultats, je trace la puissance DRAM calculée comme l'énergie divisée par le temps et montre que certains simulateurs donnent une puissance supérieure à 1 kW, ce qui est ridicule. Notre plateforme donne principalement des puissances entre 1–10 W et quelques valeurs aberrantes autour de 50 W.

Chapitre 5 : Architectures de calcul en mémoire

Maintenant que nous avons décrit notre enveloppe numérique et une plateforme de simulation, nous pouvons les combiner pour explorer de nouvelles architectures de calcul en mémoire. Comme nous l'avons montré dans l'état de l'art, calculer proche du processeur n'apporte que peu d'améliorations, alors que calculer plus haut dans la hiérarchie mémoire, vers la DRAM ou la NVM, a beaucoup plus de potentiel de gains. En utilisant les bancs d'essai introduits précédemment et la méthodologie définie dans le chapitre 4, nous explorons quatre architectures différentes où le calcul prend place soit dans le tampon de ligne réadapté, soit dans la C-SRAM placée entre la mémoire de rang inférieure (DRAM ou cache de niveau 3) et la mémoire visée (respectivement SCM ou DRAM) en tant que circuit indépendant. L'architecture de référence est basée sur un processeur SIMD 512 bits avec une hiérarchie de cache à 3 niveaux, qui est le standard dans le monde du calcul haute performance. Suit une petite DRAM qui est volontairement sous dimensionnée de sorte que les jeux de données ne puissent tenir dedans, similairement aux applications modernes à gros jeux de données. Au sommet se trouve une SCM de type PCM qui contient toutes les données nécessaires à l'exécution de l'application. L'idée de base du calcul proche de la SCM est de calculer là où se trouve la donnée. En effet, les jeux de données massifs tels que les poids des réseaux de neurones ou les bases de données d'images sont sauvegardés dans cette mémoire de masse. Ainsi, le coût de chargement de ces données depuis la SCM doit être pris en compte et est réduit d'autant qu'on rapproche le calcul de celle-ci.

Dans les deux premières architectures où la C-SRAM est soit un C-RB (*Computing Row Buffer*) dans la SCM ou entre la DRAM et la SCM, nous réduisons drastiquement la consommation énergétique et le temps d'exécution, en moyenne de 17.4× et 12.9× respectivement. Premièrement, nous notons que tous les bancs d'essai présentent une réduction de la consommation et une accélération significative dans notre espace d'exploration. Une C-SRAM dont la taille est comprise entre 128 ko et 512 ko est la taille offrant les meilleurs gains pour tous les bancs d'essai. Deuxièmement, nous distinguons différents types d'améliorations dépendant du comportement d'accès des bancs d'essai. Nous faisons les observations suivantes :

- Les bancs d'essai linéaires bénéficient de vecteurs plus larges car il n'y a pas de dépendance entre les itérations de boucle. Nous atteignons une réduction énergétique d'un facteur 50 et une accélération du temps d'exécution d'un facteur 38 pour *shift-or* et 17 pour *hamming weight*. Pour *AXPY*, nous remarquons que les gains sont bien plus faibles, entre 2× et 3× pour la réduction énergétique et

l'accélération du temps d'exécution. Ceci est dû aux nombreuses écritures dans la SCM ce qui limite les gains maximaux.

- Les bancs d'essai quadratiques montrent des gains moins impressionnants mais toujours intéressants avec une réduction de la consommation de 15× et une accélération autour de 15× pour *atax*, et 25× et 12× respectivement pour *gesummv*. Les accès plus complexes et une réutilisation des données expliquent ces différences avec les bancs d'essai linéaires. De plus, des vecteurs plus grands n'augmentent pas plus les gains au delà d'une certaine taille (1 ko). Cela est dû aux opérations de réduction qui ne peuvent être faites par la C-SRAM. Une variation de l'architecture où cette réduction est réalisée dans la C-SRAM ne montre pas d'amélioration significative (<1 %).

- Les bancs d'essai cubiques comme *gemm* ont les meilleures améliorations, avec une réduction énergétique de 403× et une accélération du temps d'exécution de 272×. Cependant, ces gains ne concernent que le noyau en lui-même et une application réelle telle que *darknet* montre des gains bien moindres, autour de 5× pour la réduction énergétique et 6× pour l'accélération à cause de la loi d'Amdahl.

Enfin, nous montrons que les accès à la SCM, qui déterminent la durée de vie du système, restent constants aussi bien en lecture qu'en écriture par rapport à la référence. Nous concluons donc que notre système n'améliore ni ne dégrade cette durée de vie tout en augmentant sensiblement les performances.

Dans un second cas où le calcul prend place proche de la DRAM, les gains sont bien plus limités, voire même négatifs car les accès DRAM ne sont pas du tout réduits et représentent toujours 80 % de la consommation d'énergie globale. Ainsi, cette solution n'apporte que peu de gain et ne doit pas être considérée pour une application réelle.

Perspectives et travaux futurs

Dans cette thèse, j'ai démontré que calculer au sommet de la hiérarchie mémoire est le meilleur emplacement pour y intégrer du calcul en mémoire, quelle que soit la technologie mémoire considérée pour la SCM. La famille des solutions IMC peut grandement améliorer les rendements énergétiques et temporels pour un large panel d'applications allant de l'algèbre linéaire au traitement des bases de données en passant par l'intelligence artificielle avec les réseaux de neurones et l'apprentissage profond. Néanmoins, plusieurs travaux peuvent encore être menés ou approfondis. Tout d'abord, une démonstration avec un véritable circuit doit confirmer nos résultats, mais cela nécessite un travail énorme aussi bien côté matériel que côté logiciel. Notre jeu d'instructions devra être matériellement implémenté dans un processeur RISC-V, mais la partie la plus compliquée est probablement l'intégration du bus par delà la DRAM pour communiquer avec la C-SRAM. Bien que j'ai démontré des gains énergétiques, je n'ai pas considéré la consommation de puissance instantanée qui est

probablement augmentée à cause des larges vecteurs utilisés. Ainsi, on peut s'attendre à une consommation de puissance avec de larges pics liés aux calculs faits dans la C-SRAM. Cela pourrait poser des problèmes à la mémoire sous-jacente, c'est à dire à la NVM couplée, si notre C-SRAM chauffe trop. Au niveau logiciel, le modèle de programmation doit toujours changer de paradigme vers un modèle centré sur les données. Des instructions spécifiques pour un contrôle plus fin des mouvements de données dans une architecture distribuée peuvent également être considérées. Enfin, les revendications d'augmentation de la sécurité doivent également être vérifiées expérimentalement.

Sur une note final, j'aimerais prévenir des effets indésirables de notre solution. Réduire la consommation d'énergie, au vu du contexte actuel, est bien évidemment une noble cause. Mais elle n'aura probablement pas beaucoup d'effet sur la consommation des centres de données et les appareils en général tant que les budgets énergétiques resteront stables. En effet, toute l'énergie non dépensée sera réinvestie dans d'autres unités de calcul pour toujours augmenter les performances des systèmes. C'est ce qu'on appelle l'effet rebond.