Intégration de technologies de mémoires non volatiles émergentes dans la hiérarchie de caches pour améliorer l'efficacité énergétique

Pierre-Yves Péneau

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Systèmes Automatiques et Microélectroniques (SyAM)

École doctorale Information, Structure, Systèmes (I2S)

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM)

Intégration de technologies de mémoires non volatiles émergentes dans la hiérarchie de caches pour améliorer l'efficacité énergétique

Présentée par Pierre-Yves PÉNEAU

Le 31 Octobre 2018

Sous la direction de Abdoulaye GAMATIÉ, directeur de thèse,
Gilles SASSATELLI, co-directeur de thèse,
et Florent BRUGUIER, co-encadrant de thèse

Devant le jury composé de

Abdoulaye GAMATIÉ, Directeur de Recherche, Centre National de la Recherche Scientifique	Directeur de thèse
Gilles SASSATELLI, Directeur de Recherche, Centre National de la Recherche Scientifique	Co-directeur de thèse
Florent BRUGUIER, Maître de conférences, Université de Montpellier	Co-encadrant de thèse
Jalil BOUKHOBZA, Maître de conférences, Université de Bretagne Occidentale	Rapporteur
Jean-Philippe DIGUET, Directeur de Recherche, Centre National de la Recherche Scientifique	Rapporteur
Daniel CHILLET, Professeur des Universités, Université de Rennes 1	Président du jury

UNIVERSITÉ DE MONTPELLIER

You can only get smarter by playing a smarter opponent.

Jack Green - Revolver (2005)

Résumé

De nos jours, des efforts majeurs pour la conception de systèmes sur puces performants et efficaces énergétiquement sont en cours. Le déclin de la loi de Moore au début du XX^{e} siècle a poussé les concepteurs à augmenter le nombre de cœurs par processeur pour continuer d'améliorer les performances. En conséquence, la surface de silicium occupée par les mémoires caches a augmentée. La finesse de gravure toujours plus petite a également fait augmenter le courant de fuite des transistors CMOS. Ainsi, la consommation énergétique des mémoires occupe une part de plus en plus importante dans la consommation globale des puces. Pour diminuer cette consommation, de nouvelles technologies de mémoires émergent depuis une dizaine d'années : les mémoires non volatiles (NVM). Ces mémoires ont la particularité d'avoir un courant de fuite très faible comparé aux technologies CMOS classiques. De fait, leur utilisation dans une architecture permettrait de diminuer la consommation globale de la hiérarchie de caches. Cependant, ces technologies souffrent de latences d'accès plus élevées que la SRAM, de coûts énergétiques d'accès plus importants et d'une durée de vie limitée. Leur intégration à des systèmes sur puces nécessite de continuer à rechercher des solutions. Cette thèse cherche à évaluer l'impact d'un changement de technologie dans la hiérarchie de caches. Plus spécifiquement, elle s'intéresse au cache de dernier niveau (LLC) et la technologie non volatile considérée est la STT-MRAM. Nos travaux adoptent un point de vue architectural dans lequel une modification de la technologie n'est pas retenue. Nous cherchons alors à intégrer les caractéristiques différentes de la STT-MRAM lors de la conception de la hiérarchie mémoire. Une première étude a permis de mettre en place un cadre d'exploration architectural pour des systèmes contenant des mémoires émergentes. Une seconde étude sur les optimisations architecturales au niveau du LLC a été menée pour identifier quelles sont les opportunités d'intégration de la STT-MRAM. Le but est d'améliorer l'efficacité énergétique tout en atténuant les pénalités d'accès dues aux fortes latences de cette technologie.

Mots-clefs : efficacité énergétique, STT-MRAM, hiérarchie mémoire, caches

A Kansas city shuffle is when everybody looks right, you go left.

Mr. GoodKat - Lucky Number Slevin (2006)

Abstract

Today, intensive efforts to design energy-efficient and high-performance systems-on-chip (SoCs) are underway. Moore's end in the early 20^{th} century pushed designers to increase the number of core per processor to continue to improve the performance. As a result, the silicon area occupied by cache memories has increased. The ever smaller technology node also increased the leakage current of CMOS transistors. Thus, the energy consumption of memories represents an increasingly important part in the overall consumption of chips. To reduce this energy consumption, new memory technologies have emerged over the past decade : non-volatile memories (NVM). These memories have the particularity of having a very low leakage current compared to conventional CMOS technologies. In fact, their use in an architecture would reduce the overall consumption of the cache hierarchy. However, these technologies suffer from higher access latencies than SRAM, higher access energy costs and limited lifetime. Their integration into SoCs requires a continuous research effort. This thesis work aims to evaluate the impact of a change in technology in the cache hierarchy. More specifically, we are interested in the Last-Level Cache (LLC) and we consider the STT-MRAM technology. Our work adopts an architectural point of view in which a modification of the technology is not retained. Then, we try to integrate the different characteristics of the STT-MRAM at architectural level when designing the memory hierarchy. A first study set up an architectural exploration framework for systems containing emerging memories. A second study on architectural optimizations at LLC was conducted to identify opportunities for the integration of STT-MRAM. The goal is to improve energy efficiency while reducing access penalties due to the high latency of this technology.

Keywords : energy efficiency, STT-MRAM, memory hierarchy, caches

A man who doesn't spend time with his family can never be a real man.

Don Vito Corleone - The Godfather (1972)

Remerciements

Après huit années d'études, dont trois d'un doctorat particulièrement intense, je peux enfin écrire ces remerciements. Toutes les personnes mentionnées ci-dessous ont contribué, de près ou de loin, à faire ce que je suis aujourd'hui et je leur en suis reconnaissant.

J'exprime toute ma reconnaissance envers mes directeurs de thèse, Abdoulaye Gamatié et Gilles Sassatelli, pour m'avoir donner l'opportunité de réaliser ce doctorat. Ils ont su me transmettre leur rigueur scientifique et leur force du travail. Leurs précieux conseils tout au long de ces trois ans m'ont remis dans le droit chemin lorsque je m'en éloignai. Je suis également très reconnaissant envers Florent Bruguier, encadrant de ma thèse, qui a toujours pris le temps de suivre mon travail malgré un emploi du temps très difficile. Il a su m'écouter, me conseiller et aussi attiser ma curiosité. Je remercie enfin Jalil Boukhobza, Jean-Philippe Diguet et Daniel Chillet, pour avoir accepté d'être membre de mon jury de thèse et d'avoir pris le temps de lire et d'évaluer mes travaux.

Je souhaiterai ensuite remercier les membres du corps académique qui m'ont amené jusqu'ici. À Monsieur Sébastien Faucou (Monseigneur pour les intimes), pour m'avoir donner le goût de la programmation en C, de m'avoir transmis ses connaissances et surtout de m'avoir parlé de l'université Pierre et Marie Curie. À Monsieur Florian Richoux et Monsieur Frédéric Goualard pour la confiance qu'ils m'ont apporté pendant ma licence, ainsi que pour leur lettres de recommandation à l'université Pierre et Marie Curie. À Gaël Thomas pour m'avoir appris en Master le fonctionnement des noyaux monolithiques, une de mes envies les plus chères depuis l'IUT. À Alain Greiner, pour m'avoir mis sur la voie de l'architecture des ordinateurs. À Franck Wajsbürt pour m'avoir fait travailler sur un système d'exploitation massivement multicœurs lors de mon stage de fin d'études.

Durant ces trois années au LIRMM, j'ai été entouré par des personnes pour qui je souhaite écrire ces quelques mots. À Caroline Lebrun, secrétaire du département microélectronique, pour sa bonne humeur permanente, pour avoir du gérer pendant trois ans

toute la partie administrative de ma thèse, et pour avoir supporté mes blagues pas drôles. À Cécile Lukasik, Directrice des Ressources Humaines, pour sa gestion des conflits professionnels et ses encouragements. Dans l'ordre alphabétique, je souhaiterai remercier Alejandro, Anastasiia, Artem, Charles, David, Jérémie, Louisa et Thibault. Des collègues, des ami(e)s, avec qui j'ai partagé des moments forts, inoubliables, des discussions professionnelles sérieuses comme pas sérieuses, des apéros, des restaurants, des matchs de football, des fous-rires… J'ai énormément appris à vos côtés, merci.

J'exprime toute mon amitié aux copains de longue date qui, malgré notre éloignement géographique persistant, ont su rester les mêmes et maintenir intact ce lien entre nous : Armel, Guillaume, Gwendal, Jérome et Kévin. Merci à vous. Merci également à Steve qui me suit de loin depuis toutes ces années, mais qui a pris le temps de venir boire des bières avec moi lorsqu'il le fallait. Enfin, je ne peux m'empêcher de penser à Alexandre, à qui j'ai fait cette promesse de doctorat et qui n'est plus là pour la voir réalisée. Peace.

Parce que la vie n'est qu'une mélodie sur laquelle glisse les notes de nos existences, j'aimerai remercier les artistes suivants. Leur musique a su m'aider dans les moments de faiblesse, me motiver dans les moments de flemme, et me mettre en transe dans les moments d'écriture. Dans l'ordre alphabétique : Angerfist, Booba, Cypress Hill, Damien Saez, Hans Zimmer, Howard Shore, IAM, James Newton Howard, Kery James, Keny Arkana, Kid Cudi, Linkin Park, Nino Rota, Ramin Djawadi, Rockin' Squat, Stupeflip, et tellement d'autres…

Enfin, il convient de rendre un dernier hommage aux personnes sans qui tout cela ne serait pas arrivé. À mon parrain, qui a forgé en moi cette envie de réussir. À mon frère, Erwann, et Gwennaïg, ma sœur, pour avoir toujours été là et avoir su rester proche malgré la distance qui nous sépare depuis toutes ces années. Je vous aime. Papa, Maman, merci d'avoir tout jeune éveillé ma curiosité et de m'avoir montré la voie vers la Science. Merci pour votre soutien indéfectible et pour votre aide tout au long de mon parcours. Sachez que je n'en tire aucune fierté car tout le mérite est pour vous. Cette thèse, c'est la votre. Ma réussite est votre réussite.

Pierre-Yves Péneau, à Monptellier le 31 Octobre 2018.

Table des matières

Liste des abréviations

API	Application Programming Interface
CMOS	Complementary Metal Oxide Semiconductor
CO	Compteur Ordinal
CPU	Central Processing Unit
CRC	Cache Replacement Championship
CSV	Comma-Separated Values
DIMM	Dual In-line Memory Module
DRRIP	Dynamic Re-Reference Interval Prediction
DVFS	Dynamic Voltage Frequency Scaling
EDP	Energy-Delay Product
FL	Free Layer
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
ITRS	International Technology Roadmap for Semiconductors
KIPS	Kilo-instructions par seconde
KIPS	Kilo-instructions par seconde
LPDDR	Low Power Double Data Rate
LRU	Least-Recently Used
LSB	Least Significant Bit
MAGPIE	Multicore Architecture enerGy and Performance evaluatIon Environment
MSB	Most Significant Bit

MSHR	Miss Status Holding Registers
MTJ	Magnetic Tunnel Junction
NVM	Non-Volatile Memory
pWCET	Partial Worst-Case Execution Time
RL	Reference Layer
ROI	Region of Intereset
RRPV	Re-Reference Interval Prediction
RTL	Register-Transfer Level
SHiP	Signature-based Hit Predictor
SPM	Scratch-Pad Memory
SRAM	Static Random Access Memory
SRRIP	Static Re-Reference Interval Prediction
SSD	Solid-State Drive
STT-MRAM	Spin-Transfer Torque Magnetic RAM
TLB	Translation Lookaside Buffer
XML	Extensible Markup Language

Liste des figures

Liste des tableaux

This is your last chance. After this, there is no turning back. You take the blue pill - the story ends, you wake up in your bed and believe whatever you want to believe. You take the red pill - you stay in Wonderland and I show you how deep the rabbit-hole goes.

MORPHEUS - THE MATRIX (1999)

Chapitre 1

Introduction

Du simple outil de recherche au milieu du $20^{ème}$ siècle, l'informatique est devenue en l'espace d'un demi-siècle indispensable. Aujourd'hui, il existe une large variété d'usages : calcul haute performance (HPC), internet des objets (IoT), calcul dans les nuages (cloud) etc. Cette diversité change la façon dont les architectes abordent la conception des processeurs. Dans certains cas, la simplicité est recherchée pour avoir un environnement contrôlé, comme dans le domaine de l'avionique. Dans d'autres cas, on recherche la rapidité de calcul, ce qui implique des processeurs optimisés et plus complexes. Néanmoins, quelque soit l'usage des processeurs, tous aujourd'hui ont un point en commun dans leur conception : diminuer le plus possible la consommation énergétique. La question de *l'efficacité énergétique* est un des grands enjeux de nos jours dans le domaine de l'informatique. A titre d'exemple, les États-Unis ont estimé en 2015 qu'une machine de calcul d'une puissance d'un exaflops (10^{18} opérations flottantes par seconde) basée sur l'infrastructure chinoise Tianhe-2 [107] consommerait autant d'énergie que les foyers de la ville de San Francisco [37]. Une telle puissance de calcul n'est économiquement pas rentable et la communauté scientifique doit apporter une réponse à ce challenge financier.

1.1 Consommation énergétique : nature et répartition

Depuis une quinzaine d'années, l'International Technology Roadmap for Semiconductors (ITRS) a identifié l'augmentation de la consommation énergétique comme un problème majeur [53]. L'analyse de cette consommation énergétique montre qu'il existe deux types de consommation : dynamique et statique. La consommation dynamique est le besoin en courant requis par un transistor pour le faire changer d'état ($0 \rightarrow 1$ et $1 \rightarrow 0$). La consommation dynamique du circuit est la somme de toute l'énergie nécessaire à ces changements d'état sur tous les transistors. Cette énergie est nécessaire uniquement lorsqu'il y a une tâche à effectuer. A l'inverse, l'énergie statique, ou courant de fuite, est un besoin énergétique indépendant de toute activité. La miniaturisation à l'extrême des composants entraîne une fuite d'électrons permanente dans les transistors. Ces derniers doivent être remplacés en continu pour conserver l'état du transistor, générant une consommation électrique. Plus un composant contient de transistors, et donc occupe une grande surface de silicium, et plus son courant de fuite est important.

C'est de cette énergie statique que provient une part significative de la consommation énergétique des circuits [53]. A titre d'exemple, Carroll et al. [20] ont montré que dans certains cas, l'énergie statique d'un smartphone lorsqu'il n'est pas en veille est d'au moins 50%, et ce chiffre augmente considérablement si l'on ajoute le système d'affichage. La diminution de la finesse de gravure entraîne également une augmentation de la consommation statique. Les constructeurs intègrent donc des mécanismes d'économie d'énergie et les activent le plus souvent possible, sacrifiant les performances pour une durée de batterie plus élevée.

Nos travaux se placent au niveau d'un nœud de calcul illustré par la Figure 1.1. Ce système comporte un ou plusieurs cœurs, une hiérarchie de caches, un système d'interconnexion, une mémoire principale et un système de stockage.

Sur un nœud de calcul, la consommation énergétique provient majoritairement des cœurs et de la hiérarchie mémoire [22]. En effet, la majeure partie de la surface est occupée par ces composants, et la place accordée à la mémoire est de plus en plus importante. Cette tendance est illustrée par la Figure 1.2, qui donne la répartition du silicium entre la partie logique (processeur) et mémoire (caches).

La Figure 1.3 représente une projection de l'ITRS sur l'évolution de l'énergie statique et dynamique pour la partie logique et la partie mémoire du processeur. Pour la partie logique, on observe une augmentation progressive de l'énergie dynamique. La partie statique augmente également mais de manière moins importante. A l'inverse, la consommation statique des mémoires augmente au fur et à mesure des années, à cause de l'augmentation de la surface occupée. Entre 20% et 30% de la consommation totale provient de ce

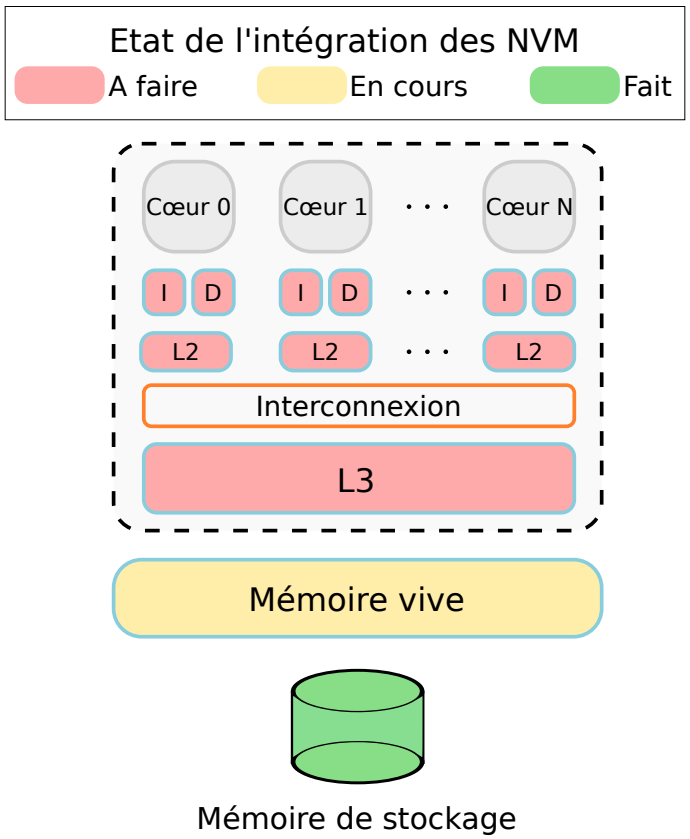


FIGURE 1.1 – Schéma d'un nœud de calcul comprenant des cœurs, une hiérarchie de cache à trois niveaux, une mémoire vive principale et une mémoire de stockage.

courant de fuite.

Il existe des techniques de réduction de consommation que l'on trouve dans les processeurs comme le *clock-gating* ou le *power-gating*. Elles consistent à éteindre tout ou une partie du cœur de calcul lorsque celui-ci n'est pas utilisé. Ces méthodes sont particulièrement efficaces et permettent à la consommation énergétique du cœur d'être dominée par l'énergie dynamique. Ce comportement est observable sur la Figure 1.3, ou la partie logique dynamique est plus importante que la partie logique statique. Le cœur consomme donc l'énergie nécessaire pour ses opérations de calcul.

L'extinction par intermittence de certains circuits dans un cœur est possible car en dehors des bancs de registres, le cœur n'a pas besoin de sauvegarder des données. D'autres parties du processeur sont responsables de cette tâche, comme la TLB ou les caches. Ces techniques de diminution de la consommation énergétique ne peuvent s'appliquer sur des composants à mémoire volatile. En effet, la coupure de l'alimentation entraîne une perte d'information, et donc une inconsistance dans les données.

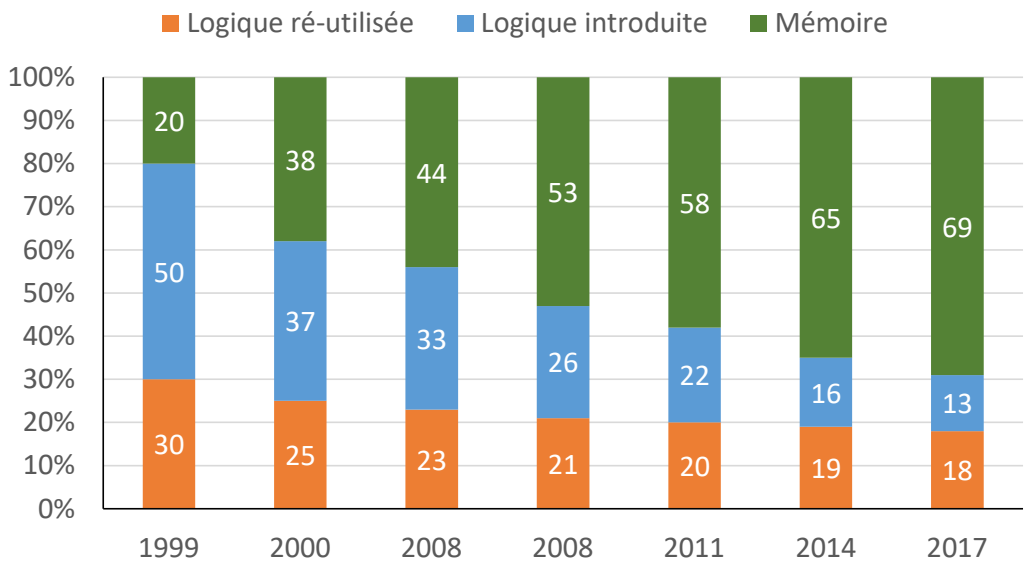


FIGURE 1.2 – Répartition de la surface de silicium entre les parties logiques et mémoires. Source : Semico Corp. [94]

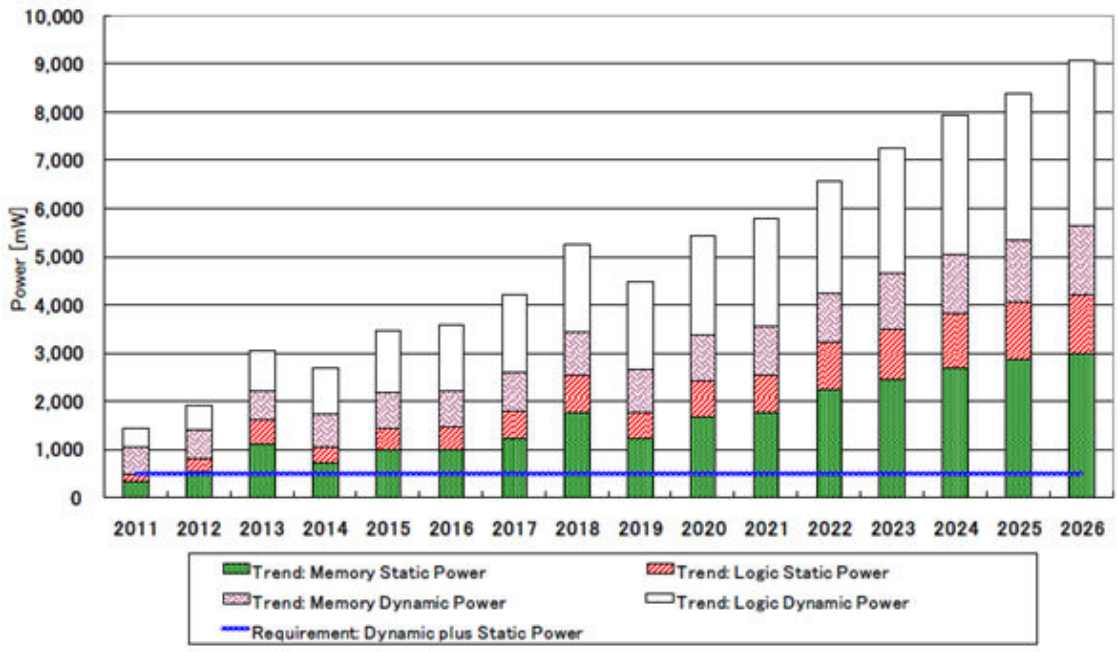


FIGURE 1.3 – Répartition de la consommation énergétique entre les parties logiques et mémoires [44]

La nature de la consommation énergétique des mémoires caches est à l'exact opposé des cœurs : l'énergie statique est plus importante. Sur la Figure 1.3, on observe que les mémoires sont dominées par l'énergie statique. Cette situation s'explique notamment par

le fait que l'activité des caches est moins importante que celle du cœur. Plus un cache est à un niveau élevé, par exemple le cache de niveau 3, moins il est sollicité. C'est également les niveaux de caches élevés qui sont les plus grands en termes de capacité et donc en surface de silicium occupée sur une puce. Or, l'énergie statique est proportionnelle à la surface. Les caches de dernier niveaux sont donc pour beaucoup responsables de l'énergie statique totale des mémoires.

La domination de l'énergie des mémoires cache par le courant de fuite est dû à l'utilisation de la technologie SRAM , et de son besoin d'alimentation en continu.

1.2 Problématique

Beaucoup d'approches existent pour diminuer la consommation énergétique des puces. Au niveau logiciel, le système d'exploitation peut moduler la fréquence de fonctionnement du processeur via le DVFS. Les puces *Single ISA Heterogeneous Architecture* et son implémentation commerciale ARM big.LITTLE^{TM} sont de nos jours très répandues notamment dans les smartphones Samsung et Apple. Ces puces sont composées de deux types de cœurs qui peuvent être puissants et énergivores ou de faible puissance mais très efficaces énergétiquement. De nouvelles technologies de mémoires émergentes et non volatiles sont également étudiées depuis une dizaine d'années.

Ces technologies apportent un changement de paradigme dans la façon d'utiliser les caches. Grâce à leur non volatilité, elles peuvent stocker une information puis couper l'alimentation de la cellule mémoire concernée. La puissance statique est ainsi considérablement diminuée comparé aux technologies CMOS classiques.

La Figure 1.1 illustre l'état de l'intégration des mémoires non volatiles (NVM) dans les architectures de nos jours. Celles-ci sont déjà présentes dans les disques durs de type SSD depuis plusieurs années et en 2019, Intel devrait commercialiser sa technologie 3D XPoint au niveau de la mémoire centrale [13]. Le prochain niveau d'intégration est la hiérarchie de caches.

En général, les NVM souffrent de plus longues latences ainsi que de coûts énergétiques d'accès supérieurs à la SRAM [12]. Si l'intégration native de NVM dans une architecture diminue la consommation énergétique, les pénalités dues aux fortes latences sont trop importantes pour être acceptables. L'utilisation de ces technologies nécessite donc en amont une phase de réflexion et d'exploration avant leur intégration.

Les travaux existant tendent vers une modification physique de la cellule de mémoire, ou bien des optimisations au niveau du circuit. Ces approches nécessitent cependant une modification technologique à un grain très fin et sont limitées par les informations dispo-

nibles sur les NVM. On trouve également des approches hybrides mêlant les technologies SRAM et NVM dans un même niveau de cache. Ces travaux se placent à un niveau supérieur que précédemment, mais impliquent un mélange de technologies mémoire au sein d'un même composant. Bien que certaines NVM soient compatibles CMOS, les circuits périphériques nécessaires au fonctionnement des NVM et des SRAM ne sont pas identiques, ce qui implique une plus grande complexité de conception. Enfin, des approches architecturales existent pour la diminution des écritures. Ici, on se place à un niveau supérieur où l'on ne mixe pas les technologies mais où l'on modifie l'architecture de la hiérarchie mémoire pour prendre en compte les caractéristiques différentes des NVM.

La problématique principale de cette thèse est d'**étudier l'impact d'un changement de technologie et de trouver des techniques pour tirer profit de ces mémoires émergentes lors de leur intégration dans une mémoire cache**. Ainsi, cette thèse essaye de répondre à la question suivante :

Q1 : Comment évaluer l'impact d'un changement de technologie dans la hiérarchie de mémoires caches ?

Pour répondre à cette question, nous verrons quels sont les différents niveaux d'abstraction qui existent pour simuler des architectures, et nous justifierons notre choix parmi toutes les possibilités. Nous définirons un schéma de principe du déroulement de nos explorations, et nous présenterons les outils utilisés.

Pour nos travaux, nous nous focalisons sur la technologie Spin-Transfer Torque Magnetic RAM, ou STT-MRAM . Sa maturité dans le monde de l'industrie permet d'envisager à moyen terme son utilisation dans une hiérarchie de caches. Ce choix sera justifié plus longuement dans le chapitre 3. Néanmoins, les propositions développées dans ce manuscrit ne sont pas spécifiques à la STT-MRAM.

On cherche à identifier des moyens nous permettant de profiter des bénéfices de la technologie STT-MRAM sans devoir effectuer des modifications à bas niveau. Nous considérons les approches s'intéressant à la cellule mémoire ou au circuit comme trop complexes à mettre en place et trop aléatoires dans un milieu où la technologie change rapidement. Nous nous plaçons à un niveau architectural avec les hypothèses suivantes [72] :

— la STT-MRAM a des temps d'accès plus lents que les SRAM en lecture et en écriture

— les coûts énergétiques d'accès pour la lecture et l'écriture sont plus élevés que pour la SRAM

— il existe une asymétrie en termes de latence et d'énergie entre la lecture et l'écriture pour une mémoire STT-MRAM

Ainsi, une autre question à laquelle tente de répondre cette thèse est la suivante :

Q2 : Comment tirer profit de la technologie de mémoire non volatile STT-MRAM au niveau de la hiérarchie de caches sans en modifier la technologie ?

Nous ne cherchons pas à modifier les caractéristiques de la STT-MRAM mais à les prendre en compte lors de la conception d'une hiérarchie de caches en mettant en place des optimisations architecturales intégrant ces caractéristiques. Nous verrons quelles sont les possibilités offertes par cette nouvelle technologie en matière d'architecture et on analysera les leviers permettant d'atténuer ses aspects négatifs. Une étude des transactions entre les composants ainsi qu'une meilleure gestion de la mémoire permettent par exemple de diminuer le nombre d'écritures.

1.3 Objectifs

La technologie STT-MRAM semble idéale pour résoudre le problème de la consommation énergétique mais son utilisation n'est pas gratuite. Certaines de ses caractéristiques posent la question de l'intégration de cette technologie dans un système mémoire qui nécessite d'être le plus rapide possible pour ne pas affecter les performances.

L'objectif de cette thèse est d'étudier l'impact de l'intégration de la STT-MRAM dans une hiérarchie de mémoire cache. La métrique principale d'évaluation est l'efficacité énergétique, que nous caractérisons par l'Energy-Delay Product (EDP). Néanmoins, on s'attardera également sur le temps d'exécution et la diminution de la consommation énergétique.

Pour mener nos travaux, il nous faut analyser les opportunités d'action au niveau de la mémoire. La gestion des caches est un domaine complexe qu'il faut appréhender. Il est également nécessaire de construire des modèles d'évaluation pour nos explorations. Ces derniers sont requis pour *i*) la simulation d'architectures complexes *ii*) les technologies de mémoires émergentes et *iii*) l'évaluation de la consommation énergétique de plateformes intégrant différentes technologies. Enfin, nous devons mettre en application nos propositions et les valider avec ces modèles.

1.4 Contributions

Les principales contributions de cette thèse sont les suivantes :

— **La définition d'un cadre générique d'évaluation d'architectures.** Nous proposons d'évaluer les différents niveaux d'abstraction de simulation (RTL, FPGA, etc.) puis, une fois le niveau choisi, nous définissons le processus d'évaluation en découpant les besoins et en proposant un outil pour chaque besoin.

— **Deux implémentations de ce modèle générique.** Nous présentons deux flots d'exploration automatisés ou semi-automatisés pour évaluer rapidement à l'échelle d'un système l'impact d'un changement de technologie dans la hiérarchie de caches. A notre connaissance, il n'existe aucune solution automatique dans la littérature pour le niveau de simulation que nous avons choisi

— **La proposition de deux optimisations architecturales visant la STT-MRAM.** Nous analysons les possibilités d'optimisations architecturales rendues possibles par un changement de technologie. Les caractéristiques différentes de la STT-MRAM comme sa densité ou sa latence d'écriture sont des leviers que nous allons exploiter. On s'intéressera à l'évolution de l'architecture interne de la mémoire cache lorsque l'on souhaite agrandir sa capacité de stockage. On regardera aussi les différents types d'écritures qui existent au sein de la hiérarchie mémoire et comment les réduire.

— **L'évaluation de nos propositions via notre cadre d'exploration.** Nous proposons de valider nos propositions en utilisant les deux implémentations du flot d'exploration générique cité ci-dessus.

1.5 Organisation de la thèse

A la suite de cette introduction, le chapitre 2 présente le fonctionnement de la hiérarchie de mémoires caches ainsi qu'une étude sur la gestion de la mémoire via les politiques de remplacement.

Le chapitre 3 introduit les mémoires non volatiles et leur caractéristiques. On s'intéresse en particulier à la STT-MRAM, et on détaille les approches adoptées dans les précédents travaux pour pallier aux problèmes de cette technologie. Pour ces deux chapitres, on veille également à l'écosystème logiciel permettant de simuler des caches et des technologies non volatiles.

Le chapitre 4 présente un flot générique d'exploration nécessaire à nos évaluations. Deux implémentations de ce flot sont ensuite développées.

Le chapitre 5 propose une analyse des opportunités d'optimisations au niveau de la hiérarchie de caches. On s'intéresse à la densité supérieure de la STT-MRAM face à la SRAM ainsi qu'aux différents types d'écritures existant au sein de la hiérarchie mémoire. Ces analyses sont basées sur l'utilisation des flots développés au chapitre 4.

Le chapitre 6 présente les résultats expérimentaux obtenus par les optimisations architecturales proposées au chapitre 5. On s'attarde également sur les optimisations de conception de la mémoire cache sur lesquelles appliquer nos propositions.

Enfin, le chapitre 7 conclut nos travaux et présente quelques perspectives pour les travaux futurs.

You see, in this world there's two kinds of people my friend: those with loaded guns, and those who dig. You dig.

Blondie - The Good, the Bad and the Ugly (1966)

Chapitre 2

Fonctionnement de la hiérarchie mémoire d'un processeur

Ce chapitre introduit le fonctionnement de la hiérarchie mémoire. L'organisation des caches ainsi que les échanges entres ces composants sont présentés. On s'intéresse notamment aux mécanismes matériels générant des écritures dans les caches. Après avoir introduit les enjeux des politiques de gestion de la mémoire, ce chapitre développera une étude sur la performance des politiques de remplacement. On évaluera différentes stratégies de

remplacement présentées dans la littérature afin de sélectionner la plus performante pour la suite de nos travaux.

2.1 Principes des mémoires caches

2.1.1 Structure des caches et échange de données

Un cache est un composant mémoire intégré dans le processeur. De quelques kilo-octets à plusieurs méga-octets, il s'intercale entre la mémoire principale et le cœur de calcul. Un cache contient un sous-ensemble des données de la mémoire principale. Il a pour but d'accélérer l'exécution du processeur en lui fournissant les données nécessaires aux calculs. L'intérêt d'un cache est sa rapidité d'accès, de l'ordre de quelques cycles d'horloge.

La Figure 2.1 illustre l'organisation interne d'un cache. Un cache se décompose en *set*, eux-mêmes contenant des *lignes*, ou *way*. Chaque ligne est ensuite découpée en *blocs*. Le nombre de lignes sur un set défini *l'associativité* d'un cache.

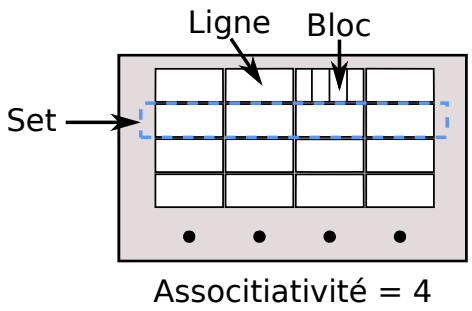


FIGURE 2.1 – Composition d'un cache

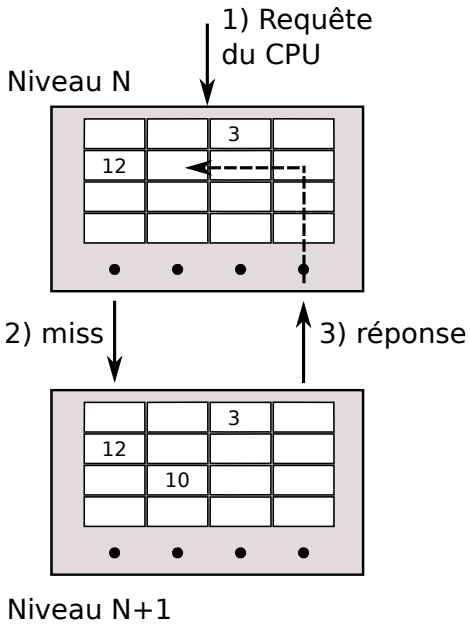


FIGURE 2.2 – Illustration des phénomènes de *hit* and *miss*

Lorsqu'un cache reçoit une requête et que la donnée demandée est contenue dans ce cache, cela génère un *hit*. A l'inverse, lorsque la donnée demandée n'est pas dans le cache, on parle de *miss*. La donnée sera alors cherchée dans le niveau de mémoire supérieur. Cet exemple est illustré par la Figure 2.2. Si le cœur demande la donnée 12 au cache de niveau N, cela génère un *hit*. S'il demande la donnée 10, c'est un *miss*. La requête est alors

transférée vers le cache de niveau $N+1$, qui renvoie la donnée au niveau N. La donnée 10 est alors contenue à la fois dans le niveau N et le niveau $N+1$.

2.1.2 Localité spatiale et temporelle

L'efficacité des caches repose sur les principes de localité spatiale et localité temporelle. [1] La localité temporelle indique qu'une donnée accédée récemment dans le cache a une forte probabilité d'être accédée de nouveau dans un futur « proche ». La localité spatiale indique que les adresses ont une forte probabilité d'être accédées de manière consécutive. Ces principes sont décrit sur les Figures 2.3a et 2.3b.

(a) Localité temporelle

(b) Localité spatiale

FIGURE 2.3 – Principes de localité

L'algorithme 2.1 ci-dessous illustre ces principes de localité par l'addition des éléments d'un tableau. Dans cet exemple, la variable `sum` est accédée à chaque itération de la boucle et respecte le principe de localité temporelle. Les éléments du tableau sont eux accédés consécutivement et respecte la localité spatiale. Enfin, l'instruction d'addition des variables `i` et `sum` est également accédée à chaque itération.

```
sum = 0;
for (unsigned int i = 0; i < n; i++) {
    sum += a[i];
}
```

Algorithme 2.1 – Somme des éléments d'un tableau

2.1.3 Niveaux de cache

Plusieurs niveaux de cache sont généralement utilisés entre un cœur et la mémoire. Le niveau de cache L1 le plus proche du CPU contient quelques dizaines de kilo-octets de données, ce qui est suffisant lorsqu'une application respecte les deux principes de localité.

1. Ces principes sont respectés pour la majorité des applications mais il existe des exceptions.

Les niveaux supérieurs sont plus importants en termes de capacité. Les derniers processeurs Intel-i7 de 8^{e} génération contiennent par exemple un cache de niveau 3 de 12Mo. [2] La combinaison de plusieurs niveaux de cache permet de profiter de ces principes sur un plus large volume de données. Ainsi, la hiérarchie de caches minimise les accès externes à la mémoire principale qui sont coûteux en temps et en énergie.

2.1.4 Mécanismes matériels générant des écritures

Mode d'écriture

Lorsque le CPU écrit dans son cache L1, il existe deux modes pour propager l'information à la hiérarchie mémoire : *Write-Through* et *Write-Back*.

Avec le mode *Write-Through*, toute écriture dans le cache L1 est parallèlement propagée à l'ensemble de la hiérarchie mémoire, y compris la mémoire principale. De cette façon, la hiérarchie mémoire contient en permanence les valeurs les plus à jour pour toutes les données. Ce mode génère cependant un nombre important de transactions à gérer pour les systèmes d'interconnexion, ainsi que pour les caches. Ainsi, il n'est que rarement utilisé dans les architecture multicœurs classiques. A l'inverse, le mode *Write-Back* délaie l'écriture dans les niveaux supérieurs de la mémoire au plus tard. Typiquement, cela arrive lorsqu'une ligne de la mémoire cache L1 est pleine et qu'une donnée doit alors être retirée pour en stocker une nouvelle. Dans cette situation, la ligne qui est retirée est écrite dans le cache L2. Une autre possibilité est lorsque le protocole de cohérence des caches est utilisé (détaillé dans la section suivante).

Le mode *Write-Back* est le mode le plus utilisé dans les processeurs aujourd'hui. Pour nos travaux, nous considérons une hiérarchie de caches utilisant le mode *Write-Back*. L'impact de ces écritures sera étudié au chapitre 5.

Cohérence des caches

Dans la majorité des architectures multicœurs utilisant un mode d'écriture *Write-Back*, les données contenues dans les caches sont cohérentes. Cela signifie que si un cœur modifie une donnée dans son cache L1 privé et qu'un second cœur souhaite y accéder, il existe un mécanisme implémenté entièrement en matériel permettant de s'assurer que la valeur lue par le second cœur soit la valeur modifiée par le premier cœur. Ce mécanisme peut être implémenté de différentes manières, via du *snooping* [66] ou l'utilisation d'un *directory* [1]. La Figure 2.4 illustre la cohérence de cache avec un *directory*. Dans cet exemple, le cœur 0 lit une donnée, 10, puis la modifie. Ensuite, le cœur 1 souhaite accéder à la donnée

2. https://ark.intel.com/products/148263/Intel-Core-i7-8086K-Processor-12M-Cache-up-to-5_00-GHz

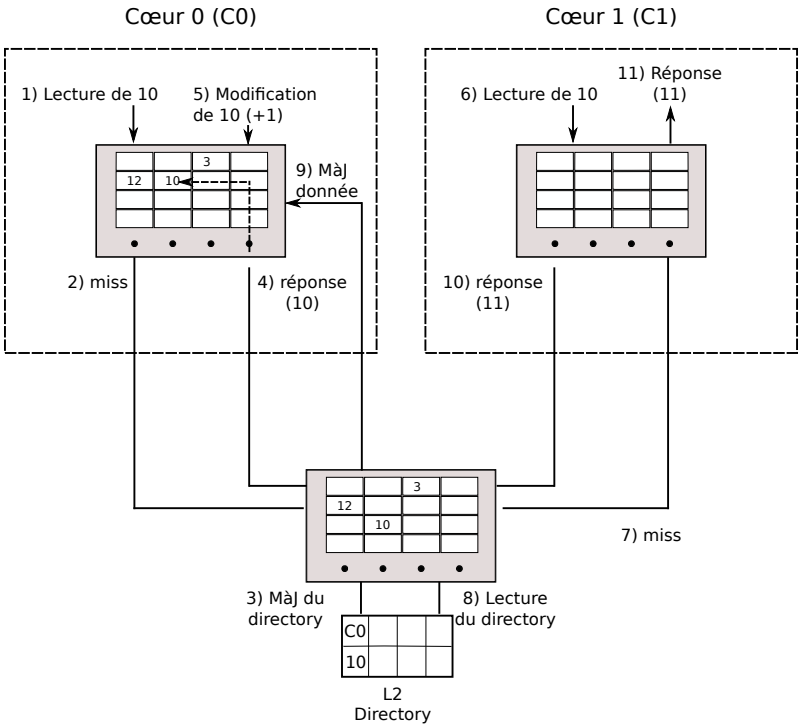


FIGURE 2.4 – Illustration du protocole de cohérence de caches avec *directory*

10. Le cache L2 consulte alors le *directory* et remarque que le cœur 0 possède une copie de cette donnée. Elle est alors mise à jour dans le cache L2, puis celui-ci répond au cœur 1 avec la donnée à jour, 11 dans cet exemple.

Quelque soit l'implémentation considérée, la cohérence des caches est un mécanisme automatique géré par le matériel qui ajoute des écritures supplémentaires dans les caches. Ces écritures ne sont pas des écritures contenues dans le code des applications exécutées. Ces travaux de thèse ne proposent pas de s'attaquer à ce type de transactions, qui impactent majoritairement les caches de premiers niveaux. On verra dans la suite de ce manuscrit que notre intégration de NVM se situe à un niveau de cache plus élevé. Néanmoins, de précédents travaux existent quant à l'utilisation du protocole de cohérence de caches pour une gestion différente de la mémoire en présence de NVM [61, 110].

Pré-chargement des données

Aussi appelé *prefetching*, ce mécanisme a pour but d'amener dans le cache des données qui seront très probablement utilisées dans un futur « proche ».

Nous avons vu en section 2.1.2 que la majorité des applications respectent les principes de localité spatiale et temporelle. C'est sur le principe de localité spatiale que se

base le système de *prefetching*. [3] Pour cela, les caches sont équipés d'un mécanisme appelé *prefetcher* qui est chargé de surveiller les adresses des données qui sont accédées. Si l'on reprend l'exemple de l'algorithme 2.1, les variables du tableau sont accédées consécutivement. Ce comportement sera détecté par le *prefetcher* de la mémoire cache L1. Dans ce cas, des transactions de lecture seront envoyées automatiquement au niveau de la mémoire cache L2 pour pré-charger les futures données du tableau qui seront accédées.

Par l'ajout de transactions de lecture vers un niveau de mémoire supérieur, le système de *prefetching* ajoute des écritures dans les caches. Cependant, il permet aussi d'augmenter la probabilité de *hit* sur les données, diminuant le temps passé par le processeur à attendre des données depuis la mémoire.

Gestion de la mémoire : remplacement, promotion et insertion

La gestion de la mémoire par un cache est défini par trois politiques : le remplacement, la promotion et l'insertion.

La politique de remplacement est utilisée lorsqu'une donnée doit être écrite dans une ligne de cache qui est pleine. Il faut alors sélectionner une donnée à remplacer. On appelle cela un *évincement*. Le choix du remplacement est fait par une heuristique construite au fur et à mesure de l'exécution d'une application. Typiquement, cette heuristique associe à chaque bloc d'une ligne de cache une position et évince toujours le bloc avec la position la plus élevée. La politique de promotion définit comment l'heuristique de remplacement modifie la position d'un bloc lorsque celui-ci est accédé. Enfin, la politique d'insertion définit quelle sera la position d'un bloc que l'on est en train d'insérer dans une ligne de cache. Ces mécanismes sont illustrés sur la Figure 2.5.

Ces trois politiques ont un impact direct sur le nombre d'écriture qui ont lieu sur le cache. En fonction des positions affectées à l'insertion et à la promotion, un bloc sera plus ou moins prioritaire pour être remplacé, et donc restera plus ou moins longtemps sur une ligne de cache. Si les blocs ne restent pas suffisamment longtemps en mémoire, ils seront régulièrement ramenés en cache et cela augmentera les écritures. Dans la section suivante, nous proposons d'évaluer l'impact en performance de différentes politiques de remplacement pour un cache de dernier niveau afin de sélectionner la plus performante pour la suite de nos travaux.

3. Il existe d'autres systèmes de *prefetching* avancés basés par exemple sur les compteurs ordinaux. Ces derniers ne sont pas évoqués ici dans un soucis de clarté.

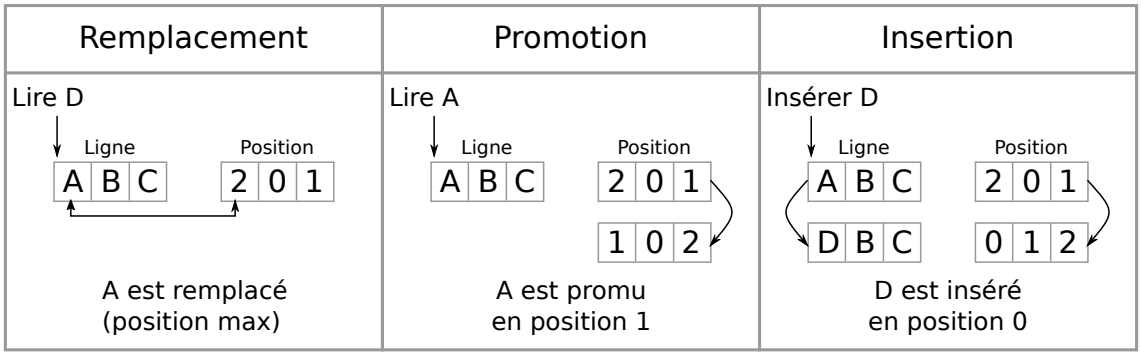


FIGURE 2.5 – Illustration des politiques de remplacement, de promotion et d'insertion

2.2 Politiques de remplacement de caches

Cette section propose une étude sur la gestion de la mémoire par les stratégies de remplacement de cache [83]. Elle sert notamment d'introduction à ce sujet qui sera abordé plus largement dans la suite de ce manuscrit.

On cherche ici à évaluer l'impact des politiques de remplacement sur le cache de dernier niveau en termes de performance. Dans un premier temps, nous expliquons quel est l'objectif de ces politiques de remplacement. Ensuite, nous détaillons leur fonctionnement en explicitant pour chacune leur algorithme et les modifications architecturales qui sont nécessaires. Enfin, nous procédons à une analyse de ces politiques afin de déterminer la plus performante.

2.2.1 Rôle des politiques de remplacement

Le cache de dernier niveau (LLC) est un composant critique de la hiérarchie mémoire qui a un impact direct sur les performances. Lorsqu'une donnée demandée par le processeur n'est pas dans le LLC, une transaction vers la mémoire principale est initialisée. Cette transaction est coûteuse en matière de temps et d'énergie. La réduction du nombre de *miss* au niveau du LLC permet donc de réduire le nombre de transaction externes, diminuant les pénalités de temps et d'énergie.

La politique de remplacement la plus répandue est la politique LRU, ou *Least-Recently Used*. Cette stratégie est utilisée depuis des années car elle est simple et peu coûteuse à mettre en place au niveau de sa complexité matérielle. Cependant, il a été montré que la LRU n'est pas la plus efficace en matière de performance [49, 57, 89], et est même sous-optimale comparé à la meilleure stratégie théorique [6].

Le but de cette section est d'analyser quatre politiques de remplacement qui ont été

proposées durant les 10 dernières années. Nous comparons leur efficacité aux politiques LRU et Random. Nous évaluons l'impact de chaque politique sur l'*Instruction Per Cycle* (IPC) et le *Miss Per Kilo Instruction* (MPKI).

2.2.2 Efficacité des politiques de remplacement

Métriques d'évaluation

Il existe principalement deux métriques pour l'évaluation des politiques de remplacement : le MPKI et l'IPC. Ces métriques sont définies comme suit :

$$MPKI = \frac{Total\,miss}{Total\,instructions/1000} \tag{2.1}$$

$$IPC = \frac{Total\,instructions}{Total\,cycles} \tag{2.2}$$

Lors de l'exécution d'une application avec différentes politiques de remplacement, le nombre d'instructions exécutées est identique, mais le nombre de *miss* varie. Une stratégie efficace verra le MPKI diminuer en conséquence.

L'IPC permet de mesurer l'efficacité du CPU. Une fois de plus, puisque le nombre d'instructions exécutées avec deux politiques est identique, seul le nombre de cycles total pour exécuter ces instructions diffère. Ce nombre de cycles dépend de l'efficacité de la hiérarchie mémoire. Plus le nombre de *miss* est faible à tous les niveaux de cache, moins le processeur est bloqué en attente des données, et plus l'exécution est rapide. Cela est d'autant plus vrai pour le cache de dernier niveau qui est sur le chemin critique vers la mémoire principale. Une politique efficace réduit donc le MPKI, ce qui réduit la probabilité pour une requête de suivre ce chemin critique, et l'IPC augmente.

Profils des accès à la mémoire

La politique de remplacement est responsable de l'évincement des blocs sur une ligne de cache lorsque celle-ci est pleine. Lors d'un *miss*, un bloc est sélectionné comme étant le bloc *victime* et est remplacé. Une telle décision est basée sur une heuristique que le LLC construit et alimente au fur et à mesure de l'exécution. Cette heuristique doit prédire de manière juste la prochaine ré-utilisation d'un bloc dans le futur, que l'on nomme la *distance de ré-utilisation*. Si cette distance de ré-utilisation est dans un futur proche, le bloc n'est pas prioritaire pour l'évincement et doit rester dans le cache pour éviter un *miss*. Sinon, la politique affecte au bloc une priorité plus élevée pour l'évincement. Ainsi, l'heuristique doit éviter au maximum les mauvaises prédictions.

Les applications ont différents schémas d'accès à la mémoire qui ont un impact sur

l'efficacité de la politique de remplacement. Ces modèles sont divisés en quatre catégories [47] : *recency-friendly*, *trashing*, *streaming* et *mixed*, respectivement présentés par les Formules (2.3), (2.4), (2.5) et (2.6). Chaque a représente un accès et k le numéro de bloc accédé.

$$(a_1, a_2, ..., a_k, a_{k-1}, a_{k-2}, ..., a_2, a_1)^N \text{ pour tout k} \tag{2.3}$$

$$(a_1, a_2, ..., a_k)^N \text{ for k} > \text{ taille de la mémoire cache} \tag{2.4}$$

$$(a_1, a_2, a_3, ..., a_k) \text{ pour k} = +\infty \tag{2.5}$$

$$\{(a_1, ..., a_k)^A P_\epsilon(a_1, a_2, ..., a_k, a_{k+1}, ..., a_m)\}^N$$
$$\text{pour k} < \text{taille du cache, m} > \text{taille du cache, } 0 < \epsilon < 1, \{A, N\} \geq 1 \tag{2.6}$$

Avec le schéma *recency-friendly*, les blocs ont une ré-utilisation quasi immédiate. Un schéma *trashing* dénote un accès cyclique à k blocs qui se répète N fois. Le modèle *streaming* apparaît lorsque les blocs n'ont aucune localité dans leur utilisation. Ce modèle est parfois appelé *scan*. Enfin, le schéma *mixed* est une combinaison des trois autres.

2.2.3 Présentation de différentes politiques de remplacement

Dans cette section, nous décrivons le fonctionnement des politiques de remplacement évaluées. La Figure 2.6 à la fin de cette section présente l'organisation d'un cache de dernier niveau avec les structures de données qu'il faut ajouter pour ces politiques. La Figure 2.7, également à la fin de cette section, donne un exemple d'exécution pour chacune des politiques.

Least-Recently Used

La politique LRU trie les bloc en fonction de leur usage dans le temps. Les blocs sont ordonnés de 0 à $N-1$, où N est l'associativité de la mémoire cache. Quand un bloc B est utilisé, il devient le bloc 0 et toutes les positions des autres blocs sont incrémentées de 1. Lors d'une éviction, le bloc $N-1$ est le bloc sélectionné comme victime. L'heuristique LRU prédit une courte distance de ré-utilisation pour les blocs récemment utilisés. A l'inverse, les blocs avec une position proche de N dans la liste ont une longue distance de ré-utilisation. Cette politique fonctionne bien avec un schéma d'accès *recency-friendly* et aussi avec un schéma *trashing* où le nombre k de blocs accédés est plus petit ou égal au nombre de blocs dans le cache.

Static Re-Reference Interval Prediction

Proposé par Jaleel et al. [47], le but de la politique SRRIP est de prédire l'intervalle d'usage des blocs en évitant de remplir le cache avec des blocs dont la distance de ré-utilisation est grande. Pour cela, une métrique appelé *Re-Reference Interval Prediction* (RRPV) est ajoutée à chaque bloc de la mémoire cache (voir Figure 2.6). Cette valeur est codée sur M bits et varie entre 0 et $2^M - 1$. Lorsque $RRPV \leq 1$, l'intervalle de ré-utilisation du bloc correspondant est prédit comme immédiat. Lorsque $RRPV = 2^M - 1$, la distance de ré-utilisation est prédite comme lointaine. Dans les autres cas, la distance est dite intermédiaire.

Lors de l'insertion d'un bloc dans une ligne de cache, la valeur du RRPV est initialisée à $2^M - 2$, soit une distance de ré-utilisation intermédiaire. Lors du second accès à ce même bloc, la politique SRRIP change le RRPV à 0. Le bloc est prédit pour être ré-utilisé dans un futur immédiat. Lors d'une éviction, la politique SRRIP cherche un bloc tel que $RRPV = 2^M - 1$, soit un bloc avec une distance de ré-utilisation lointaine. Si un tel bloc ne se trouve pas sur la ligne de cache, les RRPV de tous les blocs de la ligne sont incrémentés de 1 et le processus est recommencé jusqu'à ce qu'un bloc respecte le critère de sélection. L'avantage d'insérer un bloc avec une distance de ré-utilisation intermédiaire de $2^M - 2$ est de donner plus de temps à la politique SRRIP pour apprendre le schéma d'accès au bloc. En effet, la valeur initiale du RRPV ne permet pas d'évincer un bloc juste après son ajout dans le cache. Ensuite, soit le bloc est rapidement ré-utilisé, et son RRPV devient 0, soit il ne l'est pas et sera évincé au bout d'un temps T si aucun bloc sur la ligne ne correspond au critère d'évincement $RRPV = 2^M - 1$. Le désavantage de SRRIP est que le cache peut être pollué par des blocs ayant seulement deux accès puisque SRRIP prédit un usage immédiat après le second accès.

Avec la politique SRRIP, un bloc qui est utilisé une seule fois a plus de probabilité d'être évincé comparé à un bloc qui est au moins ré-utilisé une fois, même si cette ré-utilisation n'est pas fréquente.

Cette politique est efficace contre les modèles d'accès *streaming* puisque les blocs n'ayant aucune localité temporelle ne polluent pas le cache. Elle est également efficace dans le cas de schémas *trash* lorsque le nombre k de blocs accédés est plus petit ou égal au nombre de blocs de la mémoire cache.

Dynamic Re-Reference Interval Prediction

Lorsque la distance de ré-utilisation d'un bloc est plus grande que la taille de la mémoire cache, cela signifie que l'on est en présence d'un modèle d'accès de type *trashing*, avec un nombre $k > associativite$. Dans ce cas, la politique SRRIP devient inefficace.

Pour contrer cette situation, Jaleel et al. ont proposé la politique DRRIP, pour *Dynamic Re-Reference Interval Prediction*, comme une extension de SRRIP. Lors d'une insertion, le RRPV d'un bloc a une plus forte probabilité d'être initialisé à une valeur de $2^M - 2$ comme avec SRRIP. Cependant, il existe une probabilité que le RRPV soit mis directement à la valeur maximale, $2^M - 1$. Ainsi, pour les applications ayant un schéma d'accès de type *trashing*, chaque bloc inséré est évincé directement après son usage et ne pollue pas le cache. Comme avec SRRIP, les blocs qui sont utilisés voient leur valeur de RRPV mise à 0 et restent dans le cache. Pour détecter précisément le modèle d'accès à la mémoire d'une application, les auteurs utilisent la technique du Set-Dueling [90] (voir Figure 2.6). 32 lignes de la mémoire cache de dernier niveau sont dédiées à la politique SRRIP seulement et 32 lignes à la politique DRRIP seulement. Un compteur global, initialisé à zéro, est incrémenté et décrémenté de 1 respectivement lors d'un cache *miss* sur les lignes dédiées à SRRIP et DRRIP. Lorsque la valeur de ce compteur est supérieure à zéro, cela signifie que la politique SRRIP génère plus de cache *miss* que DRRIP. A l'inverse, quand le compteur est inférieur ou égal à zéro, c'est la politique DRRIP qui est moins efficace. Le cache utilise ce compteur global pour adapter dynamiquement la politique de remplacement et choisir entre SRRIP et DRRIP pour tout le cache (excepté les 64 lignes dédiées). Cela permet de choisir l'algorithme le plus efficace entre les deux pour gérer les modèles d'accès *streaming* ou *trashing*.

Un des problèmes de la politique DRRIP est sa granularité de prédiction. En fonction du compteur global, tout les blocs de la mémoire cache suivent la politique sélectionnée. Il n'existe pas de prédiction au niveau d'un bloc seulement.

Signature-based Hit Prediction

Une des limitations de la politique SRRIP est que la prédiction de distance de réutilisation lors de l'insertion est statique. L'algorithme DRRIP adresse ce problème en choisissant aléatoirement entre deux possibilités. Néanmoins, le choix lors de l'insertion est toujours statique et la phase d'apprentissage est faite après. La stratégie d'insertion ne prend pas en compte l'activité passée des blocs de la mémoire cache. La politique SHiP [114] (*Signature-based Hit Predictor*) s'attaque à cet aspect en faisant des prédictions en fonction des schémas d'accès pour chaque bloc. Wu et al. ont ajouté à côté du LLC une table de compteurs de saturation (*saturating counters*) indexée par un *hash* appelé une signature (voir Figure 2.6). Les signatures sont générées en fonction du compteur ordinal (CO) de l'instruction à l'origine de l'accès.

Pour mettre en place cette stratégie, deux champs ont été ajoutés dans les méta-données de la mémoire cache : la signature et le bit de résultat (*outcome bit*). Ces deux champs sont initialisés à 0. Lors de l'insertion d'un bloc, la signature est mise à jour en fonction du

CO. Si un second accès a lieu sur ce bloc et que c'est un *hit*, le bit de résultat est mis à 1. Cela signifie que ce bloc a été ré-utilisé au moins une fois depuis son insertion dans la ligne de cache. L'entrée correspondante à ce bloc dans le prédicteur est alors incrémentée de 1. Dans le cas d'un *miss* lors du second accès, l'entrée correspondante dans le prédicteur n'est pas modifiée. Cette entrée est décrémentée lors de l'évincement d'un bloc si le bit de résultat est toujours à 0, signifiant que ce bloc n'a jamais été ré-utilisé depuis qu'il a été inséré. Avec cette technique, le prédicteur connaît les CO qui génèrent des accès à des blocs n'ayant aucune ré-utilisation, et peut donc agir en conséquence. Lors de l'insertion d'un nouveau bloc, le prédicteur est consulté. Si la signature pointe vers une entrée dont la valeur est 0, la distance de ré-utilisation du bloc est prédit comme lointaine. Toute autre valeur est traitée comme une distance intermédiaire. De cette manière, la distance de ré-utilisation des blocs est prédit dynamiquement en fonction du CO responsable de l'accès.

Assigner une signature pour chaque bloc de la mémoire cache n'est pas réaliste pour des raisons de ressources matérielles. L'implémentation proposée par Wu et al. échantillonne 64 lignes de cache pour entraîner le prédicteur. Celui-ci est mis à jour uniquement lors des accès sur ces 64 lignes, mais il est consulté à chaque accès. Ainsi, une petite partie des blocs dirige les prédictions pour le cache entier.

Cette politique offre deux principales contributions. Premièrement, la possibilité de faire des prédictions sur le comportement de l'application grâce au CO. Deuxièmement, chaque bloc a sa propre signature, ce qui rend les prédictions plus justes que les politiques SRRIP ou DRRIP. Cette politique est résistante aux schémas d'accès *trashing* grâce aux prédictions qui reposent sur la signature.

Hawkeye

Le politique Hawkeye [46] est basée sur l'algorithme théorique MIN [5]. Pour chaque ligne de cache, MIN classifie les blocs en deux catégories : *hit blocks* et *miss blocks*. Cette classification a été prouvée optimale [6], i.e., le nombre de *miss* trouvé par l'algorithme MIN ne peut pas être réduit.

Lors de l'évincement d'un bloc, il est impossible de savoir immédiatement si la décision prise est la meilleure. Il faut attendre la prochaine ré-utilisation de bloc qui vient d'être évincé. A titre d'exemple, imaginons un cache avec 2 blocs par ligne et une seule ligne contenant B_0 et B_1. B_1 est le bloc victime lors de l'insertion d'un nouveau bloc B_2. La ligne contient maintenant B_0 et B_2. Si le prochain accès à la ligne est B_1, la décision initiale d'évincement n'est pas optimale et B_0 aurait du être évincé. L'algorithme MIN est capable de détecter cela et classifie B_1 comme un *hit block* et B_0 comme un *miss block*, ce qui conduit à l'éviction de B_0. Néanmoins, cet algorithme est théorique et repose sur la

connaissance des futures transactions.

Jain and Lin proposent un mécanisme appelé *OPTgen* qui est capable de reconstruire l'algorithme MIN sur un ensemble de transactions passées (voir Figure 2.6). Les accès passés sur les blocs sont alors séparés en deux catégories. La politique Hawkeye fait l'hypothèse que ce qui s'est précédemment passé sur un bloc a une forte probabilité de recommencer dans le futur. Les auteurs ont d'ailleurs montré que cette hypothèse est vérifiée dans 90% des cas. Ainsi, ils utilisent des prédictions de l'algorithme MIN sur le passé pour guider les futures décisions.

Le prédicteur Hawkeye utilise le même concept d'échantillonnage que Wu et al. Un sous-ensemble du LLC est surveillé dans le but de minimiser le coût matériel d'implémentation tout en maintenant une certaine justesse dans la représentation de l'activité du cache. Le prédicteur est entraîné seulement lorsque les lignes monitorées sont accédées. Un tableau de compteurs de saturation indexé par les CO est utilisé. Ce tableau est mis à jour à chaque accès au mécanisme *OPTgen*. Lorsque *OPTgen* prédit un *block hit*, l'entrée correspondante est incrémentée de 1. Lorsqu'il prédit un *block miss*, cela signifie que quelque soit le choix du bloc victime, le prochain accès à ce bloc sera un cache *miss*. Dans ce cas, l'entrée correspondante est décrémentée de 1. Ainsi, ce bloc peut être évincé et remplacé par une donnée utile.

Les blocs dans le cache sont ordonnés en fonction de la valeur de leur RRPV. Une valeur haute correspond à une forte priorité pour l'éviction. Les blocs qui sont classifiés comme *block miss* dans le futur voient leur valeur de RRPV mise à $2^M - 1$, soit la valeur maximale. Ceux classifiés comme *hit block* dans le futur ont une valeur de $RRPV = 0$. Lors de l'insertion d'un bloc, si *OPTgen* prédit un futur cache *hit* sur ce dernier, les RRPV de tous les blocs de la ligne sont incrémentés de 1. Les blocs prédits comme *hit blocks* dans le futur ne peuvent jamais atteindre la valeur maximale du RRPV. Ainsi, ceux prédits comme *miss blocks* auront toujours une priorité plus grande pour l'évincement. S'il lors d'une éviction il n'y a pas de blocs avec une valeur $RRPV = 2^M - 1$, le bloc avec le RRPV le plus élevé est choisi comme victime et est évincé.

Cette politique est efficace face à des modèles d'accès de type *streaming*, *trashing* et *mixed*. Grâce à l'algorithme MIN qui est capable de prédire très précisément l'activité future des blocs, chaque comportement est détecté et peut être anticipé. Ainsi, seul les blocs utiles restent dans le cache.

Résumé

Les structure de données des différentes politiques de remplacement présentées précédemment sont visibles sur la Figure 2.6. Les rectangles rayés en rouge sur le LLC représentent les lignes échantillonnées. Elles sont copiées dans le *sampler*, en bas du LLC sur la

figure. Le *sampler* contient les méta-données des lignes comme le tag ou le CO des blocs. Lors d'une requête, le LLC est accédé et le *sampler* vérifie que l'accès en cours se fait ou non sur une ligne échantillonnée. Si oui, il se met à jour.

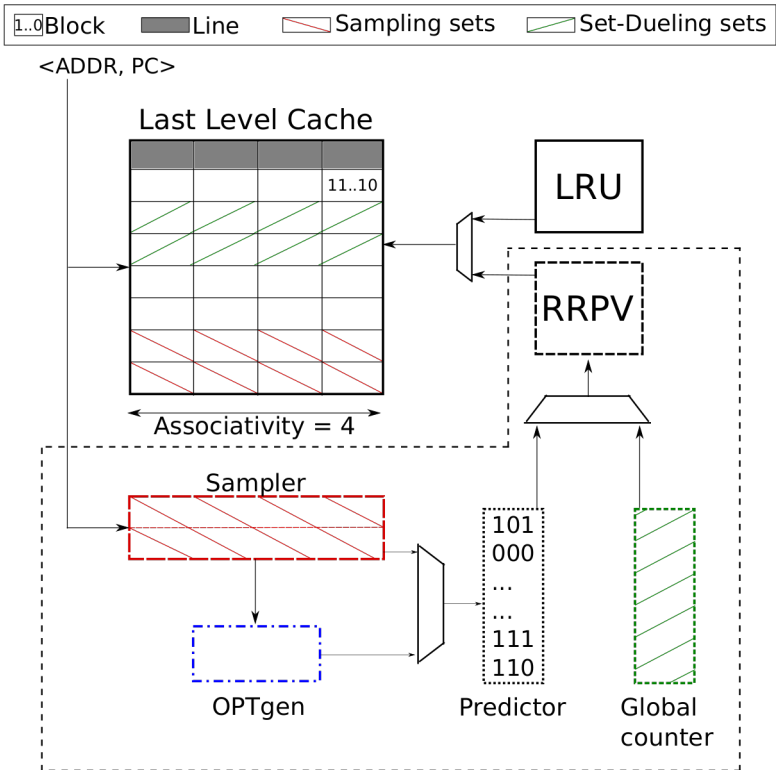


FIGURE 2.6 – Structures de données pour le LLC. La configuration par défaut pour la LRU est en dehors des pointillés. La politique SRRIP utilise le RRPV ; DRRIP utilise le RRPV, le Set-Dueling et le Global Counter ; SHiP utilise le Sampler, le Predictor et les structures de SRRIP ; Hawkeye utilise OPTgen en plus des structures de SHiP.

Le *sampler* est connecté au prédicteur. Quand le premier est accédé, ce dernier est mis à jour. La politique Hawkeye nécessite une étape supplémentaire entre le *sampler* et le prédicteur appelée *OPTgen*. Cette étape reproduit l'algorithme MIN présenté précédemment. Le résultat de cette étape est utilisé pour entraîner le prédicteur. Le prédicteur est utilisé pour changer la valeur des heuristiques représentées par les deux carrés à droite du LLC. La LRU met à jour les bits LRU seulement. SRRIP, DRRIP, SHiP et Hawkeye mettent à jour les RRPV.

Le mécanisme de *Set-Dueling* est représenté par les rectangles rayés en vert. Les rectangles dans le LLC représentent les lignes monitorées et en fonction de leur activité, i.e., *hit* ou *miss*, le compteur global est modifié.

Les politiques de remplacement évaluées ici peuvent être classées en deux catégories : gros grain et grain fin. SHiP et Hawkeye sont considérées comme des politiques de rem-

	Receny resistant	Trash resistant	Scan resistant	Mixed resistant	Set-Dueling	Sampling	HW budget
LRU	+++	+	+	+	✗	✗	16Ko
Random	+	+	+	+	✗	✗	0Ko
SRRIP	+++	++	+	+	✗	✗	12Ko
DRRIP	++	+++	++	+	✓	✓	13.6Ko
SHiP	+++	+++	+++	++	✗	✓	35Ko
Hawkeye	+++	+++	+++	+++	✗	✓	31.8Ko

Tableau 2.1 – Caractéristiques des politiques de remplacement évaluées. Le budget matériel correspond à un LLC de 2Mo 16 associatif.

placement à grains fins de par leur usage des CO pour faire des prédictions adaptées à chaque bloc. A l'inverse, LRU, SRRIP et DRRIP sont considérées comme des politiques de remplacement à gros grains car elles n'ont pas de mécanisme d'identification des blocs comme les CO. Leur système de prédiction est global à tout le cache.

En contrepartie, les coûts d'implémentation matériel de ces politiques sont peu élevés comparé à SHiP et Hawkeye, même si ces dernières utilisent l'échantillonnage pour réduire les coûts. Le tableau 2.1 résume les politiques de remplacement présentées en fonction de leur efficacité face aux schémas d'accès à la mémoire. Deux fonctionnalités sont également mentionnées, le *Set-Dueling* et l'échantillonnage. Nous reportons enfin le budget matériel des structures de données additionnelles. Ces budgets sont valables uniquement pour la configuration de notre étude et peuvent différer des coûts existant dans la littérature. A titre d'exemple, notre implémentation de SHiP échantillonne 256 lignes de cache tandis que Wu et al. mentionnent 64 lignes.

FIGURE 2.7 – Exemple d'application de chacune des politiques de remplacement évaluées

2.2.4 Évaluation des performances

Environnement de simulation

Nous évaluons les politiques de remplacement avec le simulateur ChampSim [52] utilisé pour le *Cache Replacement Championship*. Des précisions supplémentaires seront apportées sur ce simulateur dans le chapitre 4. Le système modélisé est décrit par le tableau 2.2.

L1 (I/D)	32Ko, 8-way, LRU, Privé, 4 cycles
L1 D prefetcher	next-line
L2	256Ko, 8-way, LRU, Unifié, 8 cycles
L2 prefetcher	Basé sur le CO
L3	2Mo, 16-way, Partagé, 20 cycles
L3 énergie/accès	0.217 nJ
Puissance L3	79.34 mW
Nombre de CPU	1
Fréquence CPU	Out-of-Order, 4GHz
Taille DRAM	4Go, hit : 55 cycles, miss : 165 cycles

Tableau 2.2 – Configuration du système

Nous exécutons 20 traces d'applications de la suite SPEC CPU2006. Chaque cœur exécute 1 milliard d'instructions avec un période de *warm-up* de 200 millions d'instructions. La moyenne de l'IPC est calculée en appliquant la moyenne géométrique sur les IPC mesurés pour les 20 applications.

Deux architectures sont évaluées : *i*) un cœur sans *prefetching* et *ii*) un cœur avec *prefetching*. Sur ces deux architectures, nous évaluons le MPKI et l'IPC.

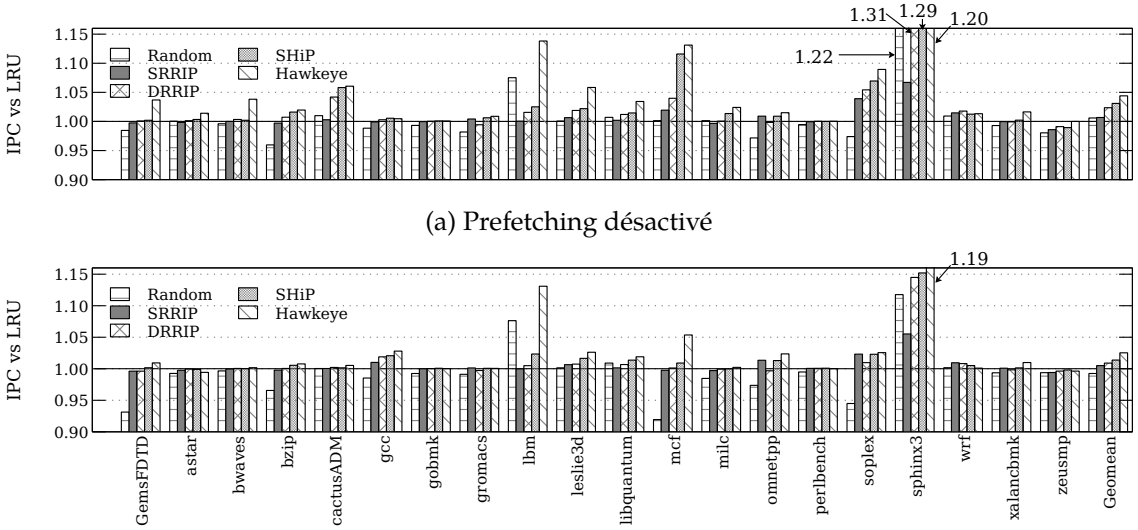
Résultats expérimentaux

La Figure 2.8a détaille les résultats d'IPC avec une configuration sans *prefetching*. Les résultats sont normalisés à la politique LRU. En moyenne, toutes les politiques ont un IPC supérieur à la LRU. La plus efficace est la politique Hawkeye avec un *speedup* de $1.04\times$. Les applications dites *memory intensive* comme *lbm*, *mcf* ou *soplex* montrent une accélération supérieure aux autres. La meilleure amélioration de performance est atteinte par l'application *sphinx3*, où les politiques Random, SRRIP, DRRIP, SHiP et Hawkeye donnent respectivement un *speedup* de 1.22, 1.06, 1.31, 1.29 et 1.20. On note également que Hawkeye est la seule politique avec des résultats constants sur toutes les applications sans aucune dégradation des performances.

Les résultats de performance pour une plateforme monocœur avec *prefetching* sont

donnés par la Figure 2.8b. On observe des tendances similaires, mais le *speedup* est moins important que précédemment. De plus, la politique Random est maintenant moins efficace que la LRU. Le *speedup* moyen de quatre autres politiques varie entre $\times 1.01$ (SRRIP) et $\times 1.03$ (Hawkeye).

Cependant, le cache est plus efficace lorsque le *prefetching* est activé. La Figure 2.9 montre que le MPKI moyen est drastiquement diminué de 48.2% en moyenne. Cette amélioration affecte l'IPC qui est augmenté de 16% en moyenne pour toutes les configurations de cache. Le *prefetching* a donc un impact positif sur le MPKI et l'IPC.



(a) Prefetching désactivé

(b) Prefetching activé

FIGURE 2.8 – Amélioration de l'IPC normalisé à la LRU pour une plateforme monocœur sans prefetching

On note également que cet effet positif est plus visible avec la LRU que les autres politiques de remplacement, comme montré par la Figure 2.9.

En effet, les politiques de remplacement et le système de *prefetching* sont deux mécanismes distincts qui ne communiquent pas. Pour SHiP et Hawkeye, leur système de prédiction repose sur les CO des blocs accédés. Or, le *prefetcher* génère des accès supplémentaires au cache qui n'ont pas de CO, rendant impossible les prédictions. Cette situation entraîne deux conséquences. Premièrement, le cache peut être pollué par de mauvais blocs amenés par le *prefetcher* et qui ne peuvent pas être détectés. Deuxièmement, les algorithmes d'évincement peuvent prendre des décisions en conflit avec le *prefetcher* et évincer des blocs qui doivent rester dans le cache.

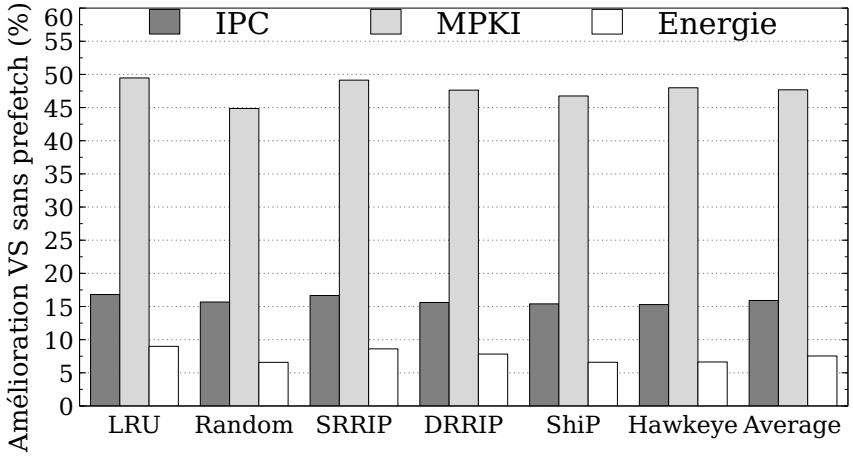


FIGURE 2.9 – Amélioration de l'IPC, du MPKI et de la consommation énergétique normalisé à une configuration sans prefetching

Résumé

Cette étude sur les politiques de remplacement permet de tirer plusieurs conclusions.

— l'algorithme le plus efficace est Hawkeye. Bien que notre étude soit limitée à une architecture monocœur et un cache de dernier d'une capacité de 2Mo, cette observation reste vraie pour différentes tailles de cache, des architectures multi-cœurs et avec un système de *prefetching* [34, 46].

— les politiques de remplacement peuvent entrer en conflit avec le système de *prefetching*

— le *prefetching* génère beaucoup de trafic supplémentaire. Au niveau du LLC, cela se traduit par un nombre accru d'écritures

Pour la suite de nos travaux, nous considérons uniquement la politique Hawkeye. De plus, le système de *prefetching* est écarté du fait de ses effets négatifs.

2.3 Environnements logiciel de simulation de caches et de consommation énergétique

Dans cette section, on s'intéresse aux différents niveaux d'abstraction possibles dans le cas d'explorations architecturales. Nous présenterons ces différents niveaux et justifierons notre choix parmi ces derniers. On présentera ensuite quelques simulateurs d'architectures répandus dans la littérature relative à la micro-architecture. Ces derniers sont résumés dans le tableau 2.4. Enfin, on s'intéressera aux outils d'estimation de consommation énergétique des processeurs.

2.3.1 Niveaux d'abstractions architecturaux

Pour un designer, l'étape d'analyse et d'exploration architecturale nécessite la mise en place d'outils. Le choix de ces outils est difficile et se résume souvent à un compromis entre la justesse de simulation, la rapidité et le coût budgétaire. Les modèles d'abstraction présentés par la suite sont résumés dans le tableau 2.3.

L'achat de puces contenant des nouvelles technologies et des capteurs de mesure permet de mesurer facilement leur impact. La rapidité d'exécution permet des observations simples et rapides. Cependant, de tels systèmes sont rares et chers, ce qui les rend difficiles d'accès. De plus, ces puces restreignent le designer à la phase d'analyse de l'existant. En effet, l'architecture n'est pas modulaire et l'ajout de fonctionnalités est impossible.

La simulation matérielle permet d'ajouter la modularité manquante à des architectures réelles. A ce niveau, tous les composants de la plateforme sont simulés, des portes logiques aux transistors. La précision est par conséquence excellente. L'utilisation de ce type de simulation est néanmoins très compliqué à mettre en place. Il est nécessaire de faire le design de toute l'architecture au niveau des portes logiques, ce qui est trop long à l'échelle d'un processeur. Des modèles librement accessibles existent au format RTL, mais la synthèse au format matériel n'est pas un processus entièrement automatique et nécessite une lourde phase de design au niveau circuit. C'est une étape principalement réservée aux industriels lorsque l'on considère un processeur complet.

La simulation RTL se place à un niveau supérieur d'abstraction. Plutôt que simuler tous les composants jusqu'au niveau le plus bas, c'est le comportement des blocs logiques entre eux qui est modélisé. Le précision se place alors au niveau du cycle, pour une flexibilité moyenne. Les changements dans l'architecture nécessitent une phase d'implémentation non négligeable. De plus, l'exploration de nouvelles technologies implique de connaitre les détails quant aux comportements de celles-ci lors des opérations de lecture, d'écriture ou de transfert de données. Ces informations bas niveau ne sont disponibles qu'à l'aide d'un partenariat industriel. Il en est de même pour l'évaluation de consommation énergétique, qui requiert des modèles construits sur la base de simulations matérielles.

Les cartes FPGA sont une solution très flexible. Un designer peut implémenter une architecture existante pour la phase d'analyse, puis la modifier pour ajouter des optimisations. Il est également possible d'implémenter des compteurs de performance pour mesurer l'activité de la mémoire ou des cœurs. Bien que plus lentes qu'un système réel,

les cartes FPGA représentent une bonne alternative en termes de rapidité. Cependant, ces cartes impliquent de maitriser un vaste écosystème de technologies mêlant différents langages de programmation et une compréhension fine à un niveau identique aux simulations RTL. Les FPGA sont également limités par leur puissance. On s'intéresse dans cette thèse à des nœuds de calcul avec une fréquence minimale d'environ $1GHz$. Une telle puissance peut rarement être atteinte par des FPGA. De plus, de la même manière que l'abstraction RTL, la modélisation de nouvelles technologies nécessite un partenariat industriel.

Le dernier niveau d'abstraction est la simulation logicielle, qui repose sur des modèles comportementaux à très haut niveau pour les composants de l'architecture. Aucune notion d'électronique n'est nécessaire dans ce cas. Seul le fonctionnement des composants est modélisé. Ces simulateurs sont par conséquent très flexibles, mais perdent en précision. En général, la justesse s'approche du cycle. Cependant, leur flexibilité et les faibles coûts d'implémentation sont un réel avantage. La modélisation de mémoires émergentes peut se faire en prenant en compte simplement quelques caractéristiques, comme leurs latences plus élevées. Néanmoins, la vitesse de simulation peut être très lente en fonction de la taille de l'architecture et des applications exécutées. Enfin, les modèles énergétiques ne sont pas intégrés dans ces simulateurs et doivent être ajoutés.

Dans cette thèse, nous avons fait le choix du niveau d'abstraction logiciel. Ce choix nous permet de mettre en place rapidement un environnement de simulation sans un effort conséquent de modélisation de circuits. De plus, cela nous permet de modéliser facilement des mémoires non volatiles émergentes. Ce point sera détaillé au chapitre suivant.

Abstraction	Précision	Flexibilité	Exécution	Coût ($)	Commentaire
Implém. matériel	Meilleure	Aucune	Temps réel	Élevé	Nécessite de créer puis fabriquer le circuit
Simu. matérielle	Forte	Lourde	Très lente	Aucun	Processus de création très long
RTL	Cycle/Bit	Moyenne	Lente	Aucun	Besoin de modèles de consommation d'industriels
FPGA	Cycle/Bit	Moyenne	Rapide, ∼temps réel	Faible	Écosystème lourd, limitations physiques
Simu. logicielle	Quasi-cycle	Simple	Lent	Aucun	Abstraction du matériel, perte de justesse

Tableau 2.3 – Récapitulatif des niveaux d'abstraction de simulation

2.3.2 Simulateurs d'architectures

SimpleScalar [16] est un framework de simulation et de modélisation d'architectures. Il supporte différents jeux d'instructions comme Alpha, ARM, PowerPC et x86. Il propose également différents modèles de processeur allant d'une version sans pipeline à une version avec plusieurs niveaux de caches, une exécution dans le désordre (*Out-of-Order*) etc. Cinq modes de simulation sont proposés, allant du plus rapide et imprécis au plus lent mais plus précis. Le taux d'erreur a été mesuré face a une vraie plateforme et est de 3%. SimpleScalar n'est plus développé depuis 2011 et beaucoup de fonctionnalités importantes manquent aujourd'hui, comme le multicœur, le choix des systèmes d'interconnexion ou encore la simulation d'une mémoire principale.

ChampSim [52] est un simulateur développé dans le cadre du *Cache Replacement Championship* [34]. C'est un simulateur *trace-driven* permettant de simuler des plateformes monocœurs et multicœurs avec une hiérarchie mémoire fixe à trois niveaux de cache. L'utilisateur peut configurer quelques paramètres de ces caches comme la taille ou l'associativité. La fréquence du cœur ou la taille et l'organisation de la mémoire principale sont également paramétrables. Plusieurs mécanismes de *prefetching* sont proposés, et sont désactivables si nécessaire. Enfin, différentes politiques de remplacement sont proposées pour le cache de dernier niveau.

gem5 [9] est un simulateur open-source développé par une communauté d'académiques (University of California, University of Wisconsin-Madison, University of Virginia) et d'industriels (ARM, AMD, Google). Ce simulateur permet de simuler une architecture complète, du cœur du processeur à la mémoire principale. Sa précision est de l'ordre du cycle (*cycle-accurate*). Il existe différents modèles de cœurs, du plus basique au plus détaillé. Par exemple, le modèle le plus pointu propose le détail du pipeline d'exécution, le ré-ordonnancement d'instructions, la prédiction de branchement (avec plusieurs modèles de prédiction disponibles) etc. La hiérarchie mémoire est très flexible et beaucoup de paramètres architecturaux peuvent être facilement changés. Les systèmes d'interconnexion sont variés et on peut ajouter des périphériques d'accélération comme des GPUs. Enfin, c'est un simulateur dit *full-system* : on peut utiliser un noyau Linux complet qui fonctionnera sur l'architecture que l'on simule. Plusieurs travaux ont mesuré le taux d'erreur du simulateur gem5 [17, 36]. Les auteurs ont montré que le taux d'erreur varie de 1 à 20%, et est en moyenne de 5%.

MARSS-X86 [80] est un simulateur *cycle accurate* basé sur l'émulateur QEMU [7] et le simulateur PTLSim [119]. Le système d'émulation des cœurs et de la hiérarchie de

caches de QEMU a été remplacé par celui de PTLSim pour une plus grande précision. MARSS-X86 est un bon compromis entre la vitesse d'exécution et la précision. Comme gem5, c'est un simulateur *full-system* permettant l'exécution d'un système d'exploitation complet. Néanmoins, ce simulateur n'est plus développé depuis 2012 et les modèles de cœurs ou d'interconnexions sont obsolètes.

Sniper [19] est un simulateur de systèmes massivement multicœurs utilisant comme une abstraction appelé "simulation par intervale", ou *interval simulation* [32]. Ce modèle permet un bon compromis entre précision et rapidité de simulation. La justesse de ce simulateur a été évaluée sur des processeurs x86 des familles Nehalem et Xeon en utilisant des applications de la suite SPLASH-2 [113]. Les résultat ont montrés que le taux d'erreur est respectivement de 11% et 25%. Du fait de son niveau d'abstraction, Sniper ne supporte pas la simulation *Full-system*. De plus, cette abstraction diminue le nombre d'informations disponibles au niveau micro-architectural, pouvant influencer la qualité de l'estimation de consommation énergétique.

Simulateur	SimpleScalar	ChampSim	gem5	MARSS-x86	Sniper
Full-system	✗	✗	✓	✓	✗
Multicœur	✗	✓	✓	✓	✓
Caches	✓	✓	✓	✓	✓
Interconnexion	✗	✗	✓	✓	✓
Mémoire DRAM	✗	✓	✓	✓	✓
Vitesse à grain fin	350 KIPS	Non mesurée	1 KIPS	200 KIPS	Non mesurée
A jour	✗	✓	✓	✗	✓
Licence libre	✓	✓	✓	✓	✓

Tableau 2.4 – Récapitulatif des simulateurs (KIPS = Kilo-instruction par seconde)

2.3.3 Évaluation de consommation énergétique

McPAT [62] est un framework d'évaluation de surface et de puissance pour les processeurs. McPAT propose des modèles d'évaluations pour les architectures ARM, Alpha, Intel et SPARC. Il fonctionne dans un mode *post-simulation*. Une simulation détaillée doit dans un premier être effectuée par un autre outil. Ensuite, les résultats de cette simulation doivent être extraits et convertis dans le format d'entrée de McPAT (XML). Typiquement, McPAT a besoin de connaître le type de cœur utilisé (*High-Performance* ou *Low-Power*), le jeu d'instruction pour adapter son modèle de processeur, le nombre d'accès aux dif-

férentes unités de calcul du cœur (Integer/Floating Point Unit, ReOrdering Buffer etc), les tailles de caches etc. Le framework McPAT a été validé par les auteurs face à de vrais processeurs, notamment Intel, ARM et SPARC [62]. Ces derniers ont mesuré des taux d'erreur pour l'estimation de la consommation énergétique allant de 10.9% à 22.6%. Pour l'estimation de surface, l'erreur varie entre 16.7% et 27.3%. Xi et al. [116] ont utilisé un outil propriétaire pour valider McPAT face à un processeur POWER7™. Leurs résultats montrent que le taux d'erreur de McPAT est élevé pour l'énergie dynamique. En revanche, la modélisation de l'énergie statique est proche de la réalité.

PowMon [109] est un framework d'évaluation de consommation énergétique développé par l'université de Southampton en collaboration avec ARM. Les modèles énergétiques qui sont intégrés sont basés sur des mesures réalisées avec une carte Odroid-XU3 [4]. Elle est dotée de cœurs ARM et de compteurs de performance. Ces derniers sont utilisés pour calculer la consommation énergétique. En se basant sur cette carte, les auteurs ont sélectionné uniquement les compteurs de performance pertinents et ont proposé deux modèles de consommation énergétique qu'ils ont validés sur la carte. Les résultats montrent un taux d'erreur variant de 2.8% à 3.8%. Bien que validé expérimentalement, le modèle énergétique de PowMon est limité aux architecture ARM big.LITTLE™ et son intégration avec un simulateur n'est pour l'instant pas possible.

gem5 met à disposition une API permettant d'avoir une estimation de la consommation énergétique dans les statistiques de la simulation. Cette évaluation est donnée par composant ainsi que pour l'architecture complète. Cette approche nécessite à l'utilisateur d'implémenter pour chaque composant dont il veut calculer la consommation un modèle énergétique.

Outil	Perspective d'évaluation	Facilité d'utilisation	Intégration	Mémoire non volatile	Open-source
McPAT	« Bottom-up »	+++	++	✗	✓
PowMon	« Top-down »	+	+	✗	✓
gem5	« Bottom-up »	+	+++	✗	✓

Tableau 2.5 – Caractéristiques des outils d'évaluation de consommation énergétique

Perspectives d'évaluation. Les solutions présentées dans le tableau 2.5 peuvent être séparées en deux catégories : « bottom-up » et « top-down ».

4. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127

Une approche « bottom-up » estime la consommation énergétique en utilisant en entrée la spécification d'un CPU, la taille des caches, un nombre de transactions entre les composants etc, puis évalue la consommation en se basant sur des modèles génériques d'énergie pour chaque évènement (cycle horloge, cache *hit*, multiplication flottante etc). Cette approche est intéressante car elle est généraliste et peut s'utiliser sur plusieurs types de CPU. Néanmoins, sa précision est moins importante que la seconde perspective.

L'approche « top-down » utilise un modèle énergétique construit de manière empirique pour un type de CPU en particulier. Ce modèle est dérivé grâce à des compteurs de performance et d'énergie existant sur certaines puces. Ces compteurs permettent d'avoir des informations très précises sur la consommation énergétique et les évènements internes au CPU, comme le nombre de *miss* sur les caches, l'exécution spéculative des instructions etc. On peut alors relier les évènements mesurés à la consommation relevée, et élaborer un modèle de consommation. L'avantage de cette méthode est sa précision pour l'évaluation de la consommation. Cependant, elle est spécifique à un type de CPU et nécessite d'avoir accès à une carte avec des compteurs de performance et d'énergie.

Remarques

Les outils présentés dans cette section nécessitent une phase de calibration pour atteindre une précision acceptable. Les modèles de performance ou d'énergie qu'ils contiennent sont génériques et doivent être adaptés aux architectures explorées. Par exemple, le simulateur gem5 propose des modèles de cœurs de type ARM In-Order et Out-of-Order. Quelque soit le cœur que l'on désire simuler, Cortex-A7, Cortex-A9 ou Cortex-A57, le modèle utilisé par gem5 est le même. Or, ces cœurs ont des caractéristiques très différentes. La configuration de base du modèle ARM ne représente donc pas fidèlement la réalité. Une étape de calibration est nécessaire, par exemple en effectuant des simulations avec gem5 et en parallèle sur une vraie carte utilisant des cœurs ARM [18]. Il en est de même pour les modèles de consommation énergétique.

D'une manière générale, ces outils peuvent être efficaces mais nécessitent d'être adaptés à la situation.

2.4 Résumé

Dans ce chapitre, nous avons présenté le fonctionnement des caches de manière individuelle ainsi que collective au sein de la hiérarchie mémoire. Nous avons étudié la façon dont la mémoire est gérée en détaillant notamment les différentes mécanismes matériels responsables des écritures. Après avoir présenté la façon dont les caches gèrent la mémoire, nous avons proposé une étude sur les politiques de remplacement. Leur utili-

sation sera abordée dans la suite de ce manuscrit. Enfin, nous avons détaillé quels sont les outils logiciels permettant de simuler des architectures et d'évaluer leur consommation énergétique.

Les principes développés dans ce chapitre sont indépendants de la technologie utilisée. Dans le prochain chapitre, on se concentrera sur les technologies mémoires à destination des caches . Après une présentation des technologies CMOS classiques, on introduira les familles de mémoires non volatiles, et en particulier la STT-MRAM. Nous verrons quelles sont les questions soulevées par l'introduction de STT-MRAM dans une hiérarchie mémoire, quels sont les leviers possibles pour y répondre, et quelles ont été les propositions de la littérature.

Roads? Where we're going we don't need roads.

EMMETT BROWN - BACK TO THE FUTURE (1985)

Chapitre 3

Technologies de mémoires non volatiles

Ce chapitre présente les technologies mémoires existantes puis quelques technologies de mémoires non volatiles existantes dans la littérature. Nous proposons de nous focaliser

sur la STT-MRAM et soulevons différentes questions quant à son intégration dans une hiérarchie de caches. Nous verrons les réponses apportées par la littérature à ces questions, puis nous justifierons l'approche adoptée pour nos travaux. Enfin, on présentera les niveaux d'abstractions de simulation possibles et différents outils permettant de simuler des composants mémoires utilisant de la STT-MRAM.

3.1 Technologies classiques : SRAM, DRAM et FLASH

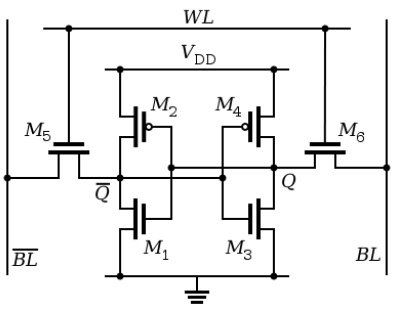
L'utilisation de la technologie mémoire SRAM est le choix le plus répandu dans nos processeurs aujourd'hui lorsqu'il s'agit de mémoire cache. Les caches sont des composants qui doivent avoir des temps d'accès très courts pour maximiser les performances. La technologie SRAM étant très rapide, quelques cycles pour un cache L1 [88], elle est le choix par défaut. Une cellule mémoire de SRAM (Figure 3.1a) est composée de six transistors connectés une *worldLine* (WL) et une *BitLine* (BL).

La DRAM est la technologie utilisée pour les mémoires principales. Les cellules DRAM ont l'avantage d'être plus petites car composées d'un seul transistor connecté à une WL et une BL (Figure 3.1b). On peut donc en combiner un grand nombre pour construire des composants avec une large capacité de stockage de l'ordre d'une dizaine de gigaoctets [98].

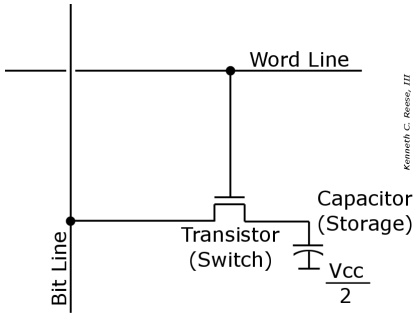
La mémoire FLASH est présente dans les disques de stockage SSD grand public depuis une dizaine d'années. Ces derniers remplacent petit à petit nos disques magnétiques classiques. La technologie FLASH est une technologie non volatile qui peut être construite à partir de cellules de type NOR ou NAND. Les NAND (Figure 3.1c) sont en général préférées pour leur faibles surface et leur coûts énergétiques inférieurs. Bien que non volatile, on ne considère plus la FLASH comme une technologie émergente. En effet, celle-ci est suffisamment mature aujourd'hui pour être commercialisée à grande échelle.

Avec les progrès relatifs à la finesse de gravure, on a pu augmenter le nombre de transistors par μm^2, et ainsi augmenter la capacité de stockage des caches et des mémoires principales tout en conservant la même surface. La Figure 3.2 montre deux phénomènes intéressants quant à l'évolution de la densité de puissance des processeurs. Premièrement, la densité de la puissance augmente à mesure que la technologie est fine, autrement dit chaque nm^2 a besoin de plus en plus de puissance. Cela est dû à l'augmentation du nombre de transistors sur une même surface évoquée précédemment.

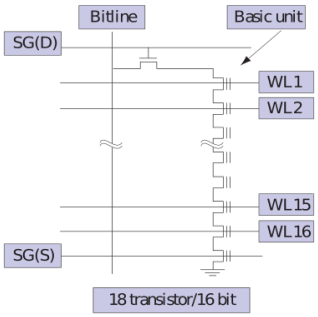
La seconde observation est la part occupée par le courant de fuite, ou *leakage* dans le total. Au départ mineur, on voit qu'il est de plus en plus important. La variation du *leakage* dépend de la température générale de la puce et non de l'activité du composant. La



(a) Cellule mémoire SRAM [4]



(b) Cellule mémoire DRAM [105]



(c) Cellule mémoire NAND [58]

FIGURE 3.1 – Composition des cellules mémoires SRAM, DRAM et NAND

Figure 3.2 permet d'observer que les composants ont un *leakage* de plus en plus important, sans pour autant être utilisés.

Lors d'une extinction de l'alimentation, les données stockées dans une cellule SRAM sont perdues, comme dans une mémoire DRAM. Ces technologies nécessitent donc une alimentation continue. De plus, la DRAM utilise un mécanisme de rafraîchissement des données pour éviter la perte d'information. C'est de cette alimentation continue que vient l'augmentation de la puissance statique. Des alternatives comme le *clock-gating* [115] ou le *power-gating* [41] ont été proposées. Néanmoins, le problème intrinsèque de l'alimentation continue n'est pas attaqué et ces techniques permettent simplement de le mitiger.

3.2 Présentation des mémoires non volatiles émergentes

A l'inverse des technologies SRAM et DRAM, de nouveaux types de technologies mémoires émergent depuis une dizaine d'années dans le monde de la recherche et de l'industrie : les mémoires non volatiles (*Non-Volatile Memory, NVM*). Dans cette section, nous expliquons le fonctionnement général de ces mémoires, donnons des exemples de diffé-

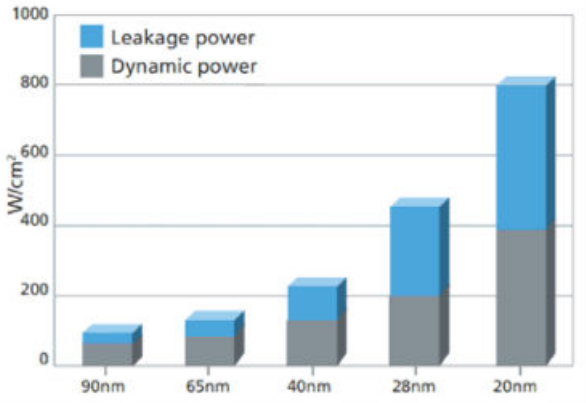


FIGURE 3.2 – Densité énergétique en fonction de la finesse de gravure [35]

rentes familles de NVM et nous choisissons la STT-MRAM pour la suite de nos travaux.

3.2.1 Principe de non volatilité

Pour stocker une information, les mémoires non volatiles utilisent un phénomène physique persistant dans le temps, par exemple la propriété de conduction électrique d'un matériau. En fonction de cet état, conducteur ou résistif, l'information stockée est 0 ou 1. L'état du matériau est changé en fonction de la valeur à stocker.

L'avantage de ces technologies non volatiles, outre leur non volatilité, est de ne pas avoir besoin d'une alimentation électrique en continu. La persistance physique permet de maintenir l'information et une impulsion électrique est nécessaire uniquement pour les opérations de lecture et d'écriture. Une cellule mémoire est donc éteinte lorsqu'elle n'est pas utilisée, ce qui réduit considérablement les besoins énergétiques.

La puissance statique de ces mémoires n'est cependant pas entièrement nulle. Si l'on considère une mémoire cache, on distingue dans celle-ci trois parties : le tag, la zone de données et les circuits périphériques. Cette organisation est visible sur la Figure 3.3. Le tag et les données sont respectivement colorés en jaune et gris, et les circuits périphériques sont en dessous, en orange. Dans le cas d'un cache non volatile, la partie tag et la partie données utilisent cette technologie. Les circuits périphériques, minoritaires sur l'ensemble de la surface d'une mémoire cache, nécessitent eux une alimentation permanente et génère tout de même une consommation statique [23]. Plusieurs travaux proposent de réduire cette énergie statique en appliquant des mécanismes de *power-gating* [26, 51, 78, 102], mais cette approche n'est pas considérée par la suite dans nos modèles.

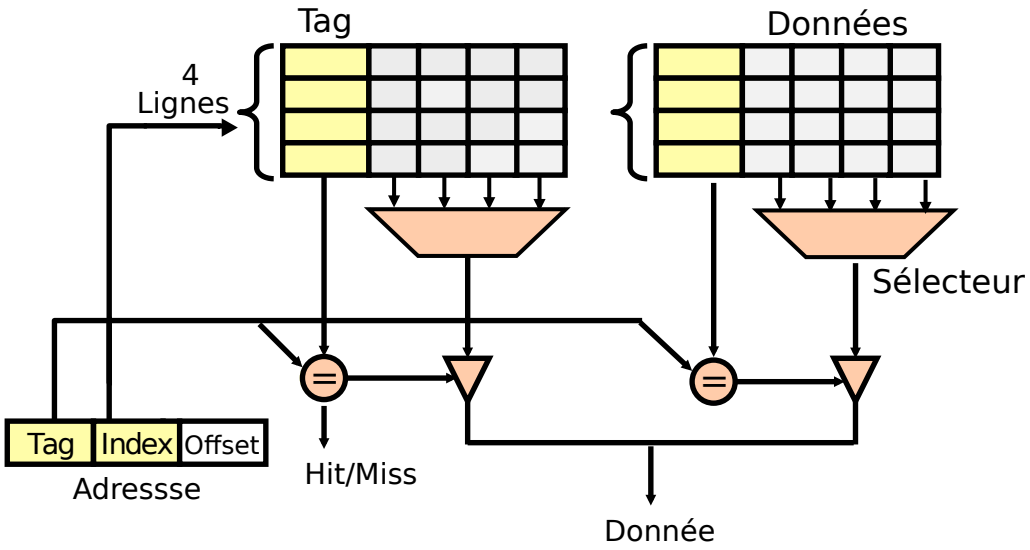


FIGURE 3.3 – Organisation architecturale d'un cache

3.2.2 Familles de mémoires non volatiles

Il existe différentes familles de mémoires non volatiles. Elles se distinguent entre elles par le phénomène physique utilisé pour les cellules mémoires.

Il existe des mémoires qui utilisent un matériau ferroélectrique (FeRAM [93]), la force de résistance (ReRAM [3]), le changement de phase d'un matériau (PC-RAM [33]) ou encore le magnétisme (MRAM [104]). D'autres mémoires non volatiles sont plus répandues dans le monde de l'industrie, comme les NAND/NOR dans les disques dur SSD ou encore la 3D XPoint^{TM} de Micron et Intel [43]. Cette dernière promet des performances 1000 fois supérieures à celles des NAND/NOR. Elle est déjà commercialisée pour les SSD et son intégration en tant que mémoire RAM est prévue pour 2019. Malheureusement, son caractère industriel ne nous permet pas de l'étudier pour nos travaux.

Comparé à la technologie SRAM, les mémoires non volatiles se caractérisent par une puissance statique très faible et une densité supérieure. Elles souffrent cependant de temps d'accès plus longs, et le coût unitaire d'une lecture et d'une écriture est plus élevé. Ce dernier varie en fonction de la NVM considérée, mais il reste supérieur à celui de la SRAM. De plus, le temps de vie d'une cellule NVM est inférieur à celui de la SRAM. En effet, le matériau utilisé comme vecteur de sauvegarde se dégrade plus rapidement qu'une cellule SRAM simplement composée de transistors.

Ces différents aspects sont reportés dans le tableau 3.1 à la fin de la section. Il contient un ensemble de caractéristiques de différentes NVM que l'on trouve dans la littérature.

Du fait de leurs propriétés différentes, ces technologies ne sont pas utilisées dans la même façon. Les mémoires NAND et NOR sont utilisées majoritairement dans les disques

durs de type SSD. En effet, leur très haute densité permet de stocker une grande quantité d'information dans une faible surface. Du fait de leur faible endurance, ces mémoires ne peuvent recevoir qu'un nombre limité de transactions en écriture. Ainsi, il est préférable de les placer le plus loin possible du processeur.

Les ReRAM et les PC-RAM sont généralement étudiées pour remplacer la technologie DRAM utilisée dans les mémoires principales. Elles sont plus dense que celle-ci, et la ReRAM est également aussi rapide voire plus rapide. La PC-RAM souffre quant à elle d'une latence d'écriture supérieure. Bien que supérieure aux mémoires FLASH, l'endurance de ces technologies est limitée (10^9 pour la PC-RAM). Leur intégration sur la puce est donc plus compliquée.

La STT-MRAM est très répandue dans les mémoires caches à la place de la SRAM. La STT-MRAM est moins dense que les autres NVM mais elle reste plus dense que la SRAM. Comme les autres NVM, les latences d'accès sont le principal obstacle de cette technologie.

3.2.3 Technologie privilégiée dans cette thèse : STT-MRAM

Dans la suite de cette thèse, nous considérons uniquement la technologie STT-MRAM pour nos travaux. En effet, sa forte durée de vie comparé aux autres technologies permet de l'utiliser au niveau de la hiérarchie de caches [72]. De plus, bien que ses latences d'accès soient supérieures à la SRAM, la STT-MRAM est la plus rapide parmi les autres technologies non volatiles émergentes. Enfin, c'est une technologie mature considérée avec intérêt par les industriels. De nombreuses entreprises développent actuellement des prototypes de puces contenant cette technologie [42, 65, 76, 79].

3.2.4 Fonctionnement d'une cellule mémoire STT-MRAM

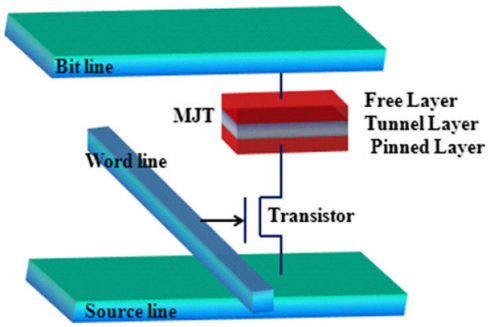


FIGURE 3.4 – Illustration d'une cellule MTJ utilisée dans une STT-MRAM [70]

L'architecture d'une cellule STT-MRAM est donnée par la Figure 3.4. L'élément de base est la *Magnetic Tunnel Junction* (MTJ). Une fine barrière isolante (grise) est placée

entre deux couches ferro-magnétiques (rouge). L'orientation de la *Reference Layer* (RL) définit l'orientation par défaut de la MTJ. On peut changer le magnétisme de la couche *Free Layer* (FL) via l'effet du transfert de spin (*Spin-Transfer Torque*, STT). Quand la couche rouge FL est dans la même direction que la couche rouge RL, alors un courant électrique qui traversera la barrière isolante grise rencontrera peu de résistance. A l'inverse, lorsque les couches rouges sont dans des sens opposés, alors la capacité d'isolation de la barrière grise sera plus forte. Un courant électrique qui la traverse mettra plus de temps et sera moins intense en sortie. C'est de cette manière que l'on différencie le 0 et le 1 avec une mémoire STT-MRAM.

De la même manière que la SRAM, la cellule MTJ est connectée à une *WordLine* et une *BitLine*, mais a seulement besoin d'un transistor. Le reste de la cellule est composé de la MTJ dans laquelle la valeur est stockée. Cette MTJ occupe une plus petite surface que les transistors. La STT-MRAM est donc une technologie plus dense que la SRAM. Pour un même nombre de cellules, la surface de silicium occupée est plus petite, laissant de l'espace sur une puce pour agrandir d'autres composants ou en ajouter de nouveaux. Cette propriété de densité peut également être utilisée pour agrandir la capacité de stockage des caches en ajoutant plus de cellules jusqu'à arriver à une surface équivalent à celle d'un cache SRAM de même capacité.

3.2.5 Questions soulevées par l'intégration de STT-MRAM

L'intégration de la technologie STT-MRAM pose néanmoins plusieurs questions. En effet, cette technologie a des latences d'accès en écriture supérieure à la SRAM. De plus, les coûts énergétiques d'accès, notamment pour les écritures, sont également plus importants. Ainsi, un changement de technologie nécessite de répondre à une ou plusieurs des questions suivantes :

— comment réduire le temps d'accès en écriture ?

— comment réduire le coût d'accès en écriture ?

— comment réduire le nombre d'écritures ?

Nous allons voir dans la suite de ce chapitre différentes approches répondant ces questions. Celles-ci se focalisent à différents niveaux : technologique, circuit, architectural et logiciel.

	SRAM	DRAM	Flash	eDRAM	STT-MRAM	ReRAM	PC-RAM
Volatile	✓	✓	✗	✓	✗	✗	✗
Taille de cellule (F^2)	120-200	6-10	4-6	60-100	6-50	4-10	4-12
Accès (R/W)	Très rapide	Rapide	Très lent	Rapide	Rapide/Lent	Rapide/Lent	Lent/Très lent
Énergie dynamique (R/W)	Faible	Faible	Forte	Moyenne	Faible/Forte	Faible/Forte	Moyenne/Forte
Énergie statique	Élevée	Élevée	Quasi nulle	Moyenne	Quasi nulle	Quasi nulle	Quasi nulle
Endurance	10^{16}	$>10^{15}$	$10^4 \sim 10^5$	10^{16}	$>10^{15}$	10^{11}	$10^8 \sim 10^9$

Tableau 3.1 – Tableau récapitulatif des caractéristiques de certaines mémoires volatiles et non volatiles [12, 71, 72]

3.3 État de l'art de l'intégration de STT-MRAM dans une hiérarchie de caches

L'intégration de mémoires non volatiles émergentes comme la STT-MRAM est majoritairement étudiée pour diminuer la consommation énergétique. Cependant, les latences, les coûts unitaires d'accès et l'endurance limitée forcent les architectes à optimiser les caches pour maintenir un certain niveau de performance. Dans la suite de cette section, nous allons présenter différentes approches utilisant la STT-MRAM et combinant des optimisations pour atténuer les effets mentionnés ci-dessus. Nous ciblons en particulier les travaux portant sur la hiérarchie mémoire. En effet, c'est de cette partie du processeur que provient la majorité de la puissance statique. Nous verrons dans un premier temps les approches technologiques, puis au niveau circuit, au niveau architectural et enfin au niveau logiciel.

3.3.1 Modifications de la cellule MTJ

Il est possible de modifier une cellule MTJ pour varier les latences ou les coûts énergétiques d'accès. Pour cela, on diminue le temps de rétention de la cellule STT-MRAM. Le temps de rétention d'une MTJ caractérise l'estimation de temps durant lequel un *bit-flip* (inversion de bit) à très peu de chance d'avoir lieu. Ce temps de rétention est défini par la stabilité thermique de la MTJ. Une forte stabilité indique que la cellule a peu de chance de produire un *bit-flip* mais en contrepartie la latence et les coûts d'accès en écriture sont plus importants.

En diminuant la stabilité thermique, on peut influencer les latences des opérations sur une mémoire STT-MRAM. La Figure 3.5 illustre ce phénomène. Sur cette figure, les lignes en noir représentent une cellule SRAM. On observe que les latences de la STT-MRAM s'en approchent lorsque le temps de rétention est modifié.

Néanmoins, cette optimisation de la technologie nécessite de trouver un temps de rétention approprié pour avoir un compromis entre la réduction des latences et la perte d'information passé un certain délai. Pour s'assurer de la cohérence des données, il faut mettre en place des solutions de sauvegarde, comme des rafraîchissements automatiques ou des mémoires tampons.

Smullen et al. [99] proposent de modifier la taille de la *Free-Layer* qui est l'élément de stockage d'un bit dans une cellule MTJ. Cela a pour effet de diminuer la quantité d'énergie nécessaire pour une lecture ou une écriture, et par conséquent de diminuer aussi la latence. Ils ont validé leur approche avec deux scenarii : *i*) toute la hiérarchie mémoire (3 niveaux) est composée de STT-MRAM et *ii*) la mémoire cache L1 utilise la SRAM et les

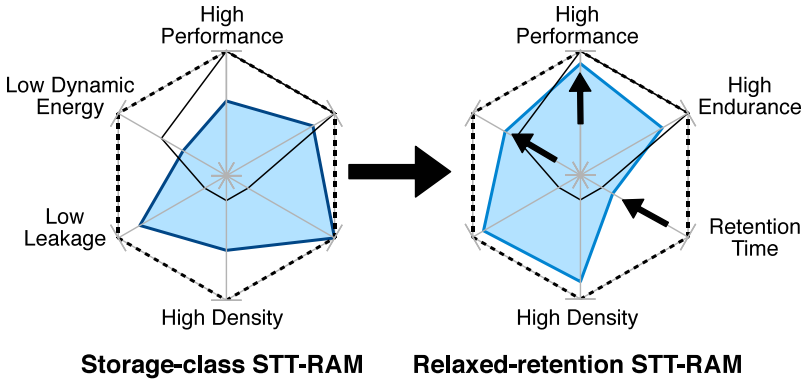


FIGURE 3.5 – Effets de la modification du temps de rétention d'une cellule MTJ. Les lignes noires représentent la SRAM [99].

autres la STT-MRAM . Les résultats montrent que le cas *i)* fournit la meilleure efficacité énergétique (caractérisée par l'EDP) mais des temps d'exécution très élevés du fait de la forte sollicitation de la mémoire cache L1-D par le processeur. Malgré une mémoire cache L1-D optimisé pour les écritures, la SRAM reste plus rapide. Dans le cas *ii)*, la performance est très proche de la SRAM tandis que l'efficacité énergétique est améliorée de manière significative.

Pour influencer le temps de rétention, Jog et al. [50] diminuent la surface de la *Free-Layer* mais également son épaisseur. De plus, les auteurs proposent de diminuer l'aimantation de la couche magnétique pour accentuer la réduction du temps de rétention. Afin de trouver le temps de rétention adéquat, Jog et al. mènent une analyse centrée sur les applications. Les écritures des benchmark PARSEC et SPEC CPU2006 sont monitorées sur différentes niveaux de caches pour établir le temps moyen entre deux utilisations d'une même donnée. Ce délai indique le temps de rétention minimal que la cellule MTJ doit fournir. Pour assurer que les données ne seront pas perdues si le temps de rétention est dépassé, les auteurs proposent d'ajouter un compteur de 2^n bits à chaque bloc de la mémoire cache cache. Ce compteur caractérise l'état d'utilisation du bloc. Une valeur de 0 indique que le bloc vient d'être amené dans le cache, une valeur de $2^n - 1$ indique que le bloc va bientôt être perdu. Ce compteur est incrémenté automatiquement selon une période définie. Lorsque la valeur maximale est atteinte, une transaction d'écriture de ce bloc vers une mémoire temporaire est envoyée afin de sauvegarder la valeur. Les résultats expérimentaux montrent une forte diminution de la consommation énergétique et une amélioration des performances de l'ordre de 10%.

Sun et al. [101] proposent un design pour les caches où les différents *ways* de la mé-

moire cache ont différentes périodes de rétention. Par exemple, sur un cache 16 associatif, le *way* 0 est optimisé pour les latences avec un faible temps de rétention, tandis que les 15 autres *ways* ont des temps de rétention plus élevés. Ils utilisent ensuite des compteurs matériels pour détecter si un bloc est plus souvent accédé en écriture qu'en lecture. Les blocs dominés par les écritures sont alloués dans le *way* 0. Pour éviter la perte de données dans le *way* 0, un mécanisme de migration est mis en place pour déplacer les blocs sur d'autres lignes avec de plus longues périodes de rétention. Dans le cas d'un cache bas niveau comme le L1-D, les cellules sont optimisées de manière à diminuer la latence d'écriture et l'énergie. Le temps de rétention est alors de quelques micro-secondes. Un mécanisme de rafraîchissement basé sur des compteurs matériels est mis en place et seuls les blocs en fin de vie sont rafraichis pour éviter la perte de données.

Considérant un cache avec des temps de rétention différents en fonction des lignes, Bouziane et al. [15] proposent de calculer le *Partial Worst-Case Execution Time* (pWCET) entre l'écriture d'une donnée et la prochaine lecture de celle-ci. Cela permet de déterminer quel est le temps minimal pour lequel le cache doit conserver une donnée avant qu'elle ne soit ré-utilisée. Ce calcul est effectué à la compilation de manière statique. Avec cette information, il est possible de guider le placement des données dans les lignes de la mémoire cache en fonction des temps de rétention.

Remarques

Les approches présentées ici nécessitent une modification profonde de la technologie. Or, les informations disponibles sur la maturité de la STT-MRAM chez les industriels sont très difficiles d'accès. Les travaux sont alors basés sur des modèles et des extrapolations dont on peut remettre en question la justesse.

La modification de la cellule MTJ se fait au prix d'une période de rétention plus faible, voire extrêmement faible (quelques ns). En cas de défaillance du mécanisme de contrôle de rétention, il y aurait une inconsistance dans les données manipulées et de potentielles erreurs dans les calculs.

3.3.2 Optimisations au niveau circuit

Zhou et al. [121] ont développé une méthode appelé *Early Write Termination* (EWT) pour diminuer le nombre de bits redondant sur les cellules STT-MRAM. On entend par redondant le fait d'écrire une valeur qui est déjà présente. Lorsqu'une opération d'écriture est effectuée sur une MTJ, la résistance de la cellule ne change pas au fur et à mesure du temps. Au contraire, elle s'inverse brutalement à la fin de l'opération. Avant ce changement, la cellule contient donc la valeur initiale avant écriture. Cette valeur est alors lue

Pattern 1	00000000000000000000000000000000
Pattern 2	00000000000000000000000000000001
Pattern 3	00000000000000000000000000000010
Pattern 4	00000000000000000000000000000100
...	
...	
Pattern 31	00100000000000000000000000000000
Pattern 32	01000000000000000000000000000000
Pattern 33	10000000000000000000000000000000

Tableau 3.2 – Système d'encodage implémenté par Yazdanshenas et al.

en parallèle de l'écriture, et s'il s'avère qu'elle est identique à celle qui est en train d'être écrite, alors la transaction est interrompue.

Yazdanshenas et al. [118] proposent une optimisation au niveau du circuit pour réduire la probabilité d'une inversion de bit dans une cellule MTJ. Pour cela, ils réduisent la probabilité pour les cellules mémoires de contenir la valeur 1, qui implique un état de résistance fort se dégradant au fur et à mesure du temps. Ils ont étudié l'évolution des données écrites dans le cache pour les applications de la suite PARSEC et ont remarqué que, en considérant des blocs de 4 octets, il existe 33 valeurs qui sont majoritairement utilisées. Ainsi, chacune de ses valeurs est encodée dans un format particulier limitant le nombre de 1 en utilisant une distance de Hamming de 2 au maximum entre deux nombres. Ce système d'encodage est décrit par le tableau 3.2. Ainsi, pour chaque nouvelle écriture, il existe une forte probabilité que seuls 2 bits soient écrits sur les 32 qui composent le bloc. Pour empêcher les écritures inutiles, cette approche est combinée avec l'approche EWT de Zhou et al..

Remarques

Ces propositions impliquent de monitorer l'état des cellules MTJ ou d'encoder ou de décoder chaque octet lu ou écrit. Bien que le surcoût matériel soit faible, il faut modifier les cellules MTJ pour ajouter les contrôleurs de lecture et d'écriture. De la même manière que les approches technologiques, la modification des cellules MTJ est une tâche complexe nécessitant un accès privilégié à des informations industrielles afin de valider l'approche expérimentalement. Enfin, ces techniques nécessitent de descendre au niveau du circuit de la mémoire cache, ce qui n'est pas l'approche que nous considérons pour nos travaux.

3.3.3 Caches hybrides

Un cache hybride est un cache utilisant deux technologies de mémoire. Cette approche est possible avec certaines NVM qui sont compatibles CMOS. Le principe est le suivant. Les données sont caractérisées en fonction de leur taux de lecture et d'écriture. Soit une donnée est plus souvent lue, soit elle est plus souvent écrite. Ensuite, les auteurs se servent de cette information pour placer la donnée dans la partie SRAM ou la partie NVM de la mémoire cache. Si une donnée est plus fréquemment lue, alors on la placera dans la partie NVM. En effet, le temps de lecture est sensiblement le même que la SRAM. En revanche, si une donnée est très fréquemment écrite, alors elle sera placée dans la partie SRAM de la mémoire cache. Cela limitera les latences en écriture et leur coût énergétique.

Li et al. [63] ont proposé une approche appelée SPD (*software dispatch*). Cette technique repose sur le compilateur est le système d'exploitation pour capturer le schéma d'accès aux données. En se basant sur cette détection, le système d'exploitation peut guider le cache pour placer les données. Celles-ci sont distribuées sur les parties SRAM et NVM selon le schéma évoqué précédemment pour diminuer le temps d'exécution et la consommation énergétique. Cette proposition est illustrée par la Figure 3.6.

Lors de la compilation, une passe de détection des accès mémoire sous forme de tableaux est effectuée. Le compilateur est également utilisé pour détecter les structures de données chainées via l'utilisation de graphes. Tous ces accès à la mémoire sont analysés et une instruction est insérée lors de la compilation avant ces accès pour guider le placement dans l'une ou l'autre partie de la mémoire cache. Lors de l'évaluation, Li et al. ont montré que leur approche diminue le temps d'exécution de 5% et la consommation énergétique de 10%.

Bien que ces résultats soient intéressants, cette proposition nécessite une mise en place assez coûteuse au-dessus du matériel. Le compilateur doit être modifié pour analyser le code source, impliquant de recompiler les applications pour bénéficier de cette technique. De plus, cela implique que le compilateur doit avoir la connaissance de la technologie de cache qui est dans le processeur. Cela brise la couche d'abstraction qui existe entre le compilateur et la technologie matérielle sous-jacente. L'allocateur mémoire du système d'exploitation doit également être modifié, et une nouvelle instruction CPU ajoutée. Enfin, cette approche est une approche dite statique. Elle repose sur une analyse du code source avant l'exécution du programme, et aucune nouvelle décision ne peut être prise durant l'exécution. Cela empêche toute adaptabilité du matériel quant au comportement de l'application.

Jadidi et al. [45] proposent une approche à la fois dynamique et implémentée entiè-

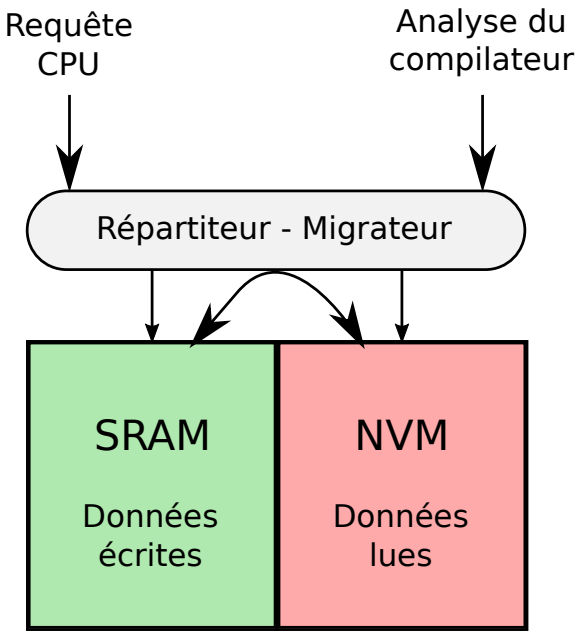


FIGURE 3.6 – Utilisation d'un cache hybride avec un gestionnaire de contrôle logiciel

rement grâce au matériel. Dans leur étude, le cache est composé à 75% de STT-MRAM et 25% de SRAM, réparti de cette manière pour chaque *set* de la mémoire cache.

Deux méthodes complémentaires pour la migration de données sont proposées. Celles-ci sont illustrées sur la Figure 3.7. La première méthode, appelée *intra-set*, migre les données entre les lignes d'un même set. Pour savoir quand migrer les données, à chaque ligne est associé un compteur de saturation qui est incrémenté à chaque écriture. Lorsque le compteur d'une ligne STT-MRAM atteint son maximum, cela signifie que cette ligne a été beaucoup écrite et doit être déplacée sur la partie SRAM du set. Les compteurs d'écriture des blocs SRAM servent à déterminer quelle sera la ligne qui sera déplacée dans la partie STT-MRAM. On choisira une ligne ayant une valeur de compteur inférieure à une certaine limite pour ne pas déplacer une ligne SRAM qui est souvent écrite. Cette méthode est également utilisée entre les lignes STT-MRAM pour distribuer les écritures et améliorer l'endurance des cellules mémoire.

La seconde méthode, *inter-set*, propose d'échanger entièrement deux sets de la mémoire cache. Pour cela, de nouveaux compteurs de saturation sont utilisés sur chaque set pour mesurer leur activité d'écriture. En suivant le même principe que la méthode *inter-set*, les sets sont interchangées.

La proposition de Jadidi et al. répond en partie aux problèmes soulevés précédemment. Elle est entièrement matérielle et donc indépendante du système et du compilateur.

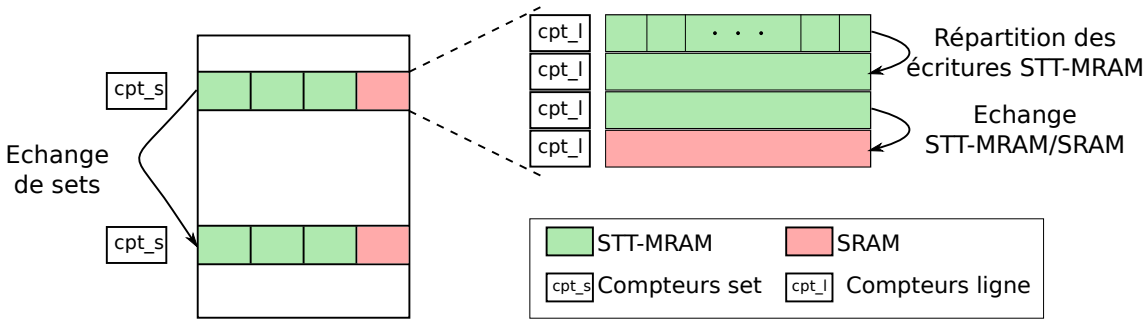


FIGURE 3.7 – Modifications architecturales de la mémoire cache proposées par Jadidi et al. [45]. Chaque ligne est associée à des compteurs, chaque *set* est associé d'autres compteurs.

De plus, cette approche est dynamique car les décisions de migration sont prises après une phase d'analyse des écritures pendant l'exécution. Néanmoins, cela nécessite une profonde modification de la mémoire cache. Il faut ajouter des compteurs à chaque ligne en veillant à la technologie utilisée. En effet, les compteurs des lignes SRAM et STT-MRAM ne sont pas les même. Il faut en plus ajouter des compteurs sur chaque set de la mémoire cache. Ces derniers doivent être consultés à chaque opération avant de prendre une décision de migration. Enfin, la politique de déplacement entre les sets nécessite d'ajouter un mécanisme de sauvegarde temporaire du set d'origine et d'implémenter une nouvelle politique de recherche de données lors des accès. Si la cache reçoit une lecture sur une donnée qui a été déplacée, cette politique doit être capable de retrouver une donnée dans son nouveau set d'accueil.

Plutôt que d'utiliser un cache hybride nécessitant un mécanisme de migration de données, Senni et al. [97] proposent d'utiliser la technologie SRAM pour la partie tag de la mémoire cache et la technologie STT-MRAM pour la partie données. La partie tag de la mémoire cache sert à détecter si l'accès est un *hit* ou un *miss*. Utiliser de la SRAM ici permet, en cas de *miss*, de rapidement transférer la requête au niveau de mémoire suivant et d'éviter quelques cycles de latence supplémentaires. De plus, le tag de la mémoire cache contient très peu de données et représente en moyenne 10% de la surface totale. Malgré son courant de fuite élevé, l'utilisation de SRAM sur une faible surface limite l'augmentation de la consommation. La partie données utilise la technologie STT-MRAM pour diminuer drastiquement la consommation statique. Les résultats en matière de consommation énergétique montre que le cache hybride augmente d'environ 48% la consommation d'un cache entièrement en STT-MRAM. Néanmoins, cette proposition réduit de 89% la consommation énergétique comparé au scénario « full SRAM ».

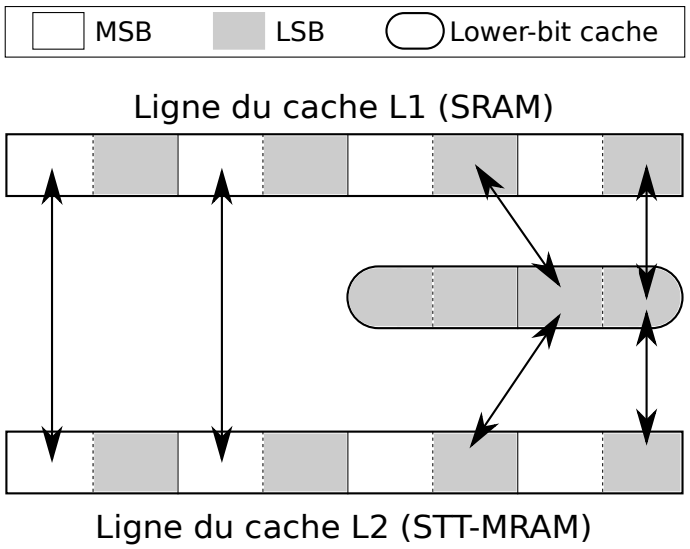


FIGURE 3.8 – Fonctionnement de la mémoire cache intermédiaire proposé par Ahn and Choi [2]

Remarques

D'une manière générale, les approches qui utilisent un mécanisme de migration sont coûteuses en matériel et complexes à mettre en place. De plus, les méthodes dynamiques dépendent de l'efficacité à prédire les migrations de données. En cas de mauvaise décision, des écritures supplémentaires seront nécessaires pour ramener les lignes de cache à leur emplacement d'origine, ce qui est coûteux en temps et en énergie. L'approche hybride tag/données proposée par Senni et al. est intéressante mais aucune optimisation sur la gestion des écritures n'est proposée pour la partie données. Les écritures restent toujours pénalisantes en termes d'énergie et de latence.

3.3.4 Modification de l'architecture

Ahn and Choi [2] basent leur travaux sur l'observation que dans une architecture de cache, les bits de poids fort sont modifiés moins souvent que les bits de poids faible. En effet, les valeurs des variables d'un programme changent dans un petit intervalle.

Ainsi, un cache intermédiaire entre le L1 et le L2 est ajouté (*lower-bit cache*). Son fonctionnement est illustré sur la Figure 3.8. Ce cache contient les bits de poids faible (LSB) de tous les mots stockés dans le cache L1. En cas d'écriture du L1 vers le L2, les bits de poids fort (MSB) sont écrits dans le cache L2 uniquement s'ils sont modifiés. Les bits de poids faible sont eux écrits dans le cache intermédiaire. En cas de lecture sur le cache L2 depuis le L1, les bits de poids faible du petit cache sont combinés au bits de poids fort du L2. En cas d'évincement dans le cache L2, si le cache intermédiaire contient les bits de

poids faible du bloc à évincer, la valeur qui est envoyée au niveau de cache supérieur est une combinaison des bits de poids fort du L2 et des bits de poids faible de la mémoire cache intermédiaire. Enfin, le *lower-bit cache* est plus petit que le L2 pour minimiser l'énergie statique ajoutée. Lors d'un évincement sur ce cache, les bits de poids faible sont écrits dans le bloc correspondant dans le L2.

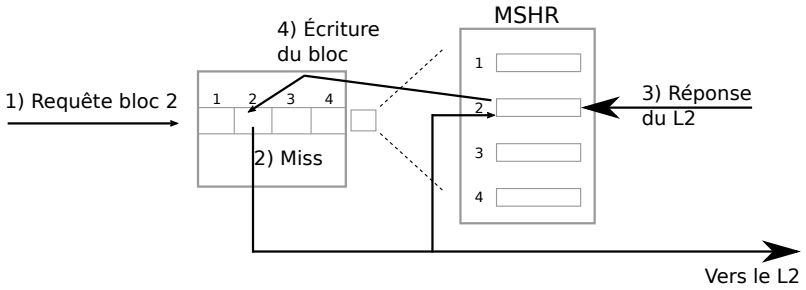
Cette technique permet de diminuer l'énergie dans deux cas. Premièrement, lorsque seuls les bits de poids faible sont écrits, aucune écriture n'a lieu dans le cache L2. Deuxièmement, si le cache intermédiaire peut servir toutes les requêtes sur un bloc avant son évincement du L2, alors aucune écriture n'aura eu lieu sur le L2. Cette technique permet d'économiser 25% d'énergie au niveau de la mémoire cache L2 comparé à un cache classique utilisant de la STT-MRAM.

Komalan et al. [55] proposent d'intégrer la STT-MRAM au niveau de la mémoire cache L1 d'instruction d'un processeur de type ARM. En effet, ce cache est en lecture seule et la seule source d'écriture provient du L2 en cas de *miss*. Dans une configuration classique, les registres MSHR (Miss Status Holding Registers) de la mémoire cache L1 contiennent des informations sur les requêtes *miss* en cours de résolution pour éviter les doublons. Lorsque le cache L2 répond à une requête, le MSHR de la mémoire cache L1 est consulté pour savoir à quel bloc correspond la réponse, puis le bloc est écrit et la donnée est envoyée au CPU. Ce principe est décrit par la Figure 3.9a.

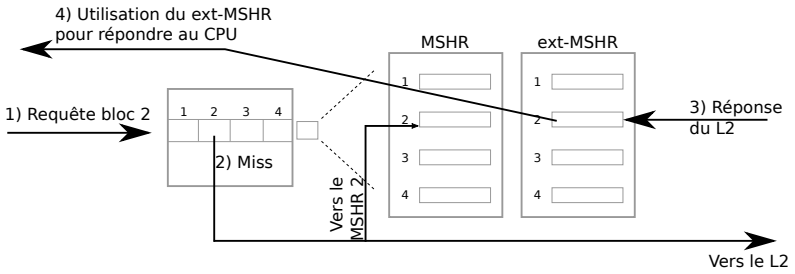
Pour diminuer le nombre d'écritures sur le cache, Komalan et al. [55] proposent d'étendre les MSHR avec des cellules SRAM pour contenir toute la requête lors de la réponse du niveau L2 (voir Figure 3.9b). La réponse du L2 est dans un premier temps uniquement contenue dans les registres MSHR étendus et est utilisée lors des accès. Le fait de garder la requête dans le MSHR étendu évite une écriture dans le cache L1. A partir d'un certain nombre d'accès au MSHR étendu, l'entrée est libérée et la requête est finalement écrite dans le cache L1. Cette technique permet de réduire la pénalité de performance de la STT-MRAM à 1% en moyenne tout en diminuant la consommation énergétique de 35%.

Sun et al. [100] proposent de modifier le *write buffer* du niveau L2 pour changer les priorités entre lectures et écritures. L'idée est la suivante : si le write buffer n'est pas vide et qu'une lecture est en cours, la ou les écritures du write buffer doivent attendre la fin de la lecture et de toutes les lectures suivantes.[1] De plus, si une écriture est en cours mais que celle-ci est commencée depuis moins de α cycles avec $\alpha = \frac{latenced'ecriture}{2}$, alors l'écriture est préemptée, remise dans le write buffer et la lecture s'exécute. L'écriture sera faite ultérieurement. Sun et al. ont également augmenté la capacité de la mémoire cache

1. Un mécanisme de contrôle assure que les lectures ne se font pas sur les données à écrire.



(a) Fonctionnement classique des MSHR



(b) Extension des MSHR

FIGURE 3.9 – Fonctionnement des MSHR et proposition d'extension pour limiter le nombre d'écritures

grâce à la densité de la NVM utilisée, ici la STT-MRAM. L'effet combiné de la densité et du write buffer donne une amélioration de performance de 10% et une réduction énergétique de 67%

Cette approche montre de très bon résultats et une exploitation intéressante de l'asymétrie des NVM. Cependant, elle augmente le nombre d'écritures à cause de son système de préemption, et donc l'énergie dynamique de la hiérarchie mémoire. Les auteurs ont proposé de combiner leur technique de préemption avec une technique de cache hybride. Les performances sont alors améliorées de 4.9%, ce qui est moins efficace que précédemment.

Rasquinha et al. [91] proposent une technique appelée *Write Biasing*. Le but est de maintenir les lignes *dirty* d'un cache SRAM le plus longtemps possible à l'intérieur de ce cache. Les lignes *clean* sont donc évincées en priorité. Chaque ligne *dirty* évincée nécessite une écriture dans le niveau de cache supérieur pour sauvegarder la nouvelle valeur. Ce n'est pas le cas des lignes *clean*. Le nombre d'évincement nécessitant une écriture est donc réduit, ce qui diminue le nombre d'écritures sur un niveau de cache supérieur. Cette technique est appliquée à un niveau de cache inférieur à celui où est intégrée la technologie STT-MRAM. Pour valider cette proposition, les auteurs ont modifié les politiques d'insertion et de promotion de la LRU. Les accès en écriture sont promus en position 0 (position la plus récente) et les accès en lecture sont promus à une position K. Les résultats expérimentaux montrent de léger gains en énergie dynamique, plus visibles en cas d'écriture. Les auteurs ne donnent aucun chiffre sur le temps d'exécution des applications. La Figure 3.10 illustre la proposition de Rasquinha et al.. La ligne de cache originelle contient A, B, C et D, puis est accédée par différentes requêtes. La gauche de la Figure 3.10 montre l'évolution de la ligne de cache avec la politique *Write Biasing*, la droite montre l'évolution avec la politique LRU. On voit que la politique LRU doit évincer une donnée *dirty* alors que la stratégie *Write Biasing* a réussi à l'éviter.

Remarques

Ces travaux présentent des résultats prometteurs pour les technologies non volatiles, et particulièrement la STT-MRAM. Elles sont par ailleurs orthogonales à nos propositions. Cependant, les approches développées se concentrent sur un design de cache en particulier. Les auteurs ne proposent pas d'étude sur la NVM et sa configuration avant son intégration. Or, nous verrons par la suite qu'il existe un large espace d'exploration lors de la conception d'un cache et que certains choix peuvent influencer les caractéristiques de latence ou d'énergie.

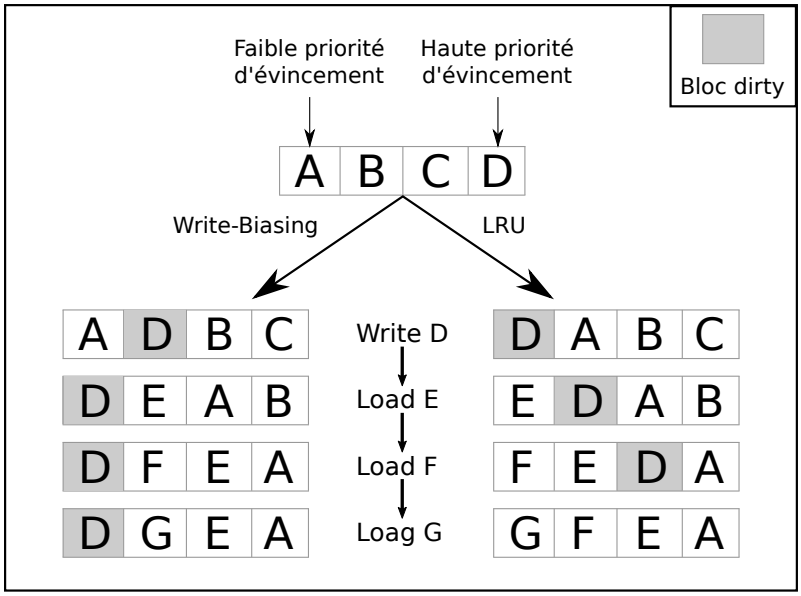


FIGURE 3.10 – Fonctionnement du *Write Biasing*. La politique LRU génère une écriture d'un block *Dirty* vers le niveau de cache supérieur, ce qui n'est pas le cas de la politique *Write Biasing*.

3.3.5 Approches logicielles

On appelle *silent store* [59, 85] l'écriture d'une valeur dans une cellule mémoire contenant déjà cette valeur. D'un point de vue logiciel, ces écritures sont inutiles au calcul et peuvent être supprimées. Bouziane et al. [14] proposent une modification du compilateur pour détecter ces écritures et les supprimer. Dans un premier temps, une analyse statique des instructions d'écriture est effectuée pendant la compilation. Cette phase sert à détecter les écritures ayant une forte probabilité d'être des *silent store*. Une optimisation est alors appliquée sur ce sous-ensemble d'instructions. Chaque instruction est remplacée par un nouveau bloc d'instructions, comme illustré par le tableau 3.3. Ce bloc comprend *i*) une lecture de la donnée présente dans la cellule mémoire *ii*) une comparaison avec la valeur que l'on souhaite écrire *iii*) un branchement permettant de décider si l'écriture doit se faire. Les auteurs ont validé leur implémentation sur le jeu d'instruction ARMv7. Pour diminuer le coût des instructions supplémentaires ajoutées, ils proposent d'utiliser une instruction ARM spécifique (`strne`) pour exécuter en une seule instruction le branchement et l'écriture.

Hu et al. [40] proposent de remplacer les caches par des mémoires de type *scratch-pad* (SPM). Ces mémoires sont entièrement gérées de manière logicielle par le programmeur ou le compilateur (ou les deux). Dans le cas choisi par Hu et al., les SPM sont hybrides et

	Avant optimisation (1)	Après optimisation (2)	Après optimisation + instruction spécifique (3)
Code	store @x = val	load y = @x cmp val, y beq next store @x, val next :	load y = @x cmp val, y strne @x, val

Tableau 3.3 – Élimination des *silent store*

contiennent de ces cellules SRAM et STT-MRAM. Lors de la compilation, les applications sont découpées en plusieurs régions délimitées par *i*) l'entrée dans une fonction et *ii*) l'entrée dans un boucle. Avant l'exécution de chaque région, une analyse de code est effectuée pour générer le schéma d'allocation des données dans les parties SRAM ou STT-MRAM. Ce schéma reste le même une fois généré.

Afin d'améliorer les performances de la mémoire cache de premier niveau, Li et al. [64] proposent un cache avec uniquement des cellules de type STT-MRAM comprenant deux modes de fonctionnement : FB pour *Fast read Block*, et SB pour *Standard Block*. Pour cela, le cache est ré-organisé de la manière suivante (voir Figure 3.11). Par groupe de deux, chaque ligne est fusionnée avec sa voisine pour créer un ligne appelée « ligne FB ». Cette ligne FB contient donc deux lignes de caches classiques, qui sont elles appelées « lignes SB ». La ligne FB est accessible très rapidement, tandis que les lignes SB sont plus lentes. Un bit de contrôle M est ajouté à la ligne FB pour préciser quel est le mode d'accès.

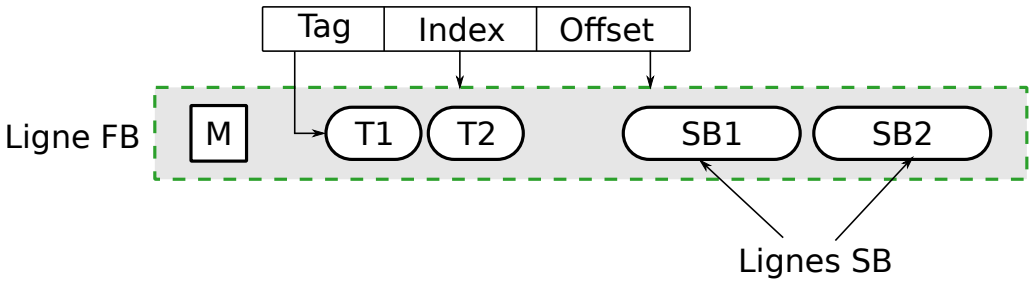


FIGURE 3.11 – Proposition de cache adaptatif de Li et al.

Lorsqu'un accès a lieu sur une donnée, on peut soit utiliser le mode FB et la donnée est lue ou écrite rapidement, soit utiliser le mode SB qui correspond au mode d'accès classique. Le changement de mode des lignes est adapté à l'application. Les besoins sont déterminés lors de la compilation, et l'information est passée par le compilateur à un *run-*

time qui ajoutent des instructions spécifiques dans le code pour changer le mode d'accès des lignes de la mémoire cache.

Remarques

Les approches logicielles utilisant le compilateur ont l'avantage d'être facilement portables entre différents jeux d'instructions. Cependant, elles nécessitent d'avoir accès au code source des applications, ce qui n'est pas toujours garanti. De plus, il est nécessaire de recompiler l'application, ce qui implique de possibles changements dans les dépendances de données ou le flot d'exécution du programme. Ces changements sont hautement indésirables pour des applications à fortes contraintes comme l'avionique au l'automobile. Enfin, le bon fonctionnement de ces optimisations peut nécessiter la désactivation de certaines optimisations de compilation habituellement utilisées pour améliorer la performance.

3.4 Environnements de simulation et de validation de technologies non volatiles émergentes

3.4.1 Simulateurs de mémoires non volatile

Cette section présente quelques simulateurs de technologies mémoires classiques et non volatiles. Ils sont résumés dans le tableau 3.4.

CACTI [112] est un simulateur de mémoires volatiles de type SRAM ou DRAM. Son développement par HP Labs a commencé il y a plus de 20 ans et il est aujourd'hui toujours maintenu, chaque version contenant de nouvelles fonctionnalités et des corrections de bugs. CACTI permet d'évaluer les temps d'accès, la consommation énergétique et la surface pour des caches ou des mémoires principales. Les modèles sont très flexibles et beaucoup de paramètres peuvent être changés, du voltage de la cellule à l'organisation architecturale. Les modèles montrent un taux d'erreur de 6% par rapport à des modèles HSPICE [2] utilisés dans l'industrie. Il existe une version étendue, **CACTI-NVM** [117] [3], pour la STT-MRAM. Cette version de CACTI n'est pas en accès libre.

NVSim [29] est un outil d'évaluation de performance de mémoires non volatiles. De la même manière que CACTI, NVSim fournit des informations sur les latences, la surface et les coûts énergétiques d'accès. En plus des mémoires non volatiles, NVSim permet de

2. https://www.synopsys.com/verification/ams-verification/circuit-simulation/hspice.html
3. Cette version de CACTI n'a pas été nommée par les auteurs. CACTI-NVM est un nom que nous avons choisi arbitrairement

simuler les technologies SRAM et DRAM. Cet outil a été validé avec plusieurs prototypes industriels et souffre au maximum d'une marge d'erreur de 15% [29]. Les mémoires non volatiles que peut simuler NVSim sont la STT-MRAM, la PC-RAM, la ReRAM et la NAND.

La configuration de NVSim se fait via deux fichiers d'entrée. Le premier définit les caractéristiques techniques d'une cellule de la technologie considérée (SRAM, STT-MRAM, PC-RAM etc.) comme la densité d'une cellule, les temps de *pulse* ou le voltage. Le second définit l'architecture de la mémoire : cache ou RAM, taille, technologie de gravure, organisation interne, etc.

En sortie, NVSim produit différentes informations comme les latences d'accès, le détail de ces latences (routage, décodage, amplificateur, etc), et fait de même pour les coûts énergétiques. On a également la bande passante ou encore la surface de chaque composant de la mémoire cache ainsi que la surface totale. Les modèles de NVSim pour la STT-MRAM ont été validés avec un taux d'erreur de 2.7% pour la surface et 4.3% pour la latence [29]. Il existe une extension de NVSim pour les NVM utilisant une technologie en trois dimensions : **Destiny** [87].

Il existe également d'autres simulateurs de mémoires non volatiles mais orientés pour les mémoires principales, parmi lesquels DRAMSim2 [92], NVMain [86], Ramulator [54], et plus récemment HME [30].

Simulateur	Niveau de détail	Facilité d'utilisation	Intégration	Mémoire non volatile	Licence libre
CACTI	Circuit	+++	+++	✗	✓
CACTI-NVM	Circuit	+++	+++	✓	✗
NVSim	Circuit	+++	+++	✓	✓
HSPICE	Porte	+	+	✓	✓

Tableau 3.4 – Différents simulateurs de technologies mémoires et leurs caractéristiques

3.4.2 Modélisation des mémoires non volatiles

Nous allons appliquer le principe de la simulation logicielle définie au chapitre 2 pour modéliser des mémoires non volatiles. La simulation de composants comprenant de la STT-MRAM se fera en modifiant uniquement les latences d'accès et en s'abstrayant des autres caractéristiques de la STT-MRAM. La propriété de non volatilité n'est pas modélisée (i.e., il n'y a pas de « on/off » sur les cellules), on considère que les cellules ont une endurance similaire à la SRAM et les phénomènes d'erreurs de lecture ou d'inversion de bits sont également négligés.

3.5 Bilan

Dans ce chapitre, nous avons présenté le principe des mémoires non volatiles, et en particulier la STT-MRAM. Nous avons vu que cette technologie possède des caractéristiques différentes de la SRAM qui soulèvent des questions quant à son intégration dans une hiérarchie de caches. En effet, malgré une diminution intrinsèque du courant de fuite, l'asymétrie en matière de latence et d'énergie et les coûts élevés en écriture sont un frein à son adoption. Pour répondre à ce problème, nous avons présenté plusieurs niveaux sur lesquels il est possible d'agir.

Modifications et utilisation des propriétés des cellules MTJ

La modification de la cellule MTJ permet d'atténuer certains désavantages de la STT-MRAM. En jouant sur la taille ou l'épaisseur de la couche magnétique, on peut diminuer la latence et l'énergie nécessaire aux transactions. Cependant, les mécanismes de contrôle ajoutés pour limiter la perte d'informations nécessitent de migrer des données. Cela génère un coût supplémentaire en matière d'énergie, de latence, et affectent de manière négative l'endurance des cellules.

De plus, notre approche vise à évaluer l'impact des STT-MRAM et à trouver une solution au niveau architectural et non technologique. Nous sommes utilisateurs de cette nouvelle technologie et nous ne cherchons pas à l'améliorer mais à trouver comment l'intégrer dans une hiérarchie mémoire.

Gestion de la mémoire par le compilateur

Les approches logicielles pour gérer la technologie STT-MRAM se basent sur la modification du compilateur. Ce dernier est utilisé pour analyser le comportement des applications et détecter les données majoritairement lues ou écrites. Le placement des données est alors guidé par le compilateur durant l'exécution par l'insertion d'instructions spécifiques. Bien que portable sur toutes les architectures supportées par le compilateur, cela nécessite d'ajouter des instructions spécifiques dans le processeur pour le placement des données. Enfin, cette approche s'affranchit de la couche d'abstraction qui existe entre le compilateur et la technologie sous-jacente. En effet, ces approches nécessitent de la part du compilateur la connaissance de la technologie mémoire utilisée pour les caches.

Approches architecturales

Les approches architecturales présentées en section 3.3 ne proposent aucune exploration architecturale au niveau de la construction de la mémoire cache. Les auteurs proposent d'utiliser une technologie non volatile et se servent d'un simulateur de NVM, en

général NVSim, pour extraire des latences d'accès et des valeurs énergétiques. Aucun détail n'est donné sur la façon d'obtenir ces valeurs et sur les possibilités de design.

Or, il existe un espace d'exploration conséquent lors de la conception d'une mémoire cache. Beaucoup de paramètres architecturaux existent avec une influence certaine sur les caractéristiques de la mémoire cache. Afin d'exploiter au mieux une optimisation architecturale, il convient de sélectionner une configuration la plus pertinente selon des critères définis par le concepteur.

Dans cette thèse, nous proposons une méthode simple permettant de visualiser un ensemble de configurations de mémoires non volatiles et d'extraire la meilleure configuration selon des caractéristiques précises. Les discriminants sont à la charge de l'architecture, ce qui permet d'adapter cette méthode aux différentes contraintes d'intégration.

Validation expérimentale

Que cela soit pour la simulation de mémoires, d'architecture ou de l'évaluation de consommation énergétique, l'écosystème des outils présenté aux sections 2.3 et 3.4 est large et propose une multitude de possibilités pour répondre aux besoins.

Néanmoins, ces outils sont indépendants et aucune communication n'existe entre eux. De fait, l'évaluation de propositions de conception pour des architectures utilisant des mémoires non volatiles nécessite des étapes manuelles pour *i*) la configuration caches NVM *ii*) l'intégration des ces caches dans un simulateur *iii*) la configuration de la plateforme à simuler et *iv*) le post-traitement pour une évaluation énergétique.

Ces différentes étapes empêchent le passage à l'échelle des explorations architecturales par la quantité de travail nécessaire en amont. Le chapitre suivant sera consacré à la définition d'un cadre d'exploration architecturale passant à l'échelle. On présentera ensuite deux implémentations automatiques et semi-automatiques respectant ce schéma et combinant un simulateur de mémoires non volatiles, un simulateur d'architecture et un framework d'évaluation de consommation énergétique.

Why so serious?

JOKER - THE DARK KNIGHT (2008)

Chapitre 4

Définition et implémentations d'un cadre d'exploration architecturale

Dans ce chapitre, nous présentons un schéma de principe du cadre d'exploration nécessaire à nos travaux. Ensuite, nous illustrons ce schéma avec deux implémentations réalisées à travers différents simulateurs d'architectures et modèles énergétiques.

4.1 Cadre générique d'exploration

Pour nos travaux, nous avons choisi le niveau d'abstraction logiciel. C'est la solution la plus simple à mettre en place si l'on veut évaluer l'effet de mémoires émergentes sur une hiérarchie de caches. Elle ne nécessite pas de descendre au niveau du circuit. De plus, cette thèse ne propose pas d'optimisation à un tel niveau. On se contente d'être utilisateur de ces technologies en se plaçant d'un point de vue architectural.

Le modèle générique présenté dans cette section est une version étendue d'un modèle de précédents travaux au sein de l'équipe du laboratoire [95].

4.1.1 Simulateur d'architecture

La pièce centrale de nos travaux est le simulateur d'architecture. Nous avons besoin de modèles de cœurs avec différentes caractéristiques de performance pour explorer plusieurs types d'architectures. L'outil doit aussi modéliser une hiérarchie mémoire complète (caches + mémoire principale) et flexible ainsi que le système d'interconnexion. Enfin, l'utilisation d'un système d'exploitation complet est un plus qui permet d'approcher la réalité d'exécution des applications via l'ordonnanceur du système.

L'outil doit être suffisamment précis pour fournir des statistiques détaillées sur les composants : nombre d'instructions exécutées, nombre de cycles nécessaires, nombre de lectures et écritures, temps d'attente du au *miss* etc. Nous devons également avoir des informations sur les échanges entre les blocs de l'architecture, comme la quantité de transactions sur les bus.

4.1.2 Estimation des caractéristiques de mémoire émergentes

On cherche à évaluer l'impact de l'intégration de NVM dans une hiérarchie mémoire. A un niveau architectural, ces technologies influent sur trois leviers : les latences d'accès, les coûts énergétiques (dynamiques comme statiques) et la surface de silicium occupée. On a donc besoin d'un simulateur de NVM nous donnant au minimum ces trois caractéristiques.

Les latences d'accès sont les principales informations nécessaires à nos simulations. La flexibilité de l'architecture mémoire nous permet de changer facilement ce paramètre. Ainsi, notre modélisation comportementale des NVM se limite à une modification des latences d'accès en lecture et en écriture. Le principe de non volatilité n'est pas simulé ici car il ne nous est pas nécessaire. On considère une cellule éteinte dès lors que l'accès mémoire est terminé.

Pour l'estimation de consommation énergétique, nous devons adopter une approche analytique. Le simulateur de NVM fournit les coûts unitaires d'accès en lecture et écriture.

Le simulateur d'architecture fournit leur nombre. On peut donc calculer la consommation dynamique de la hiérarchie de caches en combinant ces valeurs. Le modèle de consommation de l'énergie statique utilise la puissance statique renvoyée par le simulateur de NVM et le temps d'exécution donné par le simulateur d'architecture.

4.1.3 Estimation de consommation d'une architecture classique

A la différence des travaux précédents [95], nous étendons le flot générique par l'ajout d'une estimation de la consommation énergétique de la totalité de l'architecture. Nous considérons ici la perspective de la mémoire cache où est intégrée la STT-MRAM comme trop réductrice car elle ne permet pas de mesurer pleinement l'impact total de cette technologie. En effet, ses aspects négatifs en matière de latences influent sur le temps d'exécution des applications, et donc sur l'énergie statique de toute la hiérarchie mémoire. Il est donc important de considérer les caches SRAM lors des évaluations.

4.2 MAGPIE : un framework d'exploration au niveau système

Dans les chapitres 2 et 3, nous avons vu que l'écosystème logiciel existant permet des explorations architecturales mais dans un contexte limitant le passage à l'échelle. En effet, les différents outils ne communiquent pas entre eux et cette étape est à la charge de l'architecte. Dans cette section nous présentons le framework d'exploration MAGPIE (*Multicore Architecture enerGy and Performance evaluatIon Environment*) [24, 25] que nous avons développé dans le cadre de cette thèse.

4.2.1 Présentation générale

Les opportunités d'optimisation à des fins d'efficacité énergétique existent à plusieurs niveaux. Tout d'abord, des optimisations du logiciel peuvent diminuer l'empreinte énergétique de celui-ci [106]. Ensuite, des architectures dynamiques innovantes capables de s'adapter à un contexte d'exécution pour une meilleure efficacité énergétique sont un autre levier [81]. Enfin l'intégration de composants technologiques aux propriétés physiques moins énergivores est envisageable [77]. Il est intéressant d'explorer la question de manière synergique pour un gain maximal. Pour cela, il est indispensable de disposer d'outils facilitant une étude à l'échelle de l'ensemble des couches mentionnées précédemment.

Le framework présenté ici vise à répondre à cette attente en proposant un flot automatisé d'évaluation pour l'exploration d'architectures multicœurs intégrant des technologies de mémoires non volatiles, telles que les mémoires magnétiques. Ce flot permet à

un concepteur d'évaluer facilement à l'échelle globale d'un système l'impact de différents choix architecturaux et technologiques. Plus concrètement, nous montrons à travers une combinaison judicieuse d'outils comment ce flot automatique permet d'étudier les variations en surface, en performance, en puissance et en consommation énergétique selon les paramètres architecturaux ou technologiques choisis pour une application donnée. Un accent particulier est mis sur l'exploration de différentes mémoires caches. A notre connaissance, il n'existe aucune implantation d'un tel flot dans la littérature. Or celui-ci est un ingrédient indispensable pour adresser facilement à plusieurs niveaux un problème d'actualité tel que l'efficacité énergétique.

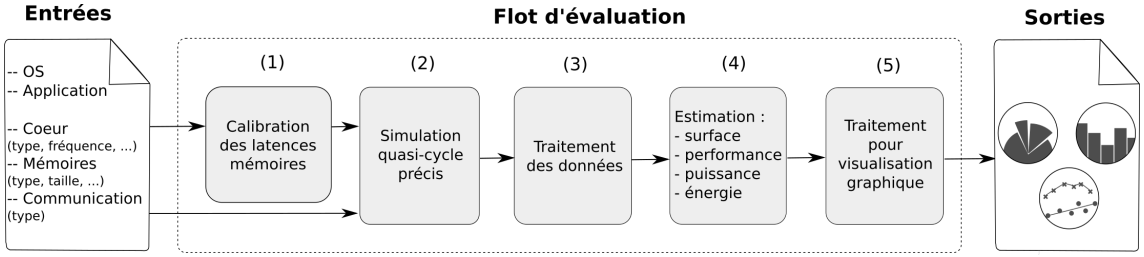


FIGURE 4.1 – Flot d'évaluation d'architectures multicœurs à mémoires non volatiles

Le flot est composé de cinq étapes représentées dans la figure 4.1. Les paramètres d'entrées spécifient : le système d'exploitation et l'application à exécuter, les paramètres des cœurs, des mémoires et des mécanismes de communication. Dans la première étape, les latences des mémoires sont calibrées en fonction des paramètres choisis, i.e., taille, type, associativité et technologie de gravure. Ces latences et le reste des paramètres d'entrée sont utilisés dans la deuxième étape correspondant à la simulation du système. Cela produit des résultats concernant les activités de chaque composant matériel. La troisième étape consiste à extraire ces informations et à les encoder dans un format adapté pour des outils d'estimation de métriques. La quatrième étape utilise ces outils et des modèles de calculs pour estimer la surface, la puissance, les performances et la consommation énergétique du système. Enfin, lors de la dernière étape, ces résultats sont mis en forme pour une visualisation graphique.

4.2.2 Mise en œuvre

Nous avons implanté ce flot en utilisant différents outils selon les étapes présentées par la Figure 4.1. La première étape correspondant à la calibration des latences de mémoires est réalisée avec NVSim [28]. L'outil est configuré grâce aux informations sur le cache fourni en entrée de MAGPIE. Nous extrayons les latences, les coûts énergétiques, la puissance statique et la surface.

La deuxième étape est mise en œuvre avec gem5 [9] qui offre une simulation « Full System » d'architectures multicœurs à un niveau proche du cycle. Dans gem5, différents paramètres du système peuvent être facilement modifiés : types et nombre de cœurs, caractéristiques des caches, système d'exploitation, etc. En résultat d'une simulation, gem5 produit de riches informations statistiques liées surtout aux performances telles que le temps d'exécution ou les transactions mémoires effectuées en lecture et en écriture.

Ces informations sont extraites dans la troisième étape par un script Python basé sur un travail antérieur [108] que nous avons amélioré. Nous avons notamment implémenté de nouveaux « template » XML pour des modèles de cœurs ARM Cortex-A15 et ARM Cortex A-7. Nous avons aussi optimisé la lecture des fichiers de sortie gem5 qui peuvent être très volumineux en taille afin de réduire le temps d'extraction des données.

L'estimation énergétique de la quatrième étape est divisée en deux parties : une estimation des mémoires non volatiles et une estimation du reste de l'architecture. L'estimation des mémoires non volatiles se fait grâce à des calculs utilisant les informations de la simulation et les informations fournies par NVSim durant la première étape du flot. La consommation énergétique des caches lors de l'exécution d'une application est obtenue en sommant les deux composantes suivantes :

$$E_{stat} = P_{stat} * T \ , \tag{4.1}$$

$$E_{dyn} = (E_{dynWrite} * N_{write}) + (E_{dynRead} * N_{read}) + (E_{dynMiss} * N_{miss}) \ , \tag{4.2}$$

où T est le temps d'exécution d'une application fourni par gem5, P_{stat} est la puissance statique d'une mémoire cache obtenue à l'aide de NVSim, les tuples ($E_{dynWrite}$, $E_{dynRead}$, $E_{dynMiss}$) et (N_{write}, $N_{dynRead}$, $N_{dynMiss}$), obtenus respectivement à l'aide de NVSim et gem5, dénotent les énergies dynamiques et le nombre de transactions mémoires selon les trois types d'accès mémoire distingués.

L'estimation du reste de l'architecture est faite avec McPAT [62], un outil développé par HP Labs, qui permet de calculer la consommation énergétique d'une architecture matérielle n'intégrant pas de technologies émergentes. Il fournit une estimation de la surface et de la consommation énergétique de chaque composant de l'architecture grâce aux statistiques extraites auparavant et aux modèles de cœurs XML que nous avons développés.

Enfin, pour permettre une bonne visualisation des résultats, différents scripts traitent ces estimations et génèrent des graphes.

4.2.3 Optimisations de simulation

L'exécution du flot MAGPIE est majoritairement dominée par la simulation gem5. Pour accélérer la simulation, nous avons intégré la possibilité d'utiliser la technique du

checkpointing, illustrée par la Figure 4.2.

Un checkpoint est une sauvegarde à un instant T de l'état de la simulation. Un checkpoint sauvegarde l'état du processeur, notamment le pointeur de code, ainsi que l'état de la mémoire principale et le disque dur.

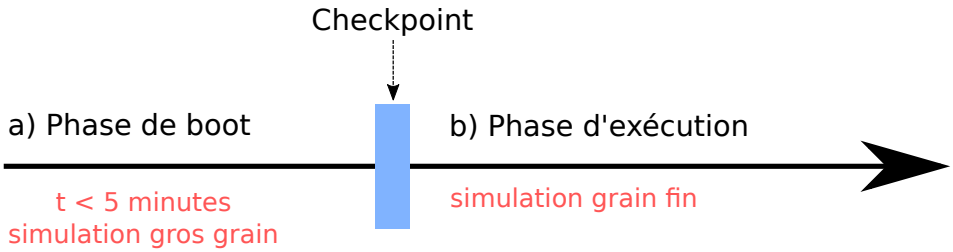


FIGURE 4.2 – Illustration du phénomène de checkpoint avec gem5

Un checkpoint est très utile si l'on doit fréquemment recommencer une simulation à un même endroit. Typiquement, une simulation MAGPIE nécessite de lancer le noyau Linux puis une fois le *boot* terminé, l'application est automatiquement exécutée. En fonction du nombre de cœurs dans l'architecture, la phase de *boot* peut prendre entre 30 minutes et plusieurs heures. Pour éviter cette perte de temps, on peut faire un checkpoint une fois la phase de *boot* terminée, puis recommencer depuis cette sauvegarde et exécuter les applications.

Pour accélérer le processus de *boot* lors du checkpoint, il est possible d'initialiser l'architecture sans les caches et avec un modèle de processeur dit « atomique ». Ce modèle s'affranchit de la complexité du modèle le plus détaillé. Il n'y a pas de pipeline, de prédiction de branchement etc, et les unités de calculs sont très simples : toute opération est résolue en un cycle. Les accès mémoires se font également en un seul cycle. Le temps de *boot* varie alors entre 3 et 30 minutes.

Une fois la sauvegarde effectuée, on peut relancer la simulation depuis ce point mais cette fois-ci en utilisant un modèle de cœur détaillé et en ajoutant la hiérarchie de caches.

4.2.4 Application sur une architecture réelle

Considérons ici la puce Exynos 5 Octa (5422) représentée sur la Figure 4.3. Sur cette architecture de type ARM, on distingue deux clusters nommés « big » et « LITTLE ». Chaque cluster comporte quatre cœurs avec des caches privés d'instructions et de données, ainsi qu'un cache L2 partagé entre les quatre cœurs. Les clusters sont connectés entre eux via un bus et les caches sont cohérents entre clusters. La mémoire RAM est une mémoire LPDDR3 de 2Go.

Sur une architecture de ce type, le cluster « big » est doté de cœurs Cortex-A15 haute performance. Ces derniers proposent comme fonctionnalités l'exécution désordonnée (*out-of-order*), l'exécution spéculative, la prédiction de branchement, une unité de calcul vectoriel etc. Ces cœurs sont cadencés à une vitesse maximale de 2GHz. Ce cluster est très performant mais très énergivore.

Sur le cluster « LITTLE », les cœurs Cortex-A7 sont plus petits et moins puissants. Par exemple, l'exécution est ordonnée et il n'y a pas d'unité de calcul vectoriel. Les cœurs sont cadencés au maximum à 1.4GHz. Cette organisation permet d'avoir un cluster moins performant que le « big » mais aussi gourmand en énergie.

Tous ces cœurs partagent le même jeu d'instruction ARM. D'un point de vue système, le noyau voit 8 cœurs sur lesquels il peut exécuter les applications. L'ordonnanceur du noyau Linux a connaissance de la disparité entre les cœurs en matière de performance.

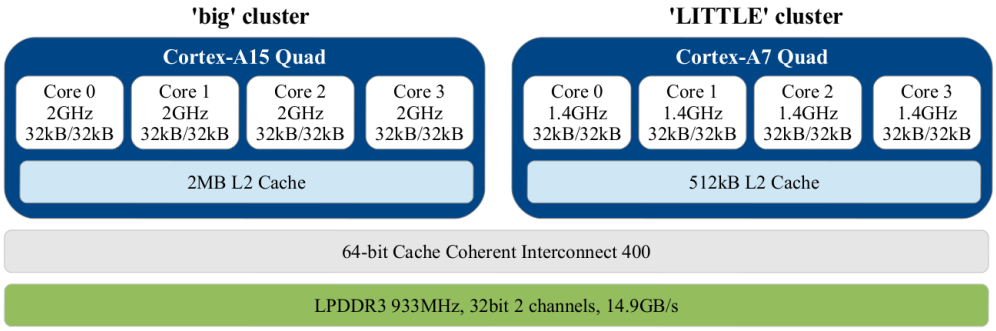


FIGURE 4.3 – Architecture de la puce Exynos 5 Octa [18]

Hormis le système d'interconnexion, nous avons créé un modèle de cette architecture pour le framework MAGPIE. Ce modèle est basé sur une carte Odroid-XU4 [39] contenant une puce Exynos 5 Octa et a été créé en plusieurs étapes. Premièrement, nous avons calibré l'architecture du processeur en utilisant des informations publiquement accessibles sur la puce. Ensuite, nous avons exécuté un ensemble d'applications de la suite PARSEC [8] sur la carte et sur la puce. Ce benchmark cible les architectures multicœurs et propose des applications avec différents schémas d'accès mémoires, de partage de données ou encore de synchronisation. En utilisant des compteurs de performances présents sur la carte, les résultats ont été comparé avec MAGPIE et nous avons pu affiner la précision du modèle en conséquence. Les configurations architecturales étudiées sont les suivantes :

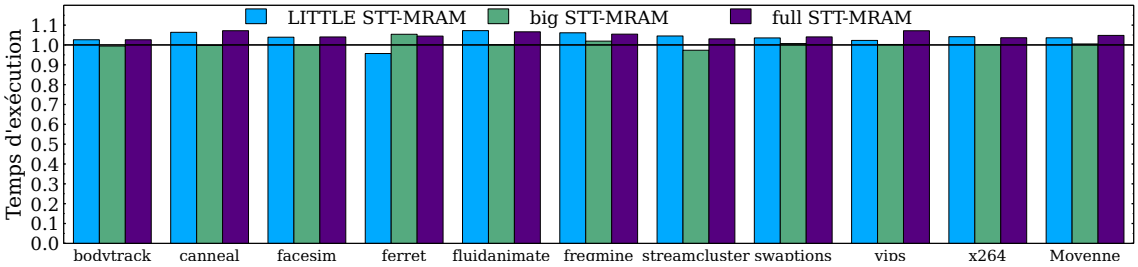
— tous les caches en SRAM (scénario de référence)

— le L2 du cluster « LITTLE » en STT-MRAM (L2 LITTLE STT-MRAM)

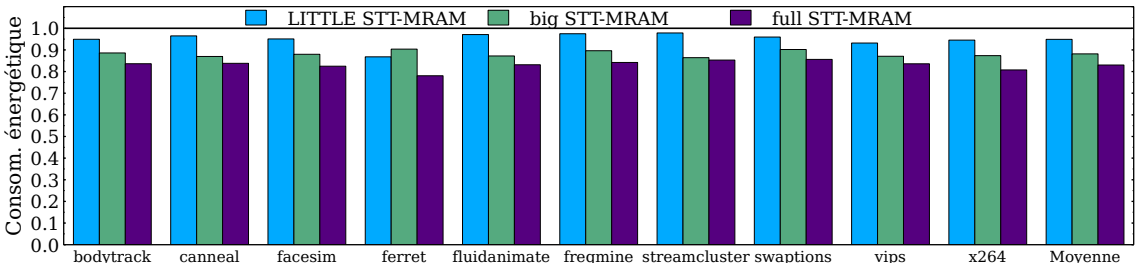
— le L2 du cluster « big » en STT-MRAM (L2 big STT-MRAM)

— tous les L2 en STT-MRAM (Full STT-MRAM)

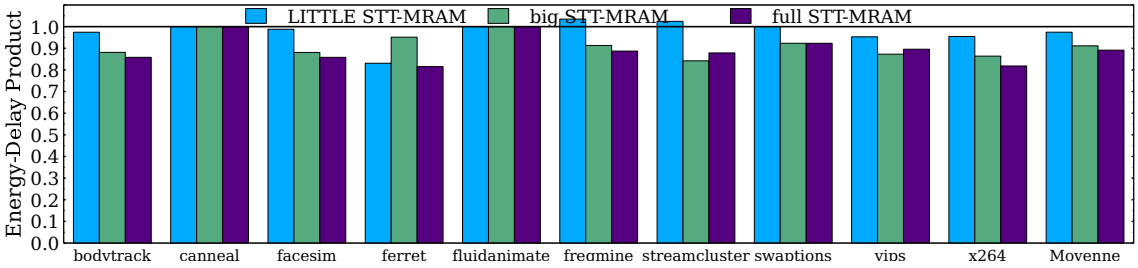
Les résultats de temps d'exécution, d'énergie et d'efficacité énergétique sont respectivement donnés par les Figures 4.4a, 4.4b et 4.4c. Tous ces résultats sont normalisés au scénario de référence. Dans chaque cas, une valeur en dessous de 1 indique une amélioration, sinon une dégradation.



(a) Temps d'exécution



(b) Consommation énergétique



(c) Efficacité énergétique

FIGURE 4.4 – Résultats de temps d'exécution, de consommation énergétique et d'efficacité énergétique sur une reproduction d'une architecture Exynos 5 Octa intégrant de la STT-MRAM. Les résultats sont normalisés au scénario « full SRAM »

On voit sur la Figure 4.4a que l'intégration de STT-MRAM sur le cluster « LITTLE » augmente en moyenne de 4% le temps d'exécution total des applications. En matière de consommation énergétique, celle-ci est en moyenne de réduite de 5%. Enfin, l'efficacité énergétique de cette configuration est supérieure à celle d'une configuration « full SRAM » de 3%.

Pour la seconde configuration qui intègre de la STT-MRAM uniquement sur le cache L2 du cluster « big », le temps d'exécution est presque identique au scénario de référence. La pénalité n'est que de 1%. En termes de consommation énergétique, celle-ci diminue de

12%. Enfin, l'efficacité énergétique de cette configuration est 9% supérieure à la configuration de référence.

La dernière configuration dite « full STT-MRAM » augmente le temps d'exécution de 5% mais diminue la consommation énergétique de 17%. Parmi les trois scenarii, celui-ci est à la fois le pire en ce qui concerne le temps d'exécution mais aussi le meilleur sur le plan de la consommation énergétique. Cependant, c'est ce scénario qui propose la meilleure efficacité énergétique. Comparé à la configuration « full SRAM », elle est améliorée de 11%.

4.3 Explorations spécifiques pour la gestion de la mémoire

Le framework MAGPIE se place à un niveau prenant en compte la technologie, l'architecture et aussi le logiciel. Malgré son automatisation, son utilisation est parfois limitée à cause du temps de simulation élevé de gem5. Pour des explorations localisées ne nécessitant pas l'intervention d'un système d'exploitation, nous proposons une alternative utilisant sur le simulateur ChampSim.

4.3.1 Présentation générale

ChampSim est un simulateur basé sur des traces d'applications. Cette technique permet d'accélérer les simulations par rapport à l'exécution d'une application complète [75]. En effet, une trace contient une partie seulement du programme, la *Region of Interest* (ROI) Dans le cas de ChampSim, les traces sont obtenues en identifiant la ROI avec l'outil SimPoint [38]. Une ROI (Figure 4.5) est définie par le numéro de séquence S de l'instruction à laquelle elle commence et le nombre d'instructions N' qu'elle contient. Pour collecter les traces, on exécute l'application sur une machine réelle en utilisant l'outil PIN [68]. Ce dernier s'intercale entre l'application et le système d'exploitation, exécute $S-1$ instructions, puis démarre la capture des instructions qui exécutées jusqu'à avoir une trace contenant N' instructions.

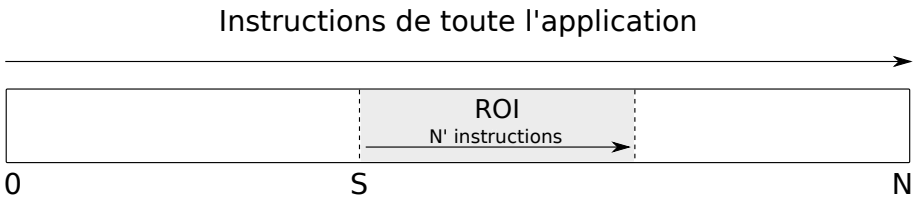


FIGURE 4.5 – Illustration d'une ROI de N' instructions

Les informations contenues dans la traces sont :

— le compteur ordinal de l'instruction

— le type d'instruction (branchement ou non)
— si c'est un branchement, doit-il être effectué
— les registres source et destination
— les adresses source et destination

Le mode du simulation est similaire au *Syscall Emulation* de gem5, c'est-à-dire sans système d'exploitation. Le but premier de ChampSim est d'évaluer la gestion de la mémoire lorsque l'on change la taille des mémoires caches, le système de *prefetching* et enfin la stratégie de remplacement du LLC. Du fait que les applications soient sous forme de traces, il n'y a pas d'ordonnanceur système. Le processeur lit la trace d'instructions et l'exécute au fur et à mesure. L'exécution peut se faire dans le désordre.

Chaque cache de la hiérarchie mémoire contient plusieurs files d'évènements pour les lectures, les écritures et le *prefetching*. A chaque cycle, les caches exécutent, s'ils le peuvent, les évènements en haut de la file. Les transactions entre les caches se font par l'ajout d'évènements dans les files. Typiquement, si un *miss* est déclenché dans le cache L1 lors de la lecture d'une donnée X, alors le cache L1 écrit dans la file de lecture du L2 une requête de lecture pour la donnée X.

L'avantage de ChampSim est sa vitesse d'exécution qui nous permet d'explorer rapidement beaucoup de configurations différentes. De plus, il est spécifiquement développé pour tester les stratégies de *prefetching* et de remplacement. D'une manière similaire à MAGPIE, nous avons automatisé une grande partie du processus de configuration et d'exécution via des scripts.

4.3.2 Flot d'exécution

Le schéma d'exécution de flot est donné par la Figure 4.6. Les entrées du flot sont restreintes à quelques paramètres seulement, comme l'application, la stratégie de remplacement et les latences des caches. A la différence de MAGPIE, le lancement de NVSim et l'extraction des latences ne sont pas automatisés complètement. Une exploration NVSim en amont est nécessaire. Une fois les latences obtenues, elles peuvent être compilées dans un fichier qui se chargera de les injecter dans le simulateur en fonction de la configuration choisie.

4.3.3 Estimation de la consommation énergétique de la hiérarchie mémoire

Le traitement des données est également partiellement automatisé. Les résultats que l'on récupère en sortie de ChampSim sous forme de texte sont le nombre de cycles to-

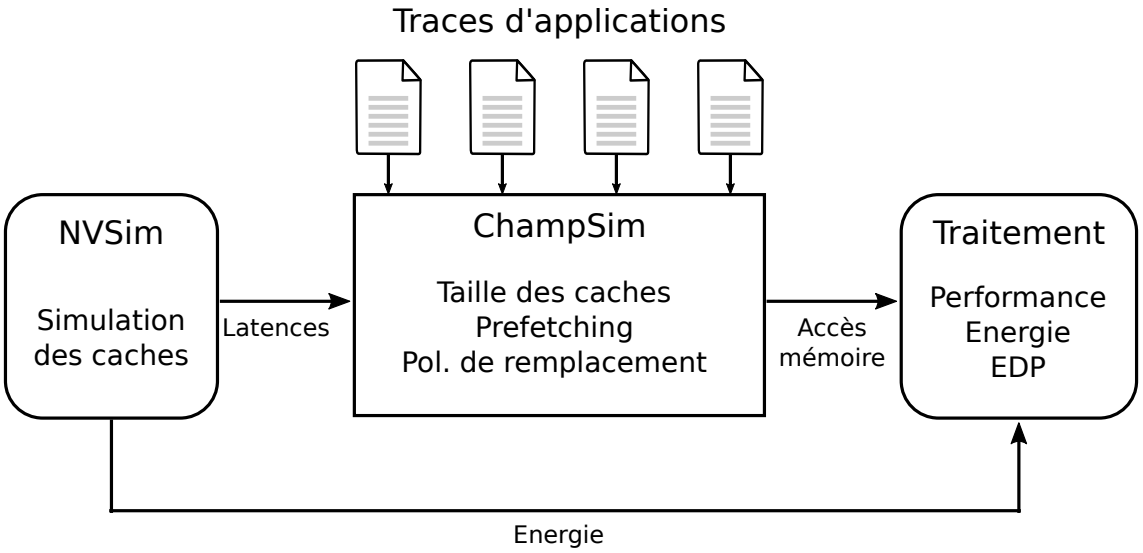


FIGURE 4.6 – Flot de simulation avec ChampSim

tal et le nombre d'accès en lecture et écriture sur les caches. Ces accès sont différenciés en fonction de leur provenance : cœur (lecture/écriture directe), *prefetching* (mécanisme indépendant) ou mémoire (échange de données entre les caches).

Via des scripts, ces résultats sont extraits et intégrés dans une feuille de calcul. L'estimation de consommation énergétique de la hiérarchie mémoire est alors effectuée en utilisant le modèle énergétique suivant :

$$E_{dyn} = (N_{read} + N_{write}) \times E_{access}\,, \tag{4.3}$$

$$E_{stat} = ExecTime \times P_{leak}\,, \tag{4.4}$$

où E_{dyn} et E_{stat} représentent respectivement l'énergie dynamique et statique, N_{read} et N_{write} représentent le nombre total de lectures et d'écritures, E_{access} est l'énergie dynamique pour une lecture et une écriture, $ExecTime$ le temps d'exécution et P_{leak} la puissance statique de la mémoire cache.

Pour la mémoire principale, nous utilisons la formule suivante pour calculer la consommation énergétique [69] :

$$E_m = E_a + E_b + E_c\,, \tag{4.5}$$

$$E_a = R \times RD + W \times WR\,, \tag{4.6}$$

$$E_b = (R_M + W_M) \times (PRE + ACT)\,, \tag{4.7}$$

$$E_c = (T/T_{REF}) \times REF + T \times ACT_{BG}\,, \tag{4.8}$$

ou E_m est l'énergie totale consommée par la mémoire, E_a est la consommation due aux lectures et écritures, E_b la consommation due au *miss* qui entraînent les opérations de pré-chargement et d'activation de pages, et E_c est l'énergie statique due au rafraîchissement et la puissance statique. On ne considère pas ici de mode *low-power* qui réduit ACT_{BG}.

4.4 Résumé

Dans ce chapitre, nous avons présenté le cadre générique d'exploration que nous devons mettre en place pour nos explorations. La Figure 4.7 illustre les différentes étapes de ce flot de conception. Nous avons identifié trois besoins que sont la simulation d'architectures, la modélisation de technologies non volatiles émergentes et l'estimation de la consommation énergétique.

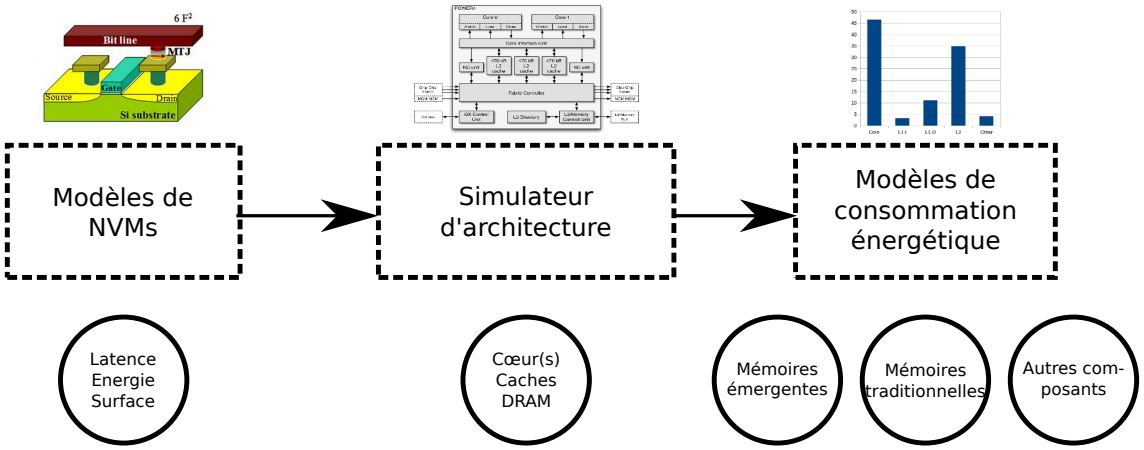


FIGURE 4.7 – Schéma de principe du cadre d'explorations architecturales

Dans un second temps, nous avons présenté une première implémentation de notre schéma générique, le framework d'explorations architecturales MAGPIE. Ce flot vise à aider un concepteur à évaluer facilement l'impact de différents choix architecturaux et technologiques dans un système intégrant des technologies de mémoires émergentes. En connectant automatiquement différents outils de simulation et d'estimation de consommation énergétique, MAGPIE permet d'accélérer le processus d'exploration [25]. De plus, cet outil permet de définir une exploration multi-niveaux ciblant à la fois les parties logicielles, architecturales et technologiques. Nous avons illustré l'utilisation du framework avec un exemple d'intégration de STT-MRAM sur une architecture réelle, une puce Exynos 5 Octa.

Enfin, nous avons présenté une seconde implémentation de notre modèle générique via le simulateur ChampSim. Cette version du flot est beaucoup plus rapide que MAGPIE et permet d'explorer facilement l'impact des politiques de remplacement de cache dans

une architecture. Ce point sera abordé par la suite.

Dans le chapitre suivant, nous présentons deux optimisations architecturales permettant d'améliorer la performance de la hiérarchie de caches contenant de la mémoire de type STT-MRAM. On vise plus spécifiquement le cache de dernier niveau. On s'intéresse également au choix des métriques d'optimisation des caches, comme la surface ou l'énergie.

I'm gonna make him an offer he can't refuse.

Don Vito Corleone - The Godfather (1972)

Chapitre 5

Opportunités d'optimisations architecturales sur une mémoire cache de dernier niveau à base de STT-MRAM

Dans ce chapitre, nous analysons les possibilités d'optimisations architecturales qui existent pour permettre d'améliorer les performances de la technologie STT-MRAM. Nous introduisons les questions soulevées par le changement de mémoire. Ces questions se posent à plusieurs niveaux. Le cache en lui-même, puis son interaction avec la hiérarchie mémoire.

Dans un premier temps, nous menons une exploration de l'architecture interne d'une mémoire cache pour montrer que la STT-MRAM permet certains choix architecturaux différents de la SRAM. Ensuite, nous montrons que la conception d'un cache peut se faire selon différents critères prenant en compte la surface, la latence ou encore l'énergie dynamique des accès. Enfin, nous analysons la gestion des données dans les caches, et plus particulièrement les interactions qui existent au sein de la hiérarchie mémoire.

La deuxième partie de ce chapitre est consacrée à des expériences préliminaires visant à répondre aux questions suivantes : à quel niveau de cache intégrer de la STT-MRAM, la fréquence du processeur peut-elle avoir un impact sur les latences de caches, quelles sont la ou les optimisations de conception à utiliser et quel est l'impact énergétique des stratégies de remplacement de cache. Nous utiliserons pour cela les environnements d'exploration présentés aux chapitre 4.

5.1 Questions soulevées par l'utilisation de la technologie STT-MRAM

5.1.1 Niveau de cache et optimisation de conception

Les mémoires caches sont définies par plusieurs caractéristiques, et notamment leur latence ou leur puissance statique. Ces critères ne sont pas fixes et varient en fonction du design de la mémoire. En effet, il existe différentes possibilités de conception ayant une influence sur ces métriques. La réalisation d'un cache est un compromis entre la latence d'accès, la puissance statique et l'énergie dynamique. On peut également ajouter la surface de silicium occupée. Ce compromis est illustré par la Figure 5.1. Pour un cache donné, il existe une multitude de design favorisant l'une ou l'autre des métriques évoquées. Un concepteur essaiera au maximum de diminuer toutes ces valeurs, mais elles sont inversement proportionnelles entre elles : l'augmentation de l'une diminue l'autre.

Pour sélectionner la métrique qui sera favorisée, on peut utiliser le niveau d'intégration

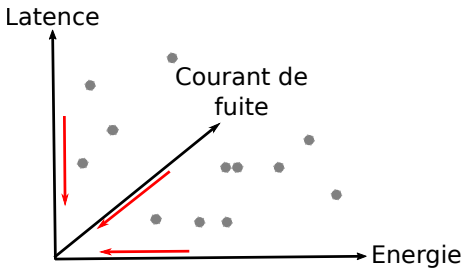


FIGURE 5.1 – Compromis entre latence et énergie pour le design d'un cache

dans la hiérarchie mémoire. Les architectes ont tendance à optimiser les mémoires caches de premier niveau pour diminuer au maximum leur latence. En effet, les mémoires de niveaux L1-I/D opèrent à des fréquences très élevées et doivent répondre au processeur le plus rapidement possible. Le second niveau de cache est généralement optimisé pour diminuer sa consommation dynamique. Ses latences sont alors plus importantes, mais elles sont en partie masquées par les caches de premier niveau. Sa puissance statique, dépendante de sa surface, n'est pas très élevée car la surface occupée par le L2 reste faible. Enfin, le dernier niveau de cache est optimisé pour diminuer au maximum la puissance statique, au prix de latences plus élevées.

Les choix de conception d'un cache ne sont pas les mêmes avec la STT-MRAM. Par exemple, il n'est pas nécessaire de diminuer la puissance statique puisque celle-ci est intrinsèquement très faible. Les latences et l'énergie dynamique sont elles plus importantes. Or, dans une cellule MTJ, ces métriques ne peuvent être optimisées conjointement. Les caractéristiques technologiques différentes de la STT-MRAM posent donc la question des choix de conception.

5.1.2 Interactions des mémoires caches dans la hiérarchie mémoire

Les questions soulevées précédemment sont liées au design et à l'architecture des mémoires caches. Néanmoins, un changement de technologie dans une hiérarchie mémoire nécessite de prendre du recul et d'analyser l'impact de ce changement en prenant en compte les autres caches.

Le principal problème de la STT-MRAM est sa latence d'écriture. On a vu au chapitre 3 qu'il existe différents angles pour attaquer ce problème (technologie, circuit, architecture, logiciel). Pour nos travaux, nous devons trouver un moyen de diminuer le nombre d'écritures par des méthodes architecturales. Cela nécessite dans un premier temps d'analyser les interactions qui existent au niveau de la hiérarchie de mémoires caches, et notamment les écritures. La Figure 5.2 illustre les communications qui existent entre ces composants. Parmi celles-ci, on doit déterminer si toutes les écritures sont de même nature et

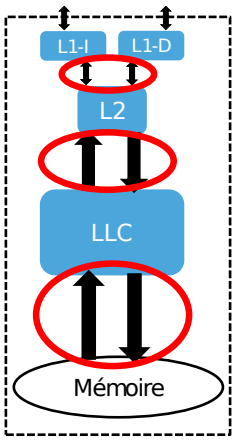


FIGURE 5.2 – Transactions dans la hiérarchie mémoire

	LLC 2Mo	LLC 4Mo	Différence
Latence (R/W) [ns]	1.33	1.47	+ 9%
Énergie R/W [pJ]	0.35	0.38	+ 9%
Énergie stat. [mW]	147	229	+ 36%
Surface [mm2]	5.32	10.88	+ 51%

Tableau 5.1 – Configuration des caches de dernier niveau

s'il n'existe pas des écritures plus pénalisantes que d'autres.

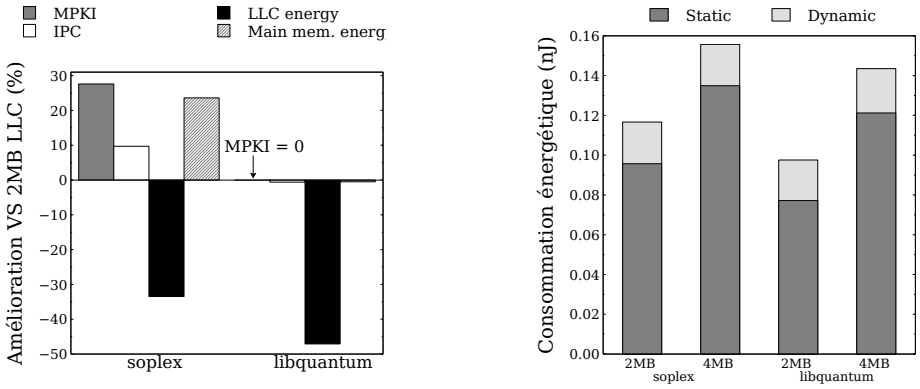
5.2 Augmentation de la capacité de stockage de la mémoire cache

On a vu au chapitre 2 que le MPKI est utilisé pour évaluer l'efficacité des mémoires caches. Une possibilité pour réduire le MPKI est d'augmenter la capacité de stockage de la mémoire cache. Cette dernière contient plus de données et cela réduit la probabilité qu'un *miss* ait lieu.

On se propose d'analyser l'effet de l'augmentation de la taille d'une mémoire cache d'un facteur 2 avec de la SRAM. Pour illustrer cet exemple, nous évaluons deux applications de la suite SPEC CPU2006 ayant deux modèles différents d'accès à la mémoire : *soplex* et *libquantum*. Nous considérons la configuration suivante : un cœur avec une hiérarchie à 3 niveaux de cache : $32Ko$, $256Ko$ et $2Mo$. Nous évaluons l'impact de ce changement de capacité sur le MPKI, l'IPC et la consommation énergétique du LLC et de la mémoire principale. Les caractéristiques des caches sont données par le tableau 5.1.

La Figure 5.3a montre l'impact d'un LLC de $4Mo$ normalisé au scénario de référence,

$2Mo$. Pour l'application *soplex* le MPKI est diminué de 27.6%, entraînant une exécution plus rapide de 9.7%. Les consommations énergétiques du LLC et de la mémoire principale sont respectivement augmentées de 33% et diminuées de 23%. Bien que ce changement soit bénéfique pour le temps d'exécution de l'application *soplex*, cela entraîne une dégradation de la consommation énergétique du LLC. Les résultats de l'application *libquantum* sont en revanche différents. Le MPKI est identique (aucune amélioration ni dégradation), tandis que l'IPC est légèrement dégradé de 0.6%. Cela indique que l'application *libquantum* n'est pas sensible à la taille de la mémoire cache. Les consommations énergétiques du LLC et de la mémoire principale sont également augmentées à cause des transactions plus coûteuses et de l'énergie statique de la mémoire cache de $4Mo$ ($+51\%$). La répartition de l'énergie dynamique et statique du LLC est détaillée sur la Figure 5.3b. On voit que 80% de l'énergie provient de la partie statique.



(a) MPKI, IPC, et énergie

(b) Répartition énergétique

FIGURE 5.3 – Évaluation d'un LLC de $2Mo$ et $4Mo$ pour les applications *soplex* et *libquantum*

L'augmentation de la taille de la mémoire cache montre des résultats intéressants pour la performance. Dans le cas d'une application sensible à la taille de la mémoire cache, le MPKI est diminué et l'IPC augmente. Dans le cas contraire, le MPKI ne change pas et l'IPC est très légèrement dégradé. Les performances sont au minimum maintenues. Cependant, ces résultats sont nuancés par plusieurs inconvénients. Premièrement, la consommation du LLC est augmentée drastiquement, parfois sans gain en performance. Deuxièmement, l'augmentation par un facteur $2\times$ de la capacité de stockage de la mémoire cache double la surface de silicium occupée. C'est aspect est crucial dans le design des caches de dernier niveau. Ces derniers occupent un espace non négligeable sur une puce [56] et augmenter leur taille n'est pas toujours réaliste par rapport aux contraintes de design. Enfin, doubler la taille de la mémoire cache entraîne des pénalités importantes en matière de latence et d'énergie. Une mémoire cache plus grande physiquement implique des accès plus lents et plus coûteux. Les circuits périphériques permettant les opérations de lecture et d'écriture

sont plus longs, entraînant un surcoût énergétique et temporel [77, 96]. Ces phénomènes sont illustrés par la Figure 5.4. On observe une augmentation progressive de l'énergie et de la latence, notamment à partir de $4Mo$ pour la latence.

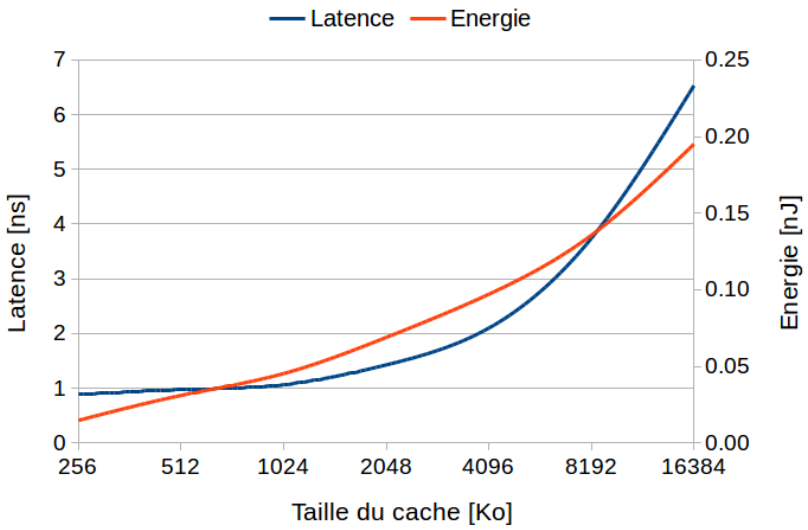


FIGURE 5.4 – Effets de l'augmentation de la taille d'un cache SRAM sur la latence et l'énergie

Dans la section suivante, nous proposons une exploration de l'organisation interne d'un cache avec de la STT-MRAM pour mitiger au maximum ces pénalités.

5.2.1 Exploration de l'organisation interne d'un cache

La Figure 5.5 présente l'organisation interne des composants d'un cache. À gauche est représenté le cache avec les différents *mat* qui le composent. Le circuit de routage est en vert. Chaque *mat* contient des *sub-array*, à l'intérieur desquels se trouve les *array* de cellules mémoire et les circuits périphériques permettant d'y accéder. Cette organisation est indépendante de la technologie.

Cas de la technologie SRAM

Avec une petite capacité de stockage SRAM, le routage est très rapide et la latence est dominée par l'accès à la cellule mémoire à l'intérieur des *sub-array*. Lorsque l'on augmente la capacité de stockage, on agrandit les *array* en augmentant le nombre de cellules. Une cellule SRAM est composée de 6 transistors, l'augmentation du nombre de cellules fait donc rapidement croître la taille des *array*. La latence augmente alors rapidement car les circuits périphériques comme les sélecteurs de lignes/colonnes deviennent plus longs. De

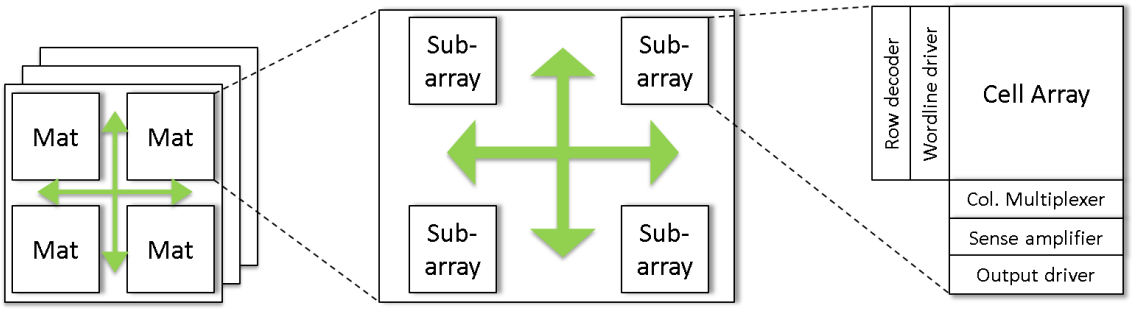


FIGURE 5.5 – Organisation interne d'un cache

plus, il existe des composants appelés *sense amplifier* chargés d'envoyer la donnée lue dans une cellule en dehors de la mémoire cache par de fortes impulsions électriques. L'énergie nécessaire croît en fonction de la taille des *array*.

Ainsi, lorsque l'on augmente la taille de la mémoire cache SRAM, la stratégie adoptée est d'augmenter le nombre de *sub-array* plutôt que d'agrandir leur capacité de stockage [1]. Cela agrandit la surface totale du cache, et par conséquent la taille des circuits de routage. A partir d'une certaine surface, c'est alors les circuits de routage qui dominent la latence, et celle-ci augmente en fonction de la surface occupée.

Cas de la technologie STT-MRAM

La STT-MRAM se comporte de manière similaire sur des caches de petites tailles. La cellule MTJ étant plus lente qu'une cellule SRAM, la latence d'accès est d'autant plus dominée par les *array*.

En adoptant une stratégie similaire à la SRAM, les latences d'accès augmenteraient de la même manière. Cependant, une cellule MTJ se compose d'un seul transistor, ce qui permet d'envisager une stratégie différente. En effet, si l'on augmente la capacité de stockage des *array*, les circuits périphériques autour s'agrandissent moins vite qu'avec de la SRAM. La latence d'accès des *array* augmentera plus lentement qu'avec la SRAM, tout en augmentant la capacité de stockage. Ce n'est qu'après avoir franchi un certain palier que, comme la SRAM, l'ajout de *sub-array* devient intéressant pour la STT-MRAM.

La Figure 5.6 résume la situation que l'on vient d'évoquer. Les chiffres présentés en ordonnée sont une illustration de ce phénomène. Cette figure est découpée en trois zones. La zone A représente les tailles des mémoires caches pour lesquelles la latence des accès SRAM est dominée par les cellules. L'augmentation de la taille de la mémoire cache passe

1. A partir d'une certaine taille seulement, l'augmentation de la taille des *array* est possible sans fortes pénalités avec des caches de faible capacité

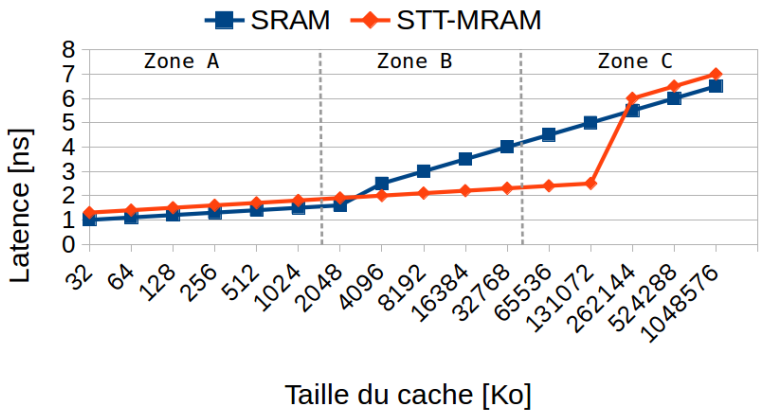


FIGURE 5.6 – Évolution théorique des latences d'accès à la mémoire cache en fonction de la taille pour les technologies SRAM et STT-MRAM

alors par l'augmentation de la capacité des *array*. A partir de $2Mo$, la SRAM entre dans la zone B, où les accès sont dominés par les circuits périphériques et la latence augmente plus vite.

A l'inverse, la STT-MRAM reste dans la zone A plus longtemps grâce à sa densité, puis entre dans la zone C. Ici, les accès deviennent dominés par les accès périphériques et la latence croît plus rapidement.

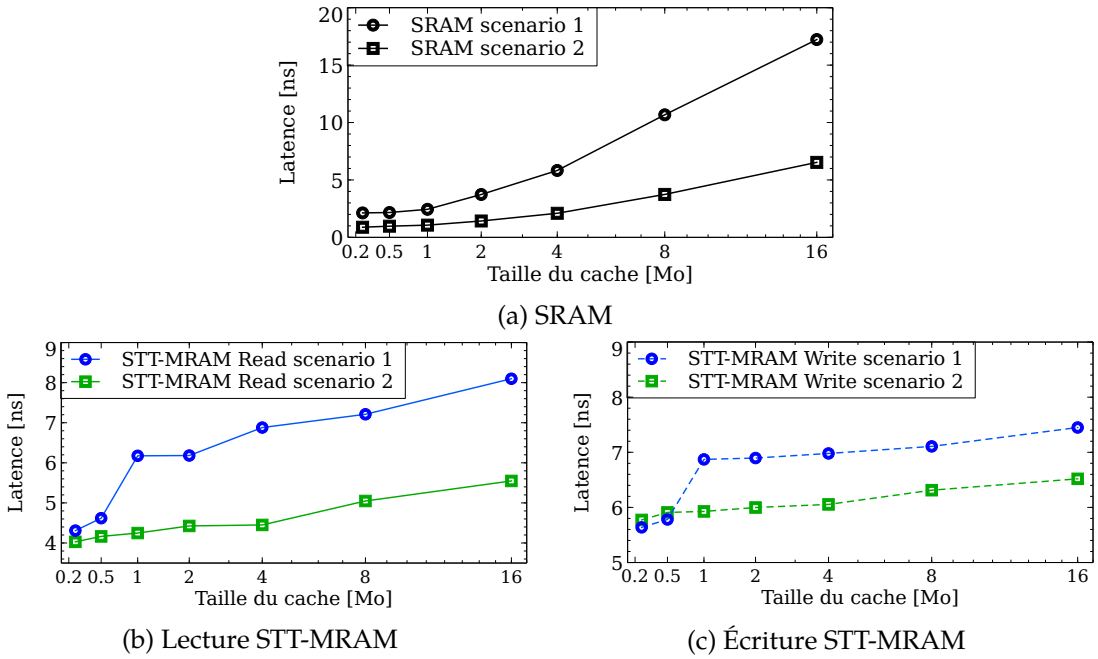
Résultats expérimentaux

Nous proposons de valider nos hypothèses expérimentalement avec NVSim. Parmi les paramètres architecturaux que l'on peut configurer, il existe une option pour spécifier l'évolution des *sub-array*. Nous utilisons ce paramètre pour explorer deux scenarii :

1. pour la SRAM, on augmente la taille des *array* et pour la STT-MRAM, on augmente le nombre de *sub-bank*. Ce scénario représente un mauvais choix d'optimisation.
2. pour la SRAM, on augmente le nombre de *sub-array* et pour la STT-MRAM on augmente la taille des *array*. Ce scénario représente un meilleur choix d'optimisation.

Le scénario 1 est le scénario dans lequel les latences des caches devraient être plus élevées car l'organisation interne n'est pas en accord avec les technologies mémoires utilisées. Le scénario 2 corrige cela en choisissant judicieusement l'optimisation adaptée à chaque mémoire.

La Figure 5.7 montre l'évolution des latences d'accès au cache en fonction de la taille et du scénario considéré. Pour la SRAM (Figure 5.7a), le scénario 1 montre des latences plus élevées que le scénario 2, à cause du mauvais choix d'optimisation de la mémoire.



(a) SRAM

(b) Lecture STT-MRAM

(c) Écriture STT-MRAM

FIGURE 5.7 – Évolution mesurée des latences d'accès au cache en fonction de la taille pour les technologies SRAM et STT-MRAM

On observe le même comportement en lecture (Figure 5.7b) et écriture (Figure 5.7c) pour la STT-MRAM. Néanmoins, l'augmentation de latence est moins marquée en lecture grâce à l'effet de densité. Le temps de routage augmente, mais les circuits périphériques restent toujours plus courts que ceux de la SRAM. On passe d'une latence d'environ $4ns$ pour $256Ko$ à une latence de $8ns$ pour $16Mo$.

La latence d'écriture augmente très peu dans les deux scenarii. En effet, le circuit d'écriture est plus court que celui de lecture. Lors d'une lecture, une fois la donnée lue, il est nécessaire de la faire sortir de la mémoire et de l'envoyer sur le bus. Pour une écriture, cette étape de sortie et de communication est inutile puisque l'initiateur de l'écriture n'attend pas d'acquittement. La latence augmente de manière modérée et la différence moyenne entre les deux scenarii est de 10%, alors qu'elle est de 35% pour la lecture.

Dans cette section, nous avons vu que la densité de la STT-MRAM permet l'augmentation de la capacité de stockage d'une mémoire cache, en général d'un facteur $4\times$. Nous avons mené une exploration architecturale sur l'organisation interne de la mémoire cache pour montrer que celle-ci diffère entre les technologies SRAM et STT-MRAM. En effet, la technologie STT-MRAM est moins sensible aux latences des circuits périphériques, ce qui permet de ré-organiser les composants en conséquence. Dans la suite de ce chapitre, nous présentons une étude sur les possibilités d'optimisation de la mémoire cache utilisant les

résultats de notre analyse de densité.

5.3 Choix d'optimisation lors de la conception d'une mémoire cache

Une mémoire cache est représentée par un ensemble de paramètres architecturaux comme la taille, l'associativité, le routage, le nombre de ports de lecture/écriture etc. Le problème non trivial ici est de trouver la bonne configuration parmi toutes les possibilités pour obtenir une mémoire la plus efficace possible selon les besoins. Pour trouver cette configuration, si elle existe, il faut dans un premier temps réaliser une exploration complète de toutes les possibilités. L'architecte doit ensuite déterminer un ensemble de critères pour trouver les solutions qui paraissent intéressantes, puis parcourir l'espace de design et extraire ces configurations.

Les travaux précédents n'ont que rarement mené ce type d'étude en le combinant à une approche architecturale. De nombreuses propositions ont été faites pour optimiser la cellule de mémoire STT-MRAM [50, 99, 121] pour atténuer les latences, le coût énergétique ou l'asymétrie. Ces approches se font à un niveau circuit. Ici nous sommes à un niveau plus élevé qui est l'architecture.

5.3.1 Approche proposée

Les propositions relevées dans le chapitre 3 intègrent de la STT-MRAM avec certaines caractéristiques de latences et d'énergie mais n'explicitent pas ces valeurs. Aucune exploration n'est menée et les détails des configurations ne sont pas donnés. Nous proposons une méthode d'exploration architecturale découpée en plusieurs étapes :

— une définition minimale des caches que l'on veut explorer (taille, associativité, technologie de gravure…)
— une exploration automatisée de ces configurations
— la définition du critère de sélection de(s) configuration(s)
— une visualisation de l'espace d'exploration
— l'extraction de configuration(s) dans cet espace

Pour ces explorations, nous considérons l'outil NVSim présenté dans le chapitre 3. NVSim permet d'estimer les latences, la puissance et la surface d'un cache. L'outil dispose de nombreux paramètres architecturaux qui peuvent être modifiés, permettant un large choix d'exploration. La configuration des ces paramètres est donnée en entrée par l'architecte. L'outil propose un mode d'exploration automatisé, laissant le designer fixer seulement les paramètres voulus.

5.3.2 Initialisation des variables et exploration

NVSim propose divers paramètres d'exploration pour les mémoires à travers un fichier de configuration. Ce fichier regroupe une trentaine de champs. Dans notre étude, nous fixons seulement les paramètres suivants :

— la taille de la mémoire cache

— la taille des mots

— l'associativité

— le nœud technologique

— la température

Une fois ces valeurs spécifiées, un script les combine à un fichier squelette de NVSim pour créer tous les fichiers d'entrées pour chaque configuration à explorer. Les autres paramètres sont laissés vides et seront explorés par NVSim. L'outil dispose de plusieurs modes d'exploration qui vont optimiser le cache selon plusieurs critères : la latence de lecture ou d'écriture, la surface, l'efficacité énergétique de la lecture etc.

Le lancement de NVsim pour toutes ces configurations est automatisé. En fonction du nombre de possibilités, cette exploration peut prendre plusieurs dizaines de minutes à plusieurs heures. Une fois l'exploration terminée, les résultats sont placés dans des fichiers CSV.

5.3.3 Visualisation et extraction de l'espace d'exploration

Une fois l'espace d'exploration défini, il est possible de le visualiser à travers un logiciel comme Matlab®. Pour cela, nous avons développé un ensemble de scripts prenant en entrée un fichier CSV produit par NVSim et produisant en sortie plusieurs graphiques représentant l'espace d'exploration. Les meilleures configurations selon les critères existants sont visibles, et sont également affichées sous forme textuelle. Les informations extraites pour ces configurations de cache sont les latences d'accès (lecture/écriture), les coûts énergétiques d'accès (lecture/écriture), la puissance statique et la surface.

La figure 5.8 est un exemple de graphique que l'on obtient en sortie de Matlab. Ici, l'espace d'exploration contient 62432 configurations. La meilleure configuration est choisie selon le critère du meilleur ratio entre la latence de lecture, le coût énergétique d'une lecture et la surface occupée par le cache. On appelle ce critère l'ADEP, pour Area-Delay-Energy Product (voir Tableau 5.2).

L'outil permet également de visualiser les latences d'écriture et le coût énergétique des accès, comme montré par la Figure 5.9.

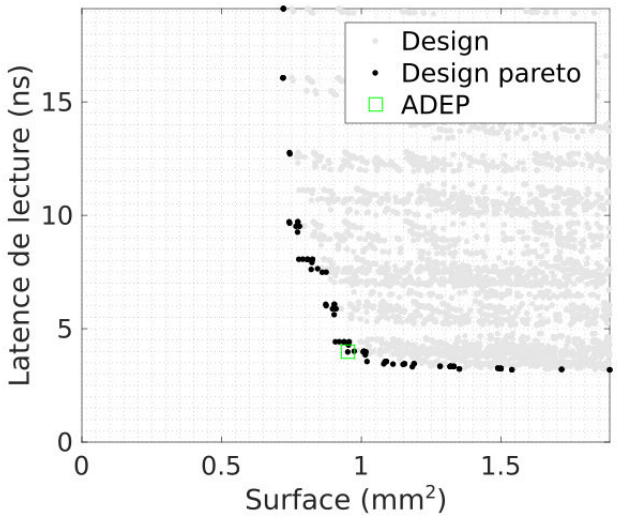


FIGURE 5.8 – Espace d'exploration pour un cache 16 associatif de 2Mo et pour une température de 70 degrés Celsius. Les points noirs sont les points Pareto de l'ensemble avec une marge de 1%.

Critère	Description
ADP	MIN(Area-Delay Product)
AD^2P	MIN(Area-Delay-Square Product)
ADEP	MIN(Area-Delay-Energy Product)
ADE^2P	MIN(Area-Delay-Energy-Square Product)
ReadEDP	MIN(ReadDelay*ReadEnergy)
WriteEDP	MIN(WriteDelay*WriteEnergy)

Tableau 5.2 – Critères de sélection des configurations des mémoires caches

Il existe différents critères de sélection pour la meilleure configuration qui sont résumés dans le tableau 5.2. En fonction du niveau de cache considéré, certains critères sont préférés à d'autres. Par exemple, un cache de premier niveau nécessite un temps de réponse très rapide. Il vaut mieux choisir un critère comme l'AD^2P qui met en avant la latence de la mémoire cache au détriment de l'énergie. A l'inverse, un cache de dernier niveau doit être optimisé pour son énergie statique, qui dépend de la surface. On peut alors sélectionner selon l'ADP, ou l'ADEP si on veut tout de même inclure l'énergie dynamique.

5.3.4 Limites de cette approche

Le critère d'une sélection d'une configuration de cache est un critère local au cache et ne tient pas compte de son intégration dans une architecture complète. Dans certains cas,

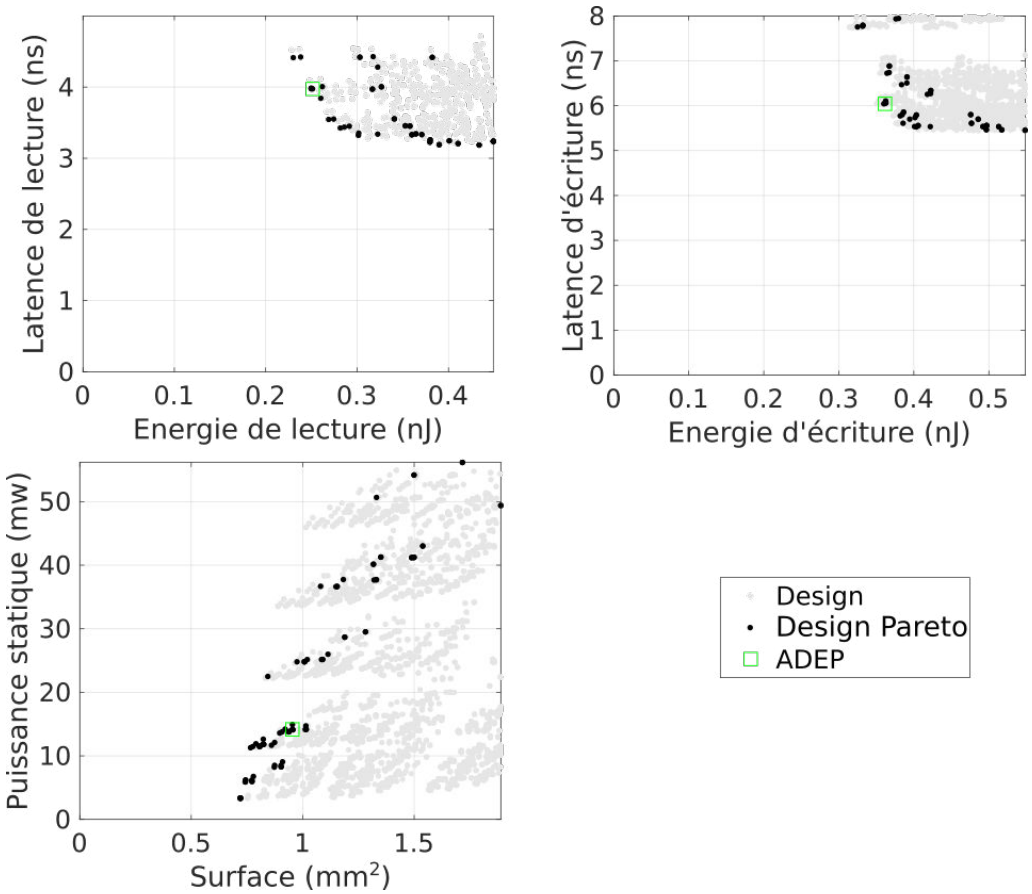


FIGURE 5.9 – Différentes visualisations de l'espace d'exploration. Les points noirs sont les points Pareto de la Figure 5.8.

il est possible après avoir identifié la meilleure configuration de trouver une configuration plus lente mais permettant d'économiser plus d'énergie dynamique ou statique. Ce cas est illustré par la Figure 5.10 et le tableau 5.3.

La meilleure configuration (notée A) est extraite selon le critère ADEP. La seconde configuration (notée B) à un ADEP plus élevé que A à cause de ses latences d'accès plus importantes. Néanmoins, elle permet de gagner 5% en énergie dynamique et 58% en énergie statique.

Cet exemple montre que le choix d'une configuration de cache n'est pas trivial et que, même après exploration, certains critères peuvent être adoucis pour une recherche plus exhaustive. Enfin, les résultats dépendent également de la perspective de l'analyse. La meilleure configuration de cache ne sera pas la même si l'on s'intéresse uniquement à l'efficacité énergétique de la mémoire cache ou si l'on considère une architecture complète.

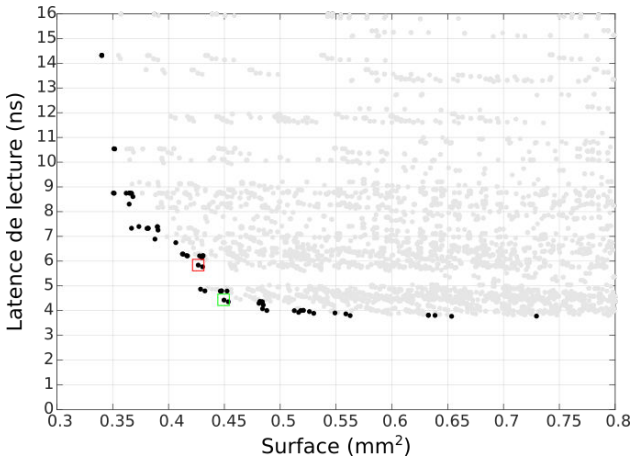


FIGURE 5.10 – Latence de lecture des configurations A et B en fonction de la surface de la mémoire cache

	Config. A	Config. B	B vs A
Latence lecture	4.42	5.83	+24%
Latence écriture	6.00	7.34	+18%
Énergie lecture	0.25	0.24	−5%
Énergie écriture	0.29	0.29	−4%
Surface	0.45	0.42	−5%
Puissance stat.	4.44	2.82	−58%
ADEP	2.53	2.93	+13.6%

Tableau 5.3 – Détail des configurations A et B

5.3.5 Évaluation de l'efficacité énergétique des choix de conception d'une mémoire cache

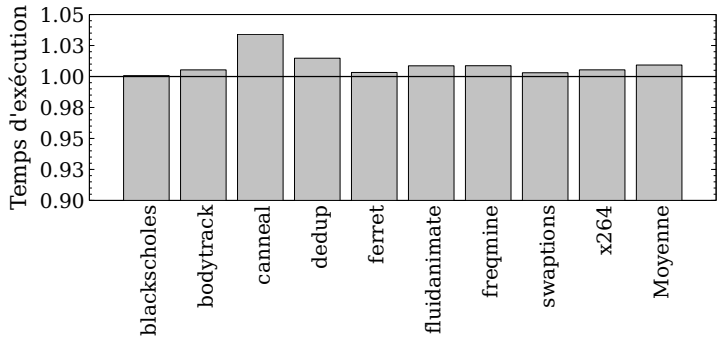
La notion d'efficacité énergétique est relative à la perspective de l'analyse. Certains travaux [63, 118, 121] analysent l'efficacité énergétique d'un cache en prenant uniquement en compte le niveau de mémoire cache où la NVM a été intégrée. Cette approche ne tient pas compte de la hiérarchie mémoire complète qui est impactée par ce changement de technologie.

Comme la STT-MRAM est plus lente que la SRAM, l'exécution des applications peut être ralentie. L'énergie statique de l'architecture, qui est fonction du temps [2], est donc augmentée. En se focalisant uniquement sur le niveau de cache concerné par le changement de technologie, l'énergie statique considérée est minime car la STT-MRAM la réduit drastiquement. Lorsque l'on considère toute la hiérarchie mémoire, on prend alors en compte l'augmentation de l'énergie statique des autres caches qui utilisent la SRAM. Dans ce cas

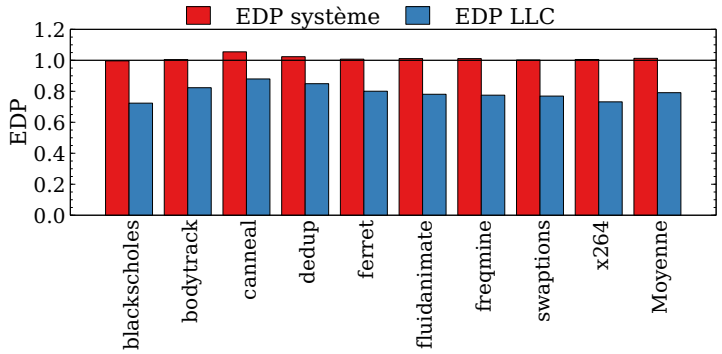
2. On s'abstrait ici de la température que l'on considère constante.

de figure, cette augmentation peut être plus importante.

La perspective de la mémoire cache seulement engendre parfois des observations différentes qu'une perspective globale. Pour illustrer cette affirmation, nous proposons d'analyser les configurations de cache A et B évoquées dans la section 5.3.4. Pour rappel, la configuration A est la meilleure configuration de cache selon le critère ADEP pour un cache de $2Mo$ 16 associatif. La configuration B est une configuration alternative qui augmente légèrement les latences mais diminue l'énergie dynamique et statique. Intégrons ce cache dans un architecture au niveau du L2 et regardons les temps d'exécution et l'efficacité énergétique selon les deux perspectives. Les résultats sont visibles sur les Figures 5.11a et 5.11b.



(a) Temps d'exécution



(b) Energy-Delay-Product

FIGURE 5.11 – Temps d'exécution et EDP des configurations A et B. La configuration B est normalisée aux résultats de la configuration A

La Figure 5.11a montre le temps d'exécution de la configuration B normalisé à celui de la configuration A. Il est sensiblement identique et la pénalité moyenne est de 1%. Le taux de *miss* sur les niveaux de caches L1 est très faible (3.5% en moyenne), l'activité du niveau de L2 est donc minime.

Lorsque l'on considère l'architecture en entier, la configuration B offre un EDP légère-

ment supérieur à la configuration A de 1%. Si l'on considère le cache L2 uniquement, la configuration B montre de meilleurs résultats d'EDP pour toutes les applications. L'écart est en moyenne de 20%. Il s'explique par le gain en énergie statique de la configuration B.

5.4 Analyse des différentes opérations d'écriture sur les caches

Sur le cache de dernier niveau, les opérations d'écriture sont divisées en deux catégories : les *Write-Back* et les *Write-Fill*. Le *Write-Back* est une écriture venant d'un niveau de cache inférieur. Le *Write-Fill* est une écriture venant d'un niveau de cache supérieur. Ces opérations sont représentées sur la Figure 5.12.

La transaction (1) est un *Write-Back* venant du niveau L2 pour la donnée X. Dans ce cas de figure, X est écrit dans la ligne de cache correspondante (transaction (2a)). Un *Write-Back* est éventuellement généré par le LLC en direction de la mémoire principale (transaction (2b)). Cela arrive si la donnée D qui est enlevée de la ligne pour stocker X a été modifiée et doit être sauvegardée.

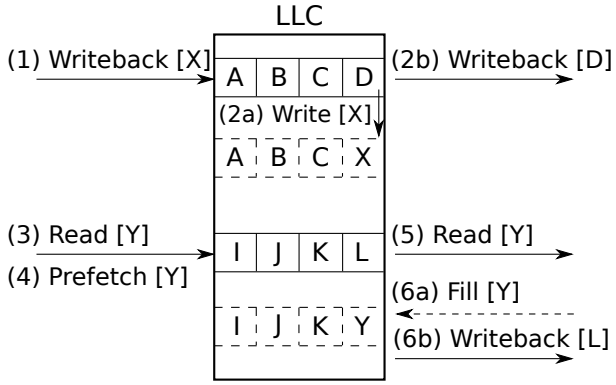


FIGURE 5.12 – Opérations d'écriture sur le cache de dernier niveau (LLC)

Pour les requêtes (3) et (4), qui correspondent respectivement à une lecture et un *prefetch*, la donnée Y qui est demandée n'est pas dans le cache. On a donc un *miss* qui génère une transaction à la mémoire centrale pour ramener la donnée Y et l'écrire dans le cache. Cette opération représente le *Write-Fill*. De la même manière que la transaction (2b), un *Write-Back* peut être généré par le LLC si la donnée L qui est ici supprimée de la ligne de cache doit être sauvegardée.

Une fois cette différenciation faite, une question importante est de savoir si les *Write-Back* et les *Write-Fill* ont le même impact sur les performances du système. Pour répondre à cette question, nous considérons 5 applications de la suite SPEC CPU2006 qui ont dif-

férentes répartitions entre les écritures de type *Write-Back* et *Write-Fill* au niveau du LLC (Figure 5.13a). Les applications *gcc* et *xalancbmk* ont environ 50% de *Write-Back* et *Write-Fill*. *libquantum* et *sphinx3* sont majoritairement dominées par les *Write-Fill*. Enfin, *perlbench* affiche le schéma inverse et contient plus de *Write-Back*.

Afin d'évaluer l'importance de ces deux types d'écriture sur le LLC, nous proposons de modifier les latences d'écriture de ces opérations et d'évaluer leur impact individuellement et conjointement. Considérons un scénario de référence avec un cache utilisant de la STT-MRAM. Pour évaluer l'impact des *Write-Back*, nous modifions la latence pour la rendre nulle tout en laissant la latence des *Write-Fill* à leur valeur par défaut. Pour évaluer le *Write-Fill*, nous inversons ces latences. Enfin, nous affectons une latence nulle aux deux opérations en même temps. Cela permet d'évaluer la borne maximale que l'on peut théoriquement atteindre si l'on supprime toutes les écritures. Cela permet également de voir si les opérations de *Write-Back* et *Write-Fill* n'ont pas d'effet de bords l'une sur l'autre. Le tableau 5.4 résume la configuration mise en place au sein du framework ChampSim.

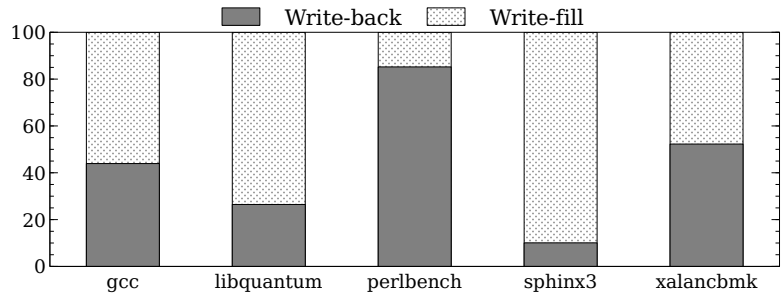
	T_{WB}	T_{WF}
Référence	STT-MRAM	STT-MRAM
WB = 0	0	STT-MRAM
WF = 0	STT-MRAM	0
WB = WF = 0	0	0
SRAM	SRAM	SRAM

Tableau 5.4 – Configuration des latences de *Write-Back* et *Write-Fill*

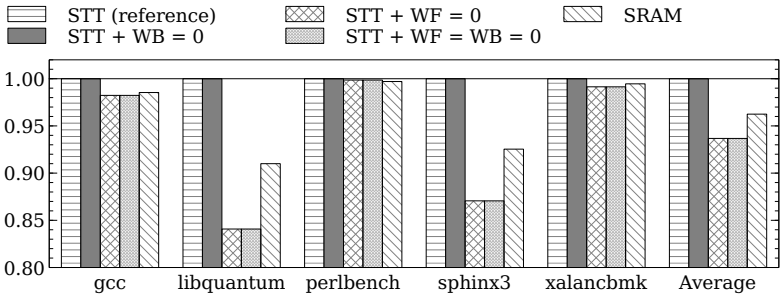
On considère une architecture avec une hiérarchie de cache à trois niveaux. La STT-MRAM est utilisée au niveau du LLC. Le scénario de référence est celui où les latences des *Write-Back* et *Write-Fill* n'ont pas été modifiées. La métrique de performance choisie est l'IPC. Les résultats sont donnés par la Figure 5.13b.

Sur cette figure, WB et WF dénotent respectivement les latences de *Write-Back* et *Write-Fill* du tableau 5.4. Dans le scénario de référence, le LLC a une taille de $2Mo$ et la latence pour les écritures est de 38 cycles. Les résultats de la Figure 5.13b sont normalisés à cette référence. Pour le scénario du LLC en SRAM, les latences sont de 20 cycles.

Lorsque $WB = 0\,cycle$ et que le *Write-Back* n'a aucun impact sur les performances, le temps d'exécution est le même que la configuration STT-MRAM de référence. Quand $WF = 0\,cycle$, le temps d'exécution est réduit d'un facteur $0.93\times$ en moyenne, et jusqu'à $0.84\times$ pour l'application *libquantum*. Enfin, lorsque les latences des WB et WF sont misent à zéro, le temps d'exécution est identique au cas où seule la latence des WF est zéro. Cela



(a) Distribution des *Write-Back* et *Write-Fill*



(b) Effets des *Write-Back* et *Write-Fill* sur les performances normalisés au scénario de référence

FIGURE 5.13 – Distribution des écritures en fonction des applications et résultats de performance

indique qu'il n'a pas d'effet de bord entre les deux types d'écriture. Les gains en performance sont particulièrement visibles sur les applications qui ont un plus grand nombre de *Write-Fill* que de *Write-Back*, comme *libquantum* ou *sphinx3*. Néanmoins, même pour une application avec plus de *Write-Back* que de *Write-Fill* comme *perlbench*, les résultats montrent que $WB = 0\,cycle$ n'a aucun impact sur les performances.

Ces résultats montrent que seuls les *Write-Fill* ont un impact significatif sur l'IPC. En effet, une opération venant d'un niveau de cache inférieur dans la hiérarchie mémoire ne nécessite pas un traitement immédiat ni même réponse de la part du LLC. De fait, le cœur n'est pas bloqué dans ses calculs. A l'inverse, un *Write-Fill* apparaît lors d'un cache *miss*, ce qui signifie que le cœur a besoin d'une donnée pour continuer l'exécution de l'application. Tant que la donnée n'est pas disponible, l'exécution peut être bloquée si les instructions suivantes dépendent de cette donnée.

L'analyse ci-dessus montre que l'on doit principalement se concentrer sur la diminution des opérations de *Write-Fill* si l'on veut réduire la pénalité de latence et améliorer les performances du système [84]. A notre connaissance, seuls Komalan et al. [55] ont également constaté ce fait et l'ont exploité au niveau de la mémoire cache L1 pour diminuer le

nombre d'écritures. Les travaux précédents attaquent le problème des écritures des NVM en voulant réduire le nombre de *Write-Back*, ou en essayant de les exécuter en parallèle pour gagner du temps.

De plus, on a vu qu'il n'existe pas d'effet de bord entre les *Write-Back* et les *Write-Fill*. Il est donc possible de supprimer des *Write-Fill* sans impact sur les *Write-Back*. Les résultats de la technologie SRAM permettent également de voir qu'une diminution drastique des *Write-Fill* pourrait donner de meilleurs résultats.

Les requêtes de *Write-Fill* sont directement dépendantes du nombre de *miss*. Une réduction de ces *miss* diminuera alors le nombre d'écritures sur le LLC. Nous avons vu au chapitre 2 que la politique de remplacement de la mémoire cache à un impact direct sur ces évènements. Nous avons également évalué différentes stratégies et nous avons trouvé que Hawkeye est la plus performante. Par la suite, on considère l'utilisation de cette politique de remplacement pour diminuer le nombre d'écritures sur le cache de dernier niveau.

5.5 Choix du niveau de mémoire cache pour intégrer la technologie STT-MRAM

5.5.1 Impact du processeur sur les latences de la mémoire cache

Un des problèmes majeurs concernant les NVM est la latence des opérations de lecture et d'écriture. Notre but ici n'est pas de proposer une optimisation de circuit mais d'identifier un ou des paramètres architecturaux ayant un impact significatif sur l'efficacité des NVM lors de leur intégration. Nous nous focalisons sur la fréquence du processeur.

Considérons les latences d'accès à une mémoire cache en termes de cycles. La durée d'un cycle en secondes dépend de la fréquence du processeur. De hautes fréquences impliquent un temps de cycle très court là où de basses fréquences allongent la durée d'un cycle. Pour une configuration de cache donnée, deux fréquences de fonctionnement donnent pour une même opération deux latences différentes en termes de cycles.

Considérons une latence de lecture d'un cache de $2ns$. Une fréquence de $1.0GHz$ donne une durée d'accès en cycle de $1ns$ et une fréquence de $2GHz$ une durée de $0.5ns$. On obtient ainsi une latence de 2 cycles à $1GHz$ et 4 cycles à $2GHz$, soit un écart de 50%.

On se propose de prendre une configuration de cache L1 et d'étudier ses latences d'accès avec la SRAM et la STT-MRAM. Le cache est 4 associatif et fait $32Ko$. La technologie considérée est $45nm$. Ces valeurs sont extraites de la configuration d'une carte ARM Odroid-XU4 [39]. Nous utilisons NVSim pour extraire les latences d'accès, puis nous transformons les latences (données en nanosecondes) en cycles.

Les caches sont détaillés dans le tableau 5.5. Les résultats sont donnés par la Figure 5.14.

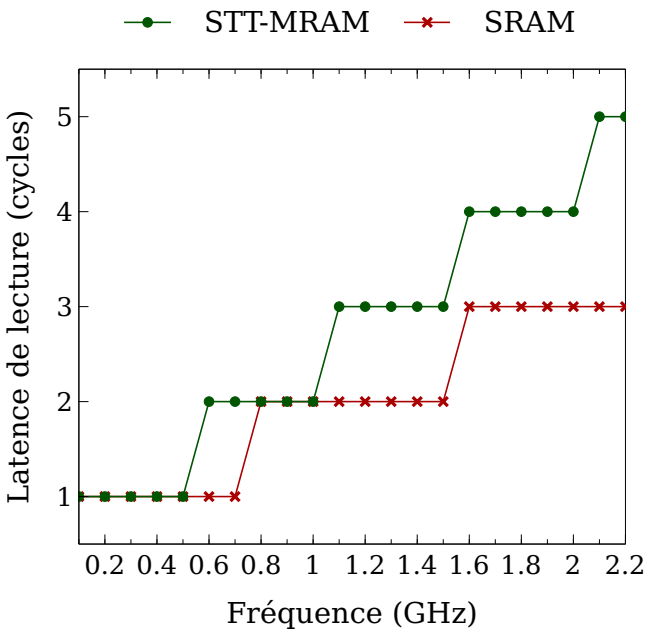


FIGURE 5.14 – Évolution de la latence de lecture d'un cache L1 de $32Ko$ en fonction de la fréquence du processeur pour les technologies SRAM et STT-MRAM [82]

	SRAM	STT-MRAM	Différence VS SRAM
Latence lecture	1.31ns	1.96ns	+ 33%
Latence écriture	1.31ns	10.94ns	+ 88%
Énergie lecture	24pJ	109pJ	+ 78%
Énergie écriture	6pJ	174pJ	+ 96%
Énergie statique	41.8mW	6.8mW	- 514%
Surface	0.091mm2	0.116mm2	+ 21%

Tableau 5.5 – Caractéristiques complètes d'un cache L1 $32Ko$ 4 associatif

Dans la plage de fréquences $[0.1, 0.5]GHz$ et $[0.8, 1.0]GHz$, la latence de lecture de la SRAM et la STT-MRAM est identique, soit 1 et 2 cycles respectivement. Dans les autres cas, la latence diffère de 1 cycle jusqu'à $2.1GHz$ ou l'écart est de 2 cycles. A partir de ces résultats, un choix de fréquence de fonctionnement de $1GHz$ semble intéressant pour la latence de lecture au niveau de la mémoire cache L1.

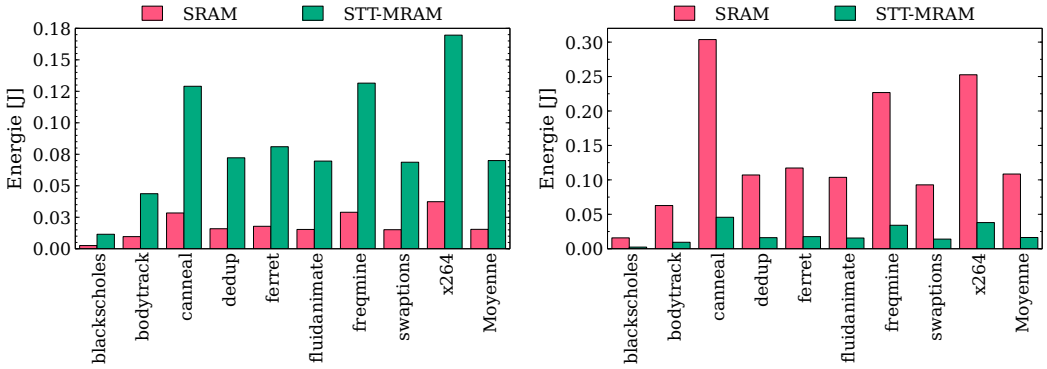
Une spécificité des architectures ARM est d'avoir un cache L1 d'instruction en lecture seule. L'intégration de STT-MRAM à ce niveau ne causera pas de pénalité sur le temps d'exécution puisque la latence est la même que la SRAM. L'énergie statique sera dimi-

nuée grâce aux propriétés intrinsèques de la STT-MRAM. En revanche, la consommation dynamique augmente. Une lecture et une écriture nécessitent respectivement 78% et 96% plus d'énergie avec de la STT-MRAM.

L'architecture simulée ici comprend un cœur ARM Cortex-A15 avec des caches L1-I/D de $32Ko$ et un cache L2 de $1Mo$. La fréquence de cœur est fixée à $1GHz$. Nous intégrons la STT-MRAM sur le cache L1-I. Notre référence est un scénario avec de la SRAM pour toute la hiérarchie mémoire. Dans cette section, on considère le framework MAGPIE comme outil de simulation.

Évolution de la consommation au niveau de la mémoire cache

Grâce à l'adaptation de la fréquence du processeur, les temps d'exécution obtenus sont identiques avec les deux technologies.



(a) Évolution de l'énergie dynamique (b) Évolution de l'énergie statique

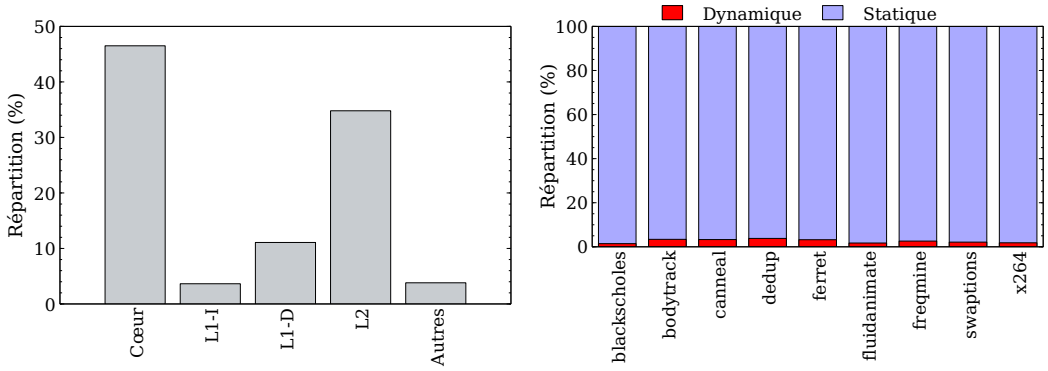
FIGURE 5.15 – Évolution de l'énergie dynamique et statique sur un cache L1 d'instruction utilisant de la STT-MRAM. Les résultats sont normalisés au scénario SRAM

Les figures 5.15a et 5.15b montrent l'évolution de l'énergie statique et dynamique du niveau L1. On voit que l'énergie dynamique de la mémoire cache STT-MRAM est augmentée par rapport à la mémoire cache SRAM, de $4.5\times$ en moyenne. A l'inverse, l'énergie statique de la mémoire cache STT-MRAM est toujours inférieure et équivaut à $0.15\times$ celle de la mémoire cache SRAM.

Impact sur l'architecture

L'impact sur la hiérarchie mémoire est cependant limité. En effet, l'estimation donnée par MAGPIE montre que le cache L1 d'instruction ne représente que 3% de la consommation totale et 7% de la consommation de la hiérarchie mémoire. Cette répartition de la consommation est illustrée sur la Figure 5.16a.

Le cache L2 représente 35% de la consommation totale de l'architecture et 70% de la consommation de la hiérarchie mémoire. De plus, sa consommation énergétique est largement dominée par l'énergie statique (Figure 5.16b). Ce cache est donc un bon candidat pour intégrer de la STT-MRAM. Cependant, sa latence d'accès est susceptible de diminuer le temps d'exécution. En effet, la différence de latence entre SRAM et STT-MRAM pour un cache de $1Mo$ est trop importante pour que les lectures s'exécutent dans le même nombre de cycles. Dans cet exemple, les accès au L2 prennent 17 cycles avec de la SRAM et 25 et 28 cycles respectivement pour les lectures/écritures avec de la STT-MRAM.



(a) Répartition de la consommation entre composants

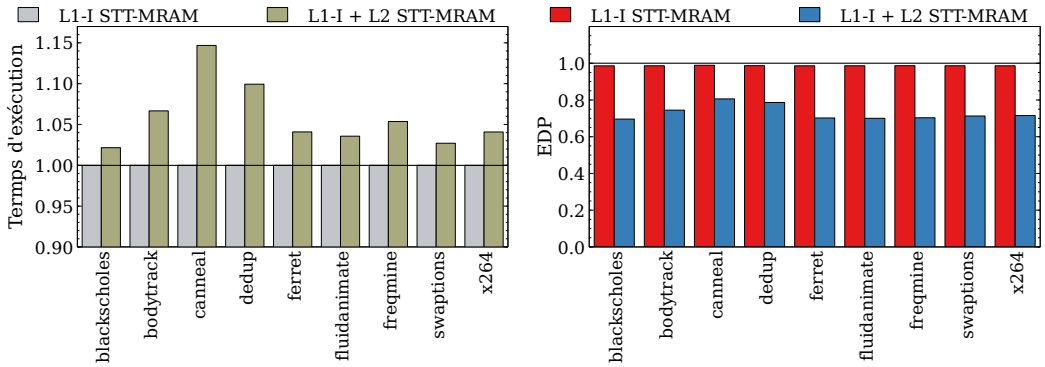
(b) Répartition énergie dynamique/statique sur le cache L2

FIGURE 5.16 – Répartition de la consommation énergétique sur l'architecture et sur le cache L2

Second scénario : L1-I et L2 en STT-MRAM

Pour ce second scénario, les résultats de la Figure 5.17a montrent que l'intégration de STT-MRAM au niveau L2 augmente de le temps d'exécution. Au maximum, la pénalité est de $1.14\times$. Néanmoins, du fait de sa contribution à hauteur de 35% de la consommation totale et de sa forte domination par l'énergie statique, l'introduction de STT-MRAM a un effet positif sur l'Energy-Delay-Product. Les résultats de la Figure 5.17b montrent que l'EDP est réduit de 27% en moyenne par rapport à une configuration « full SRAM ».

Malgré une optimisation architecturale permettant à la STT-MRAM de concurrencer la SRAM au niveau de la mémoire cache L1 d'instruction, l'effet est limité par la faible contribution du L1 à la consommation totale de la puce. Le niveau L2 est un meilleur candidat du fait de sa consommation qui représente 35% du total, et de son énergie statique qui est grandement diminuée par la STT-MRAM. Nous considérons maintenant uniquement le niveau L2 comme composant susceptible d'utiliser de la STT-MRAM.



(a) Temps d'exécution (b) Efficacité énergétique

FIGURE 5.17 – Temps d'exécution et efficacité énergétique d'une architecture monocœur introduisant de la STT-MRAM sur le cache L1 d'instruction et le cache L2. Les résultats sont normalisés au scénario « full SRAM »

5.5.2 Étude avec plusieurs fréquences de fonctionnement

On considère à présent uniquement la cache L2 comme susceptible d'utiliser de la STT-MRAM. De la même manière que dans la section précédente, on s'intéresse à l'effet de la fréquence du processeur sur les latences d'accès. Après une exploration NVSim pour un cache L2 de $2Mo$ 16 associatif, on obtient les caractéristiques données par le tableau 5.6.

	SRAM	STT-MRAM
Lecture	16.421ns	24.404ns
Écriture	-	27.361ns

Tableau 5.6 – Latences d'accès d'une mémoire cache L2 pour les technologies SRAM et STT-MRAM

Contrairement au cache L1, les latences de la STT-MRAM sont trop éloignées pour pouvoir concurrencer la SRAM même en modifiant la fréquence. Cependant, la fréquence à tout de même un impact intéressant sur les latences. Sur le tableau 5.7, on voit que certaines fréquences offrent un écart avec la SRAM inférieur à des fréquences supérieures. C'est le cas pour $0.8GHz$ et $1.3GHz$, ou $1.6GHz$ et $2.0GHz$.

Une fréquence de fonctionnement plus faible diminue la latence de la mémoire et en même temps augmente le temps de calcul du point de vue du cœur. A l'inverse, une haute fréquence accélère le calcul pour le cœur mais est beaucoup ralenti par la hiérarchie mémoire. On cherche ici à identifier un compromis entre les deux, s'il existe. En trouvant une ou des fréquences adéquates, on peut ainsi calibrer la fréquence du cœur pour une efficacité énergétique maximale.

Pour identifier les fréquences intéressantes, nous posons la contrainte suivante :

$$R_{diff} \leq 50\% \bigvee W_{diff} \leq 65\% \,, \tag{5.1}$$

où R_{diff} et W_{diff} représentent respectivement les différences de latences en lecture et en écriture de la STT-MRAM par rapport à la SRAM. Les fréquences retenues sont surlignées dans le tableau 5.7.

Nous simulons la même architecture que précédemment : un cœur ARM Cortex-A15, deux caches L1 de $32Ko$ et un cache L2 de $1Mo$. On modifie uniquement la fréquence du cœur et on mesure l'efficacité énergétique de la plateforme. Les résultats sont donnés sur la Figure 5.18. Pour chaque application, l'EDP est normalisé à l'EDP à $0.5GHz$, soit la plus basse fréquence utilisée.

Le meilleur EDP est obtenu avec la fréquence maximale $2.0GHz$. Cette fréquence est celle qui a la plus grande différence de latence avec la SRAM, respectivement de 48.5% en lecture et 66.7% en écriture. Cela suggère que malgré des délais élevés, l'efficacité énergétique est plus fonction du cœur que de la mémoire cache. En effet, le L2 en STT-MRAM ne représente que 3.5% de la consommation globale. On note cependant que certaines fréquences obtiennent un meilleur score d'EDP que de plus hautes fréquences. Pour toutes les applications excepté *ferret* et *freqmine*, la configuration $1.3GHz$ est plus efficace énergétiquement que $1.6GHz$. C'est également le cas pour la configuration $1.0GHz$, qui a un meilleur score d'EDP que la configuration $1.1GHz$. Cela montre que dans certains cas, l'augmentation de la latence de la hiérarchie mémoire a un plus grand impact que l'augmentation de la rapidité de calcul. Ce cas est bien visible avec l'application *x264* : la configuration $1.0GHz$ est plus efficace énergétique que $1.1GHz$, $1.3GHz$, $1.6GHx$, et est proche de $1.9GHz$.

Cette étude aura permis de montrer que, contrairement au cache L1, la fréquence de fonctionnement maximale doit être sélectionnée lorsque l'on vise un cache de niveau L2 au plus pour intégrer de la STT-MRAM.

5.6 Optimisation de conception d'un cache de type STT-MRAM

Pour l'intégration de la technologie STT-MRAM dans une mémoire cache L2, la fréquence maximale semble être le meilleur choix en matière d'efficacité énergétique. On s'intéresse maintenant à l'optimisation d'une mémoire cache pour maximiser l'EDP. Nous avons vu précédemment que notre méthode d'exploration pour la conception de mémoires caches propose 6 optimisations possibles (Tableau 5.2). Nous utilisons cet outil pour extraire et visualiser les différentes configurations. Les détails sont donnés par le Tableau 5.8.

Fréq.	SRAM	R. STT-MRAM	W. STT-MRAM	R. diff	W. diff
0.5	9	13	14	-44.4%	-55.6%
0.6	10	15	17	-50%	-70%
0.7	12	18	20	-50%	-66.7%
0.8	14	20	22	-42.9%	-57.1%
0.9	15	22	25	-46.7%	-66.7%
1.0	17	25	28	-47.1%	-64.7%
1.1	19	27	31	-42.1%	-63.2%
1.2	20	30	33	-50%	-65%
1.3	22	32	36	-45.5%	-63.6%
1.4	23	35	39	-52.2%	-69.6%
1.5	25	37	42	-48%	-68%
1.6	27	40	44	-48.1%	-63%
1.7	28	42	47	-50%	-67.9%
1.8	30	44	50	-46.7%	-66.7%
1.9	32	47	52	-46.9%	-62.5%
2.0	33	49	54	-48.5%	-63.6%

Tableau 5.7 – Effet de la fréquence du processeur sur la latence d'une mémoire cache L2

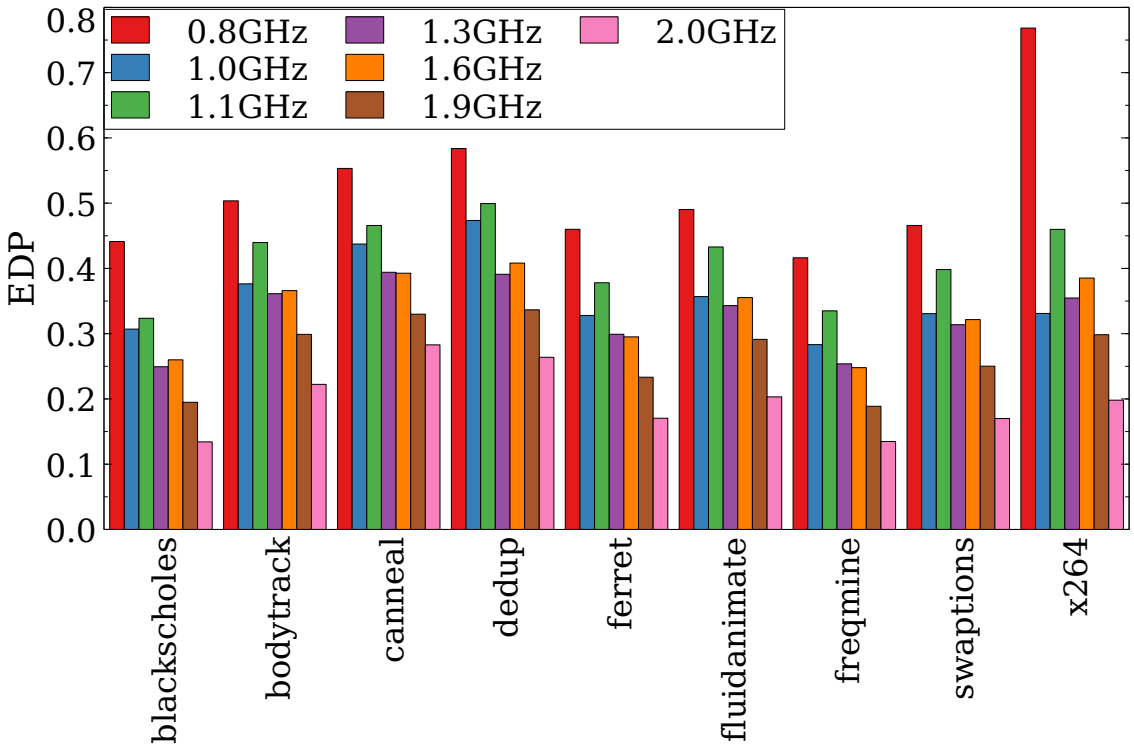


FIGURE 5.18 – Effet de la fréquence du processeur sur l'EDP. Les résultats sont normalisés à la fréquence $0.5GHz$.

La Figure 5.19 permet d'observer les disparités entre toutes ces configurations. On voit qu'une optimisation AD^{2}P favorise les latences au détriment de la surface. C'est cette configuration qui obtient les latences les plus faibles. En contrepartie, l'énergie dynamique et statique ainsi que la surface sont les plus élevés. A l'inverse, la configuration ADE^{2}P qui favorise l'énergie obtient les latences les plus élevées mais des coûts énergétiques faibles. Enfin, on voit que les autres configurations sont un compromis entre la surface, les latences et l'énergie. On note également que la configuration ADEP et ReadEDP sont confondues. Nous allons évaluer toutes ces possibilités avec MAGPIE sur une architecture monocœur avec une fréquence de $2.0GHz$.

	Latence		Énergie			
	Lecture	Écriture	Lecture	Écriture	Puissance	Surface
ADP	3.26	5.91	0.172	0.297	1.41	0.94
AD^{2}P	2.96	5.75	0.201	0.322	2.54	1.00
ADEP	3.52	6.38	0.119	0.274	1.37	0.92
ADE^{2}P	4.39	7.08	0.123	0.244	0.83	0.89
ReadEDP	3.52	6.38	0.119	0.274	1.37	0.92
WriteEDP	3.15	5.87	0.161	0.262	1.40	1.24

Tableau 5.8 – Détails des configurations de cache L2 en fonction de leur optimisation. Les latences sont en ns, l'énergie dynamique et statique en nJ et mW, et la surface en mm^2.

Les résultats présentés sur la Figure 5.20 sont normalisés à la configuration ADP. En moyenne, les configurations ADEP, ReadEDP et WriteEDP sont égales à la configuration ADP. C'est la configuration AD^{2}P qui fournit la meilleure efficacité énergétique. On observe cependant que la différence est faible, à peine 2%. L'application *dedup*, contenant beaucoup d'accès au cache L2, est toutefois représentative des différences qui existent entre les optimisations de cache. La stratégie AD^{2}P qui favorise la latence au détriment de l'énergie montre un meilleur EDP grâce à une pénalité de latences plus faible que les autres configurations.

La faible différence qui existe entre ces configurations montre que les choix d'optimisation pour cette architecture monocœur avec deux niveaux de cache ne sont pas déterminant sur le plan de l'efficacité énergétique. De fait, nous choisissons pour la suite un compromis entre les différentes métriques et optons pour le critère ADEP (*Area-Delay-Energy Product*).

5.7 Politiques de remplacement et gains énergétiques

Dans le chapitre 2, nous avons comparé à l'aide du simulateur ChampSim et de la suite SPEC CPU2006 les améliorations sur l'IPC de 6 politiques de remplacement : LRU,

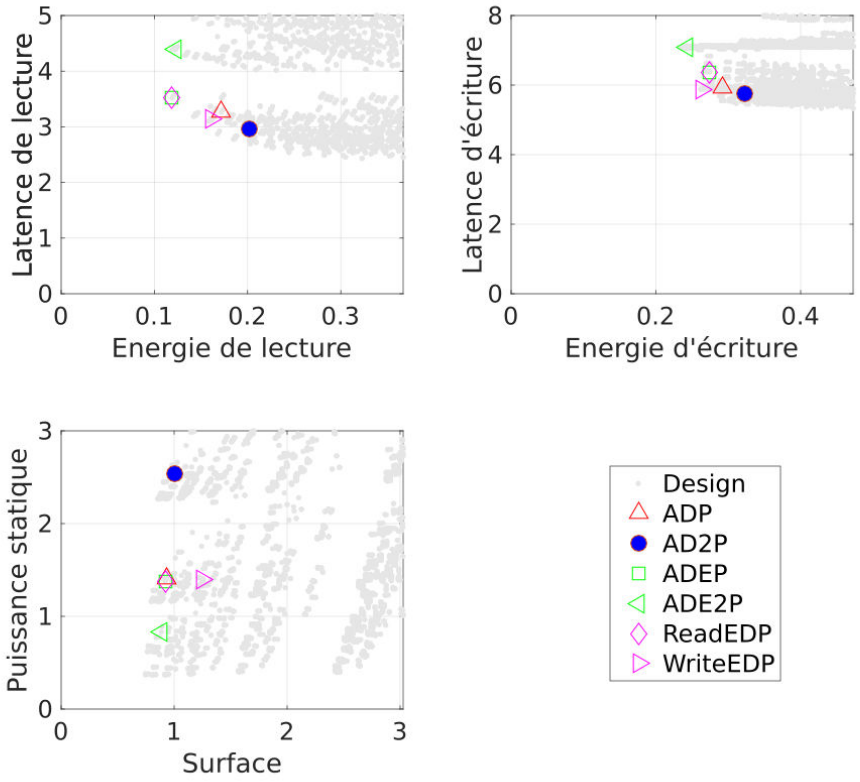


FIGURE 5.19 – Visualisation des caractéristiques des différentes configurations de caches explorées

Random, SRRIP, DRRIP, SHiP et Hawkeye. Notre évaluation est basée sur une plateforme monocœur avec un système de *prefetching* activé ou désactivé. Pour cette étude, nous utilisons l'environnement d'exploration basé sur ChampSim que nous avons présenté au chapitre 4. Ce dernier nous permet d'effectuer rapidement une étude de l'impact énergétique des stratégies de remplacement.

La Figure 5.21a montre l'amélioration de l'IPC par rapport à l'amélioration énergétique (normalisés à la LRU) lorsque le prefetching est désactivé. On observe une tendance linéaire : quand l'IPC est amélioré, la consommation énergétique l'est également. En effet, si l'exécution est plus rapide, l'énergie statique diminue. De plus, la réduction du nombre de *miss* diminue l'énergie dynamique de la mémoire cache.

La Figure 5.21b montre que le système de *prefetching* a un impact plus faible que précédemment sur la consommation énergétique. De plus, la tendance linéaire observée précédemment est beaucoup moins prononcée. Cela vient du fait que le système de *prefetching*

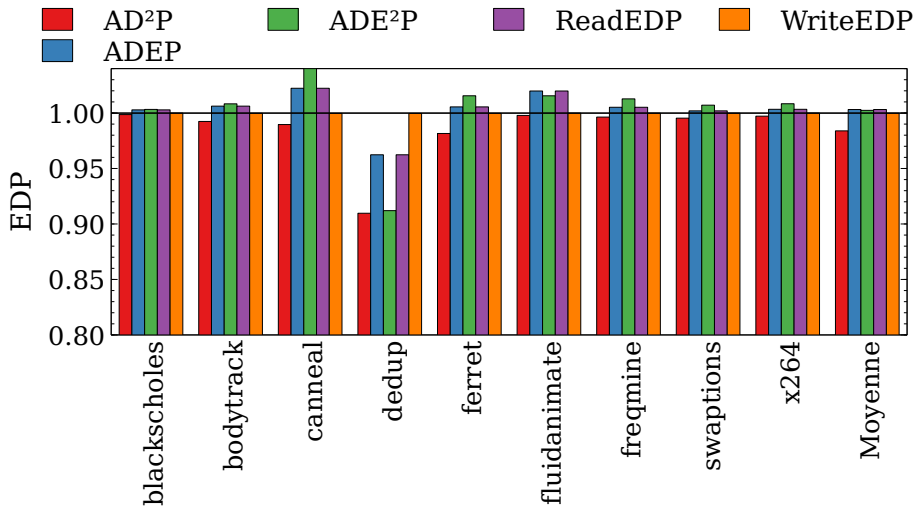


FIGURE 5.20 – EDP de l'architecture pour les différentes optimisations de cache normalisé à la configuration ADP

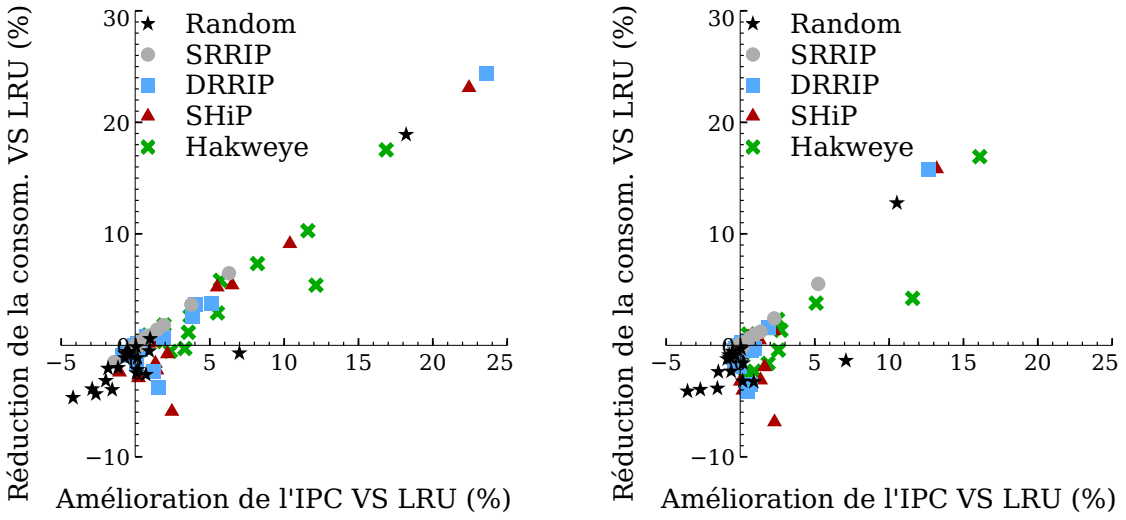
augmente de manière significative l'énergie dynamique de la mémoire cache par l'activité supplémentaire qu'il génère. A titre d'exemple, le nombre de requêtes reçues par le LLC pour l'application *mcf* augmente de 40%, et ce quelque soit la politique de remplacement considérée. La consommation statique est réduite de 9.4% grâce à une exécution plus rapide, mais la consommation dynamique augmente de 37.2%. Ainsi, la consommation globale n'est réduite que de 5.5%.

5.8 Résumé

Ce chapitre a été consacré à l'étude des opportunités d'amélioration à un niveau architectural sur des caches utilisant de la STT-MRAM. Cette technologie ayant des propriétés différentes de la SRAM, comme sa densité ou ses fortes latences d'écriture, ces deux points ont permis de mettre à jour plusieurs possibilités d'optimisation.

Premièrement, la densité supérieure de la STT-MRAM permet augmenter la capacité de stockage de la mémoire cache. Cependant, cette densité ne s'utilise pas de la même manière qu'avec de la SRAM. Nous avons vu que les caractéristiques de latences des cellules MTJ entraînent un changement dans la conception des *bank* et des *array* de la mémoire. Sans ce changement, les latences de la STT-MRAM, déjà élevées par rapport à la SRAM, seraient augmentées.

Deuxièmement, nous avons présenté une méthode d'exploration architecturale au niveau des caches via l'outil NVSim. Nous avons montré qu'il existe un espace d'exploration conséquent et que le choix d'une configuration peut se faire selon plusieurs critères, no-



(a) Prefetching désactivé

(b) Prefetching activé

FIGURE 5.21 – Amélioration de l'IPC et de la consommation énergétique pour différentes politiques de remplacement sur 20 applications par rapport à la LRU

tamment en fonction du niveau de cache considéré. Nous avons également vu les limites de cette approche.

Troisièmement, nous avons vu que toutes les transactions d'écriture dans une hiérarchie mémoire n'ont pas le même effet sur un cache contenant de la STT-MRAM. Via l'utilisation du framework ChampSim, nous avons vu que les opérations de *Write-Fill* sont les plus pénalisantes en matière de latence. Afin de réduire au maximum ces écritures, nous allons considérer par la suite une politique de remplacement avancée sur le cache de dernier niveau.

Enfin, nous avons mené des expériences préliminaires sur l'impact *i*) de l'intégration de STT-MRAM et *ii*) des stratégies de remplacement sur la consommation énergétique. Nos premières conclusions indiquent qu'il est préférable de viser un cache de niveau 2 ou plus. L'optimisation de conception retenue est un compromis entre trois métriques que sont la surface, les latences et l'énergie. Enfin, les politiques de remplacement vont nous permettre grâce à une gestion plus fine de la mémoire de diminuer le nombre d'écritures sur les cellules MTJ et de diminuer la consommation énergétique.

Le chapitre suivant sera consacré à l'évaluation des opportunités d'optimisation au niveau architectural pouvant bénéficier à la technologie STT-MRAM.

How about a magic trick?

THE JOKER - THE DARK KNIGHT (2008)

Chapitre 6

Application d'optimisations architecturales sur un cache de dernier niveau à base de STT-MRAM

Le chapitre 5 a permis de montrer qu'il existe des opportunités d'amélioration de la mémoire cache lorsque celle utilise la technologie STT-MRAM. Nous avons vu qu'il existe

deux types d'écriture, *Write-Back* et *Write-Fill*, et que seuls les *Write-Fill* sont pénalisants. Les *Write-Fill* étant la conséquence des *miss*, une politique de remplacement avancée permettrait d'en diminuer le nombre. De plus, la densité supérieure de la STT-MRAM peut être utilisée intelligemment pour agrandir la capacité de stockage sans augmenter drastiquement les latences d'accès, ni les coûts énergétiques de lecture et d'écriture. Cela a également pour effet de diminuer le nombre de *miss*, et donc de diminuer les *Write-Fill*.

Dans ce chapitre, nous proposons de combiner ces deux optimisations en agrandissant la capacité de stockage de la mémoire cache de dernier niveau et en utilisant la politique de remplacement Hawkeye étudiée aux chapitres 2 et 5. Cette étape permet de confirmer ou d'infirmer la validité de nos hypothèses à l'échelle de systèmes complets. On évaluera dans un premier temps l'effet de ces optimisations architecturales sur un système monocœur, puis dans second temps sur un système multicœurs.

Pour ces deux types de plateforme, on s'intéressera aux performances, à la consommation énergétique et l'efficacité énergétique des architectures de caches proposées. Le principal enjeu ici est l'amélioration de l'efficacité énergétique, caractérisée par l'EDP. Cependant, les éventuelles pénalités en matière de temps d'exécution sont également un élément important.

Enfin, on s'intéressera à l'impact du design de la mémoire cache. En fonction de son optimisation, les changements proposés peuvent avoir un effet différent, ou moins important. On verra également que le choix de la perspective de lecture des résultats influe sur les conclusions quant à l'efficacité énergétique.

6.1 Configuration du cadre de simulation

Dans cette section, nous détaillons l'environnement de simulation et les modèles mémoires utilisés pour mettre en application nos propositions. Celles-ci sont mises en place au niveau de la mémoire cache de dernier niveau (LLC).

6.1.1 Caractéristiques technologiques et architecturales des mémoires

Latences

Pour les mémoires caches, nous considérons une technologie avancée de $22nm$. La température est fixée à $350K$. Nous utilisons la méthode d'exploration vue dans le chapitre 5 et sélectionnons la meilleure configuration selon le critère ADEP. Les résultats sont donnés dans le tableau 6.1.

La configuration de la mémoire principale est basée sur une fiche technique publi-

quement accessible de l'entreprise Micron Technology [103]. Pour les architectures monocœur, nous modélisons une mémoire de $4Go$ avec 1 DIMM, 8 *ranks*, 8 *banks* par *ranks* organisés avec 16×65536 colonnes de $64o$ chacune. Chaque *bank* contient $64Mo$ de données, chaque *rank* $512Mo$, le total faisant $4Go$. Pour les architectures multicœurs, nous ajoutons un DIMM supplémentaire avec les mêmes caractéristiques pour atteindre $8Go$. Les latences de la mémoire principale sont les suivantes : $tRP = tRCD = tCAS = 11\,cycles$, $tRAS = 28\,cycles$, $tRFC = 208\,cycles$ et $tCK = 1.25ns$.

	SRAM				STT-MRAM			
	2Mo	4Mo	8Mo	16Mo	2Mo	4Mo	8Mo	16Mo
Latence lecture [ns]	1.43	2.10	3.74	6.53	4.43	4.45	5.05	5.55
Latence écriture [ns]	-				6.00	6.05	6.31	6.52
Surface [mm^2]	1.52	3.02	5.78	11.21	0.45	0.81	1.54	2.99

Tableau 6.1 – Configuration de latence et de surface pour les caches SRAM et STT-MRAM

Modèles énergétiques

Le modèle énergétique des caches et de la mémoire principale est celui présenté à la section 4.3. Les coûts énergétiques d'accès à la mémoire sont donnés par le tableau 6.2.

RD	WR	REF	PRE	ACT	ACT_{BG}	T_{REF}
0.47nJ	0.47nJ	46.33nJ	0.22nJ	0.38nJ	0.027W	64ms

Tableau 6.2 – Configuration de la mémoire principale

6.1.2 Configuration du flot ChampSim

Nous utilisons ici le framework basé sur ChampSim et 20 traces d'applications de la suite SPEC CPU2006. Ces traces sont fournies avec le simulateur ChampSim. Elles ont été collectées sur machine x86 sans mécanisme de *prefetching* avec les outils Valgrind [74], SimPoint [38] et Pin [68]. La période d'initialisation est de 200 millions d'instructions, pour une nombre total d'instructions exécutées de 1 milliard. Pour les systèmes multicœurs, si un des cœurs a terminé l'exécution de l'application, celle-ci est de nouveau exécutée jusqu'à ce que tous les cœurs du système aient atteint 1 milliard d'instructions. L'activité supplémentaire générée par cette exécution n'est pas prise en compte dans les résultats finaux. Seuls compte les 800 millions d'instructions après la période d'initialisation. Les architectures explorées sont détaillées dans le tableau 6.3. Notre scénario de référence

monocœur comprend une hiérarchie de cache entièrement en SRAM avec un LLC de $2Mo$ 16 associatif. Dans le cas du multicœur, le cache de référence fait $4Mo$.

La latence de référence du LLC dans ChampSim, basée sur un processeur Intel-i7, est de 20 cycles pour un cache de $2Mo$ 16 associatif. Cela correspond à $5ns$ à $4GHz$. Pour calculer la latence des caches STT-MRAM, nous utilisons la formule suivante :

$$L_T = L_C + L_W = 5ns\,, \tag{6.1}$$

où L_T est la latence totale du LLC pour traiter une requête en provenance du niveau L2, L_C est la latence du LLC et L_W est la latence de transmission entre le L2 et le LLC. La latence effective d'un accès est donc la somme du délai de transmission et de l'accès en lui-même. Une simulation NVSim d'une mémoire cache de $2Mo$ 16 associatif donne une latence d'accès de $1.425ns$. La latence de transmission est donc

$$L_W = L_T - L_C = 5 - 1.425 = 3.575ns \tag{6.2}$$

Nous utilisons $L_W = 3.575ns$ comme une valeur d'*offset* pour calculer la latence L_T des caches en utilisant la Formule 6.1. L_C est obtenu via une simulation NVSim.

L1 (I/D)	32Ko, 8-way, LRU, Privé, 4 cycles			
L2	256Ko, 8-way, LRU, Unifié, 8 cycles			
L3	Taille/politique variable, 16-way, Partagé			
L3	2Mo	4Mo	8Mo	16Mo
L3 SRAM (latence)	20	23	30	41
L3 STT (latence, L/E)	33/39	33/39	35/40	37/41
CPU	1 ou 4 cœur(s), Out-of-Order, 4GHz			
Mémoire (taille/latence)	4Go ou 8Go, hit : 55 cycles, miss : 165 cycles			

Tableau 6.3 – Configuration de l'environnement de simulation

6.2 Évaluation des optimisations proposées pour un système monocœur

Nous évaluons dans cette section l'impact des optimisations architecturales présentées au chapitre 5. Dans un premier temps, nous évaluons l'impact de la capacité de stockage d'une mémoire cache de dernier niveau basée sur la technologie STT-MRAM. Dans un second temps, nous ajoutons la politique de remplacement Hawkeye. Enfin, nous comparons cette politique à la LRU. A moins que cela soit explicitement mentionné, tous les

résultats présentés dans cette section sont normalisés au scénario de référence SRAM.

6.2.1 Architectures de caches considérées

Nous fixons la limite suivant pour contrôler l'augmentation la surface de la mémoire cache de dernier niveau :

$$A_{sram} \geq A_{stt} \, , \tag{6.3}$$

où A_{sram} représente la surface de la mémoire cache de référence en SRAM, et A_{stt} la surface de la mémoire cache STT-MRAM. Cela permet de profiter de la densité de la STT-MRAM sans occuper plus de place sur la puce que le cache initial. [1]

Notre référence SRAM est le cache de $2Mo$. Ainsi, huit configurations de caches sont testées au niveau du LLC :

— $2Mo$ SRAM avec LRU et Hawkeye

— $2Mo$, $4Mo$ et $8Mo$ STT-MRAM avec LRU et Hawkeye

Ces configurations sont nommées de la manière suivante. Nous associons les préfixes T (Tiny), S (Small) et M (Medium) avec la technologie utilisée et la politique de remplacement considérée. Par exemple, *T_stt_hwk* est une configuration de $2Mo$ utilisant de la STT-MRAM et la politique de remplacement Hawkeye.

6.2.2 Impact de la capacité de la mémoire cache et de la technologie

Toutes les configurations présentées ici utilisent la LRU. Le haut de la Figure 6.1 montre l'amélioration du MPKI par rapport à la configuration de référence. On observe que la configuration *T_stt_lru* n'influence pas le MPKI car la capacité de stockage est identique à la référence. A l'inverse, une réduction du MPKI est visible avec les configurations *S_stt_lru* et *M_stt_lru*. Certaines applications ne sont pas sensibles à la capacité de stockage de la mémoire cache, comme *bwaves*, *libquantum* ou *milc*. D'autres comme *lbm* sont très sensibles, notamment à partir d'une taille de LLC de $8Mo$. Pour cette application, le MPKI est réduit d'un facteur $0.56\times$. Cela indique qu'une large portion des données nécessaires à l'application réside dans le cache.

Le bas de la Figure 6.1 montre les résultats d'IPC des configurations STT-MRAM. La configuration *T_stt_lru*, i.e., un remplacement de la SRAM par la STT-MRAM, est plus lente que la référence à cause des latences plus élevées de la STT-MRAM. La configuration *S_stt_lru* égale la SRAM et *M_stt_lru* la surpasse d'en moyenne $1.06\times$. Avec *S_stt_lru*, l'IPC est dégradé pour seize applications, tandis qu'avec *M_stt_lru* la pénalité est visible

1. Le cache STT-MRAM de $8Mo$ augmente de 0.99% la surface initiale. Nous considérons cela comme négligeable.

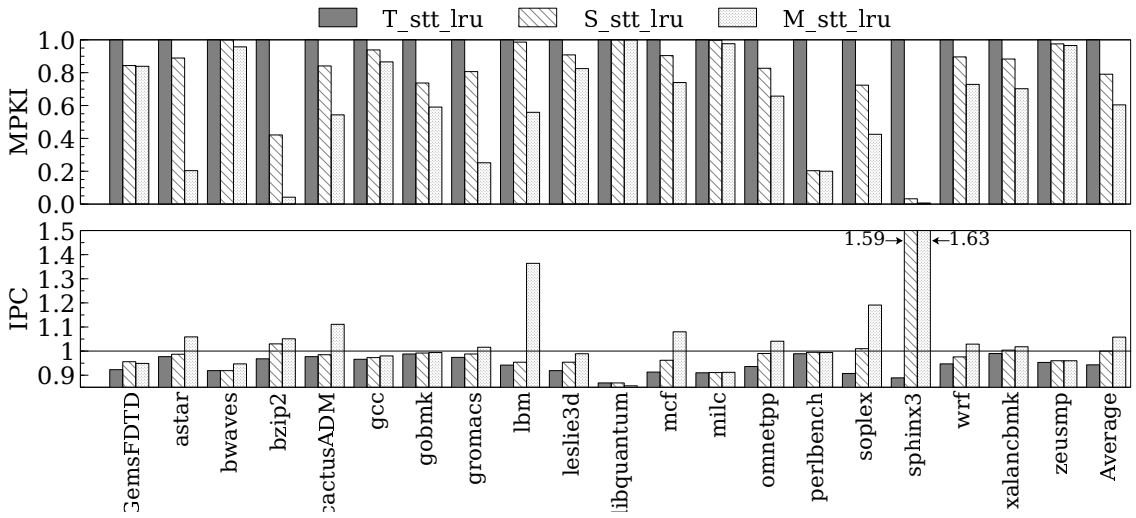


FIGURE 6.1 – MPKI (haut) et IPC (bas) avec LRU normalisé à *M_sram_lru*.

sur neuf applications. La performance de l'application *soplex* est corrélée au MPKI. En effet, il y a une tendance linéaire entre la réduction du MPKI et l'amélioration de l'IPC. A l'inverse, les applications *gobmk*, *gromacs* et *perlbench* montre une réduction importante du MPKI sans réel impact sur les performances. Cela est du au petit nombre de requêtes reçues par le LLC comparé aux autres applications. Réduire les *miss* n'est donc pas assez significatif pour améliorer les performances.

En moyenne, l'augmentation de la capacité de la mémoire cache montre que la STT-MRAM peut fournir les mêmes performances, voire mieux que la SRAM.

La Figure 6.2 montre les résultats de consommation énergétique (haut) et d'efficacité énergétique (bas). En moyenne, les configurations *T_stt_lru*, *S_stt_lru* et *M_stt_lru* diminuent respectivement la consommation énergétique par 10%, 14% et 16.6%. L'application *libquantum* est la seule application pour laquelle la consommation énergétique est augmentée. L'ajout de la STT-MRAM pour cette application diminue l'IPC d'environ 10% (Figure 6.1), ce qui se traduit par une exécution plus longue et une augmentation de l'énergie statique de l'architecture. Cette augmentation est suffisante pour annuler le gain de la STT-MRAM. A l'inverse, *sphinx3* montre un gain de performance important, et donc une diminution de la consommation énergétique.

L'efficacité énergétique est globalement corrélée à la diminution de la consommation. En moyenne, le gain par rapport à la SRAM est de 4.7%, 13.4% et 20.4% pour les configurations *T_stt_lru*, *S_stt_lru* et *M_stt_lru*. Certaines applications comme *bwaves*, *libquantum* ou *milc* dégradent l'efficacité énergétique comparé à la SRAM. Malgré une diminution de la consommation, ces applications souffrent d'une trop grosse dégradation de performance. Ces applications sont insensibles à la taille du LLC, et subissent par conséquent

l'augmentation des latences d'accès.

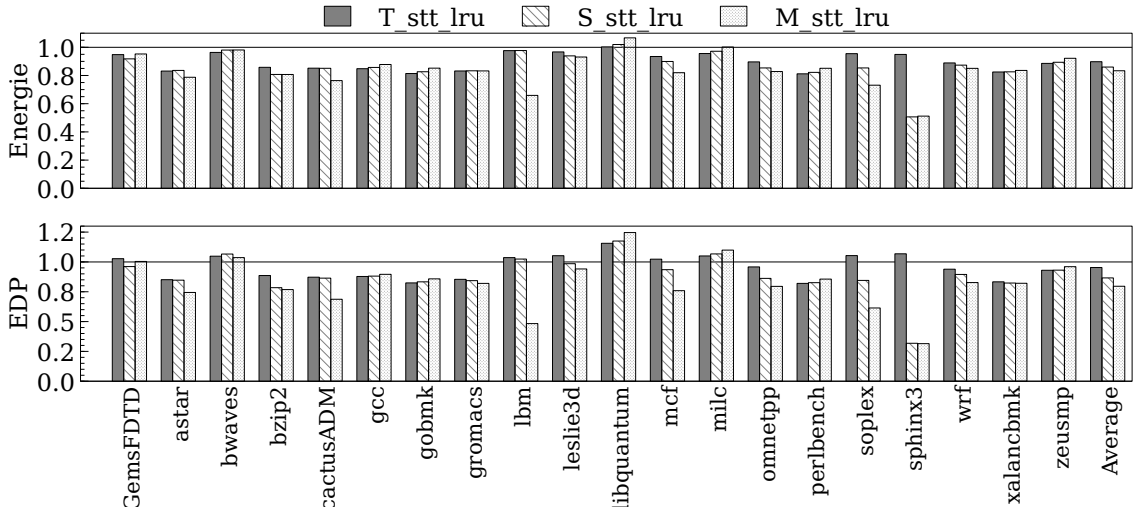


FIGURE 6.2 – Consommation énergétique (haut) et efficacité énergétique (bas)

6.2.3 Impact de la politique de remplacement

Dans cette section, nous ajoutons la politique de remplacement Hawkeye aux configurations de cache précédemment explorées. Les résultats de performance sont présentés par la Figure 6.3. On observe une amélioration avec la configuration *T_sram_hwk*, i.e., la configuration Hawkeye avec de la SRAM. Cette configuration ne dégrade jamais les performance et fournit une accélération d'en moyenne $1.05\times$. Les configurations larges de caches STT-MRAM, *S_stt_hwk* et *M_stt_hwk*, sont en moyenne plus efficaces que *T_sram_hwk*. Grâce à la politique Hawkeye, *S_stt_hwk* et *M_stt_hwk* surpassent la référence pour les applications *lbm* et *mcf*, ce qui n'était pas le cas avec la LRU.

La stratégie Hawkeye améliore les performance là où un cache plus grand ne le peut pas. Par exemple, toutes les configurations STT-MRAM avec la LRU ont le même IPC pour l'application *milc*. Lorsque la politique Hawkeye est utilisée, la performance croit linéairement en fonction de la capacité de stockage du LLC. De même, les performances de l'application *libquantum*, bien qu'inférieures à celle de la SRAM, croient avec l'augmentation de la capacité de stockage du LLC et l'utilisation de Hawkeye. Cela montre que cette stratégie est capable de traiter des modèles d'accès à la mémoire qu'un cache plus grand ne peut pas.

La meilleure configuration est *M_stt_hwk*, avec une amélioration moyenne de performance de $1.10\times$ par rapport à la référence *T_sram_lru*.

Les résultats de consommation énergétiques sont visibles sur la Figure 6.4. Pour le scénario SRAM avec Hawkeye, le diminution de consommation est minime, 4.7%. La STT-

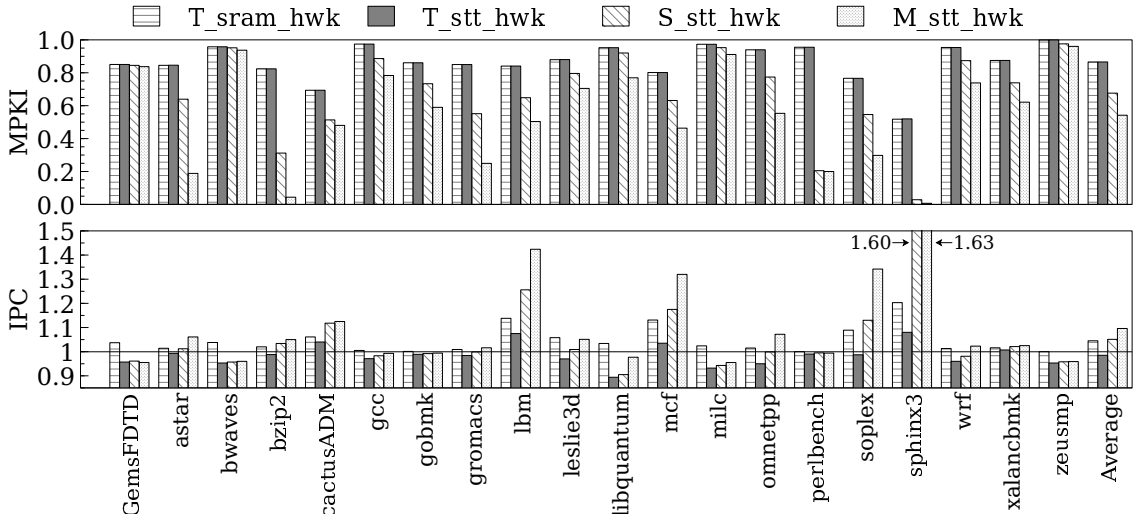


FIGURE 6.3 – MPKI (haut) et IPC (bas) avec Hawkeye normalisé à *M_sram_lru*

MRAM et sa faible puissance statique permettent de réduire l'énergie moyenne consommée de 14.3%, 18.5% et 19.7% pour les configurations *T_stt_hwk*, *S_stt_hwk* et *M_stt_hwk*. Avec la politique Hawkeye, les gains de performance sont plus marqués et aucune dégradation de la consommation énergétique n'est constatée, contrairement à la LRU.

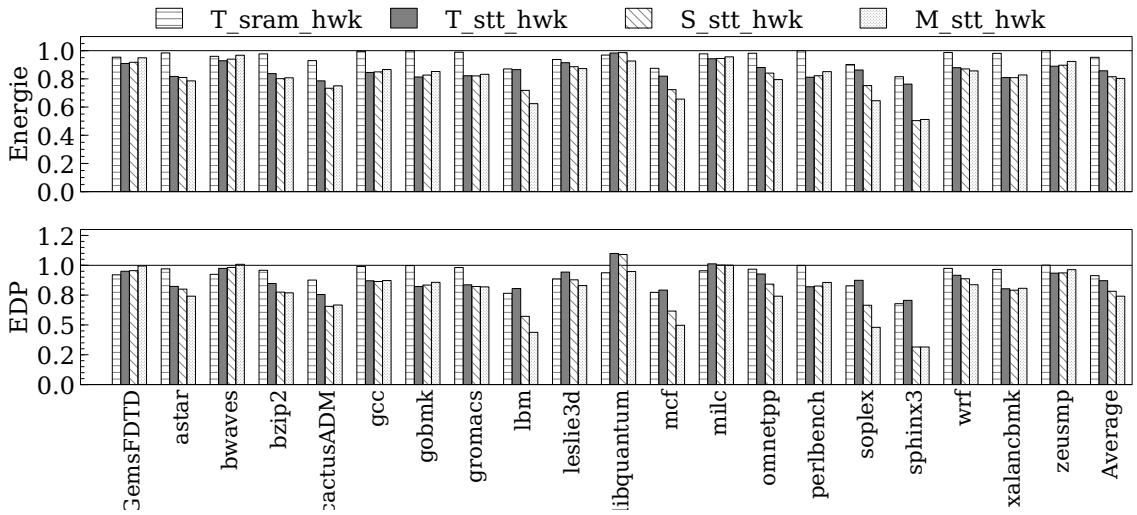


FIGURE 6.4 – Consommation énergétique (haut) et efficacité énergétique (bas)

L'efficacité énergétique est également améliorée. Seules les applications *bwaves* et *libquantum* montrent une dégradation par rapport à la SRAM, et uniquement pour certaines configurations de caches. Le meilleur EDP moyen est fourni par la configuration *M_stt_hwk* avec une réduction de 26% par rapport au scénario de référence.

6.2.4 Limites de la politique de remplacement considérée

La Figure 6.5 montre l'effet de la politique Hawkeye par rapport à la LRU. Les résultats sont normalisés pour chaque configuration à son équivalent LRU. Par exemple, *M_stt_hwk* est normalisé à *M_stt_lru*. Les configurations SRAM et STT-MRAM suivent la même tendance en ce qui concerne la réduction du MPKI. En effet, l'efficacité de Hawkeye ne dépend pas de la latence du LLC.

La Figure 6.5 montre également que la configuration $8Mo$ n'est pas aussi efficace que la configuration $4Mo$ pour l'amélioration de l'IPC. Le gain moyen pour les configurations *M_sram_hwk* et *M_stt_hwk* est inférieur à *S_sram_hwk* et *S_stt_hwk*. Cela suggère qu'il y a un problème lié à la capacité de stockage du LLC, à la politique de remplacement, ou les deux. Même si la performance moyenne reportée sur la Figure 6.3 montre que les configurations $8Mo$ sont plus rapides, nous remarquons qu'il existe peut-être une limite à l'amélioration de l'IPC par la politique Hawkeye. Ce comportement est visible avec les applications *bzip2*, *wrf* et *sphinx3*. Sur la Figure 6.1, les résultats montrent que le MPKI est réduit pour *S_stt_lru* et *M_stt_lru*. Donc, augmenter la taille du LLC est efficace. De la même manière, le MPKI est réduit pour les mêmes configurations lorsque l'on remplace LRU par Hawkeye (voir Figure 6.3). Cependant, la Figure 6.5 montre que Hawkeye augmente le MPKI comparé à la LRU pour un cache de $8Mo$. En effet, le prédicteur de la politique Hawkeye exploite tous les accès au cache pour identifier les instructions génératrices de cache *miss*. Or, l'augmentation de la capacité du LLC diminue le nombre de ces évènements. Il devient alors difficile d'entraîner le prédicteur, qui est alors plus sujet à de mauvaises prédictions. La performance de la configuration *M_stt_hwk* est cependant meilleure que toutes les autres configurations malgré ces mauvaises décisions

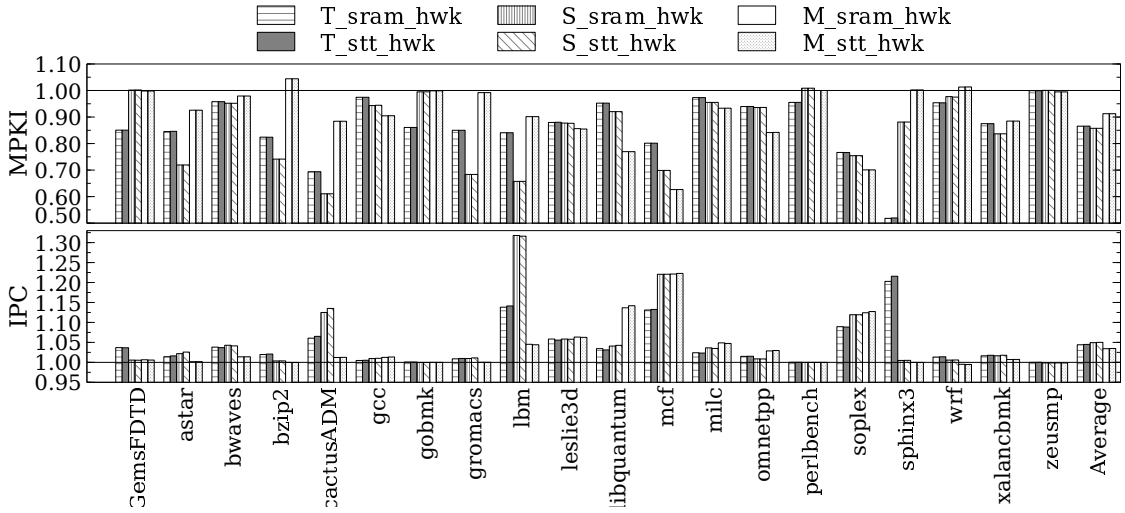


FIGURE 6.5 – Impact de Hawkeye sur les performances normalisé à la LRU

6.3 Passage à l'échelle d'un système multicœur

L'architecture multicœur évaluée se compose de 4 cœurs ayant chacun deux caches L1-I/D de $32Ko$ et un cache L2 unifié de $256Ko$. Le cache L3 est partagé par tous les cœurs. La taille initiale de ce cache en SRAM est de $4Mo$. De la même manière qu'avec l'architecture monocœur, nous utilisons la densité de la STT-MRAM pour agrandir la capacité de stockage de la mémoire cache de $4Mo$ à $16Mo$. On ajoute le préfixe B (Big) pour le cache de $16Mo$ (exemple : *B_stt_lru* pour une configuration $16Mo$ en STT-MRAM avec la LRU). Les latences des caches sont définies dans la tableau 6.1. L'environnement de simulation est identique. On évalue l'impact sur le MPKI, l'IPC, la consommation énergétique et l'EDP.

Chacun des quatre cœurs exécute une application de la suite SPEC CPU2006. On appelle *mix* l'ensemble de ces quatre applications exécutées. Le tableau 6.4 détaille les applications qui composent les 20 *mix* exécutés. Ces configurations ont été générées aléatoirement.

	Cœur 0	Cœur 1	Cœur 2	Cœur 3
mix1	gobmk	libquantum	perlbench	xalancbmk
mix2	astar	bwaves	lbm	zeusmp
mix3	cactusADM	lbm	milc	perlbench
mix4	bwaves	lbm	sphinx3	wrf
mix5	astar	cactusADM	GemsFDTD	perlbench
mix6	cactusADM	GemsFDTD	gobmk	soplex
mix7	astar	cactusADM	leslie3d	sphinx3
mix8	bwaves	libquantum	perlbench	sphinx3
mix9	cactusADM	gobmk	milc	soplex
mix10	bzip2	gobmk	lbm	perlbench
mix11	astar	gobmk	milc	soplex
mix12	gobmk	leslie3d	libquantum	perlbench
mix13	bwaves	bzip2	gobmk	wrf
mix14	gobmk	lbm	leslie3d	milc
mix15	cactusADM	gobmk	milc	perlbench
mix16	bwaves	bzip2	gobmk	leslie3d
mix17	astar	bzip2	leslie3d	xalancbmk
mix18	gobmk	libquantum	wrf	xalancbmk
mix19	gobmk	lbm	milc	zeusmp
mix20	milc	perlbench	wrf	zeusmp

Tableau 6.4 – Détail des applications exécutées sur les plateformes multicœurs

6.3.1 Impact de la capacité de la mémoire cache et de la technologie

La Figure 6.6 présente les résultats de performance pour le MPKI (haut) et l'IPC (bas). Comparé au scénario monocœur, la tendance est similaire : plus le cache est agrandi, plus le MPKI diminue. En moyenne, il est diminué de 63% pour la configuration *B_stt_lru*.

Par corollaire, les performances sont augmentées. Cela est particulièrement visible pour les configurations de cache *B_stt_lru*. Dans un contexte multicœur ou le cache de dernier niveau est partagé, les données entre les applications sont sujettes à des conflits. En augmentant la capacité de stockage, on diminue ces conflits, donc le nombre de *miss* et de *Write-Fill*.

On observe que la configuration *S_stt_lru* ralentie l'exécution des applications à cause des latences plus élevées. La configuration de cache de $8Mo$, *M_stt_lru*, fournit en moyenne la même performance que la SRAM. Enfin, la configuration *B_stt_lru* accélère l'exécution de 12%.

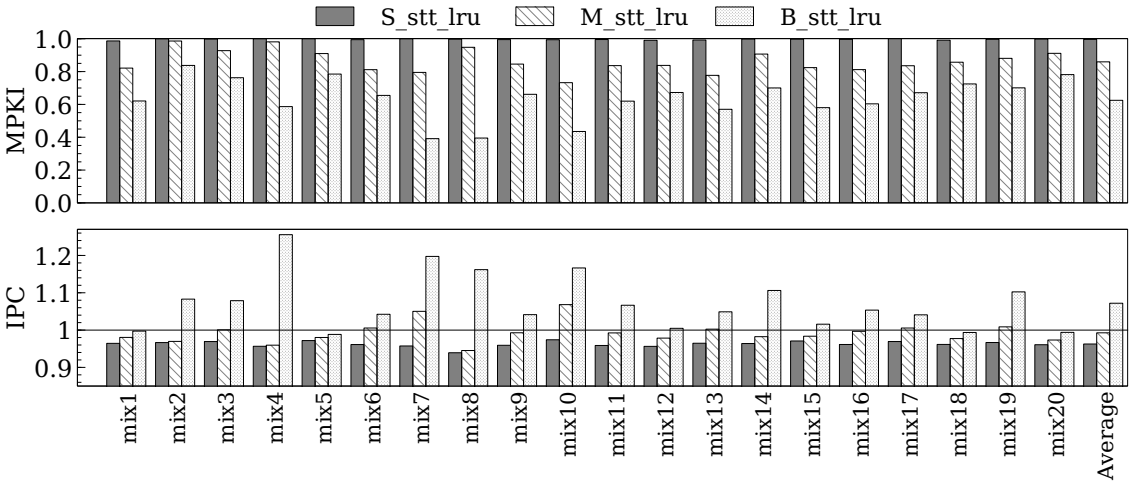


FIGURE 6.6 – Effet de l'augmentation de la taille du LLC sur le MPKI et l'IPC

Grâce à sa faible consommation statique, la STT-MRAM diminue la consommation énergétique par rapport à la SRAM pour tous les *mix* considérés. Sur la Figure 6.7, on observe cependant que dans certains cas comme les *mix* 1, 12 et 18, celle-ci augmente avec la capacité du LLC. Cela vient du fait que le temps d'exécution de l'application augmente à mesure que le cache est agrandi. Les applications souffrent des latences d'accès de la STT-MRAM qui augmentent. En moyenne, on observe une diminution de la consommation de 8%, 13.5% et 14%, respectivement pour les configurations *S_stt_lru*, *M_stt_lru* et *B_stt_lru*. En matière d'efficacité énergétique, celle-ci est dégradée par rapport à la SRAM pour 4 *mix* : 1, 12, 18 et 8. Pour les trois premiers, la diminution du temps d'exécution n'est pas assez significative pour contre-balancer l'augmentation de l'énergie dynamique de la STT-

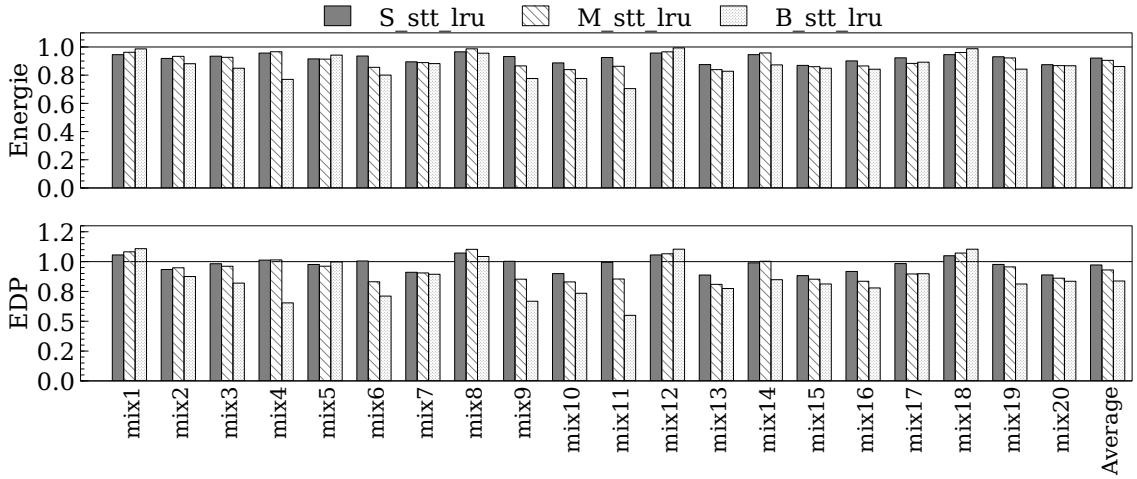


FIGURE 6.7 – Impact de la taille du LLC sur l'énergie et l'EDP

MRAM.

Le *mix* 8 montre néanmoins un comportement différent. Les performances moyennes sont augmentées d'environ 20% pour la configuration *B_stt_lru*, mais la diminution de la consommation énergétique est similaire à une configuration *S_stt_lru*. Cela est du à l'application *sphinx3*, dont l'IPC est augmentée de 50%, tandis que les trois autres applications, *bwaves*, *libquantum* et *perlbench* voient leur IPC réduit de 2.8%, 9% et 1%. La moyenne géométrique de l'IPC montre donc une augmentation des performances, alors que le temps d'exécution total de l'ensemble des applications est augmenté de 9%. Cela conduit à une amélioration très légère de l'efficacité énergétique.

Malgré ce résultat contre intuitif, la méthode d'évaluation utilisée est similaire aux précédents travaux liés aux stratégies de remplacement [34, 46, 47, 48, 114].

6.3.2 Impact de la politique de remplacement

L'ajout de la politique de remplacement Hawkeye montre une baisse significative du taux de *miss*. On observe sur la Figure 6.8 une diminution de 71% en moyenne pour la configuration *B_stt_hwk*.

En conséquence, les applications bénéficient d'un temps d'exécution plus court. Les meilleures performances sont obtenues par la configuration *B_stt_hwk* avec une amélioration moyenne de 14% sur l'ensemble des applications considérées. Sur l'ensemble des *mix*, la STT-MRAM affecte de manière négative les performances uniquement pour le *mix* 20 et la configuration de $2Mo$ avec Hawkeye (*S_stt_hwk*). Lorsque la politique Hawkeye est utilisée avec de la SRAM (*S_sram_hwk*), une seule configuration de STT-MRAM produit de moins bons résultats : la configuration $2Mo$, *S_stt_hwk*. En revanche, les configu-

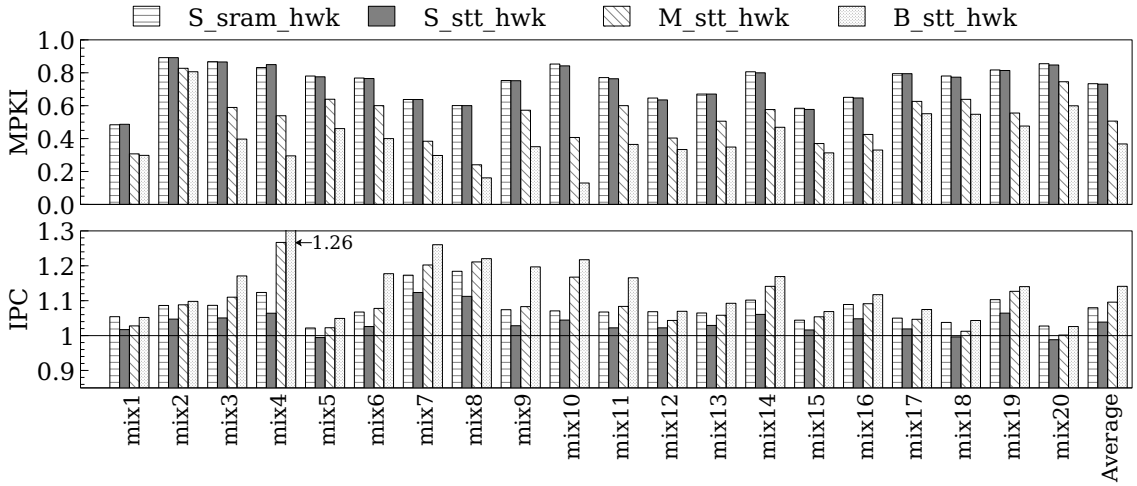


FIGURE 6.8 – Effet de l'augmentation de la taille du LLC et de Hawkeye sur le MPKI et l'IPC

rations *M_stt_hwk* et *B_stt_hwk* sont plus efficaces sur le plan de la performance que la configuration *S_sram_hwk*. Ces résultats sont possibles grâce à la combinaison des deux optimisations proposées.

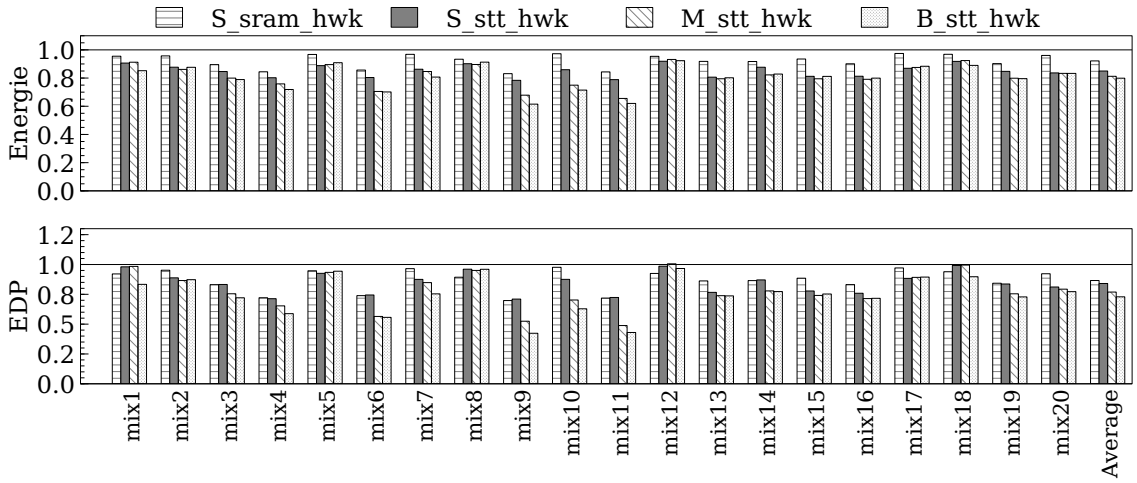


FIGURE 6.9 – Impact de la taille du LLC et de Hawkeye sur l'énergie et l'EDP

En matière d'efficacité énergétique, les résultats visibles sur la Figure 6.9 sont similaires à un système monocœur. En effet, l'amélioration est graduelle au fur et à mesure que la capacité de la mémoire cache augmente et la meilleure configuration pour l'EDP est toujours la configuration avec la plus grande capacité de stockage, ici *B_stt_hwk*. Cette configuration améliore l'EDP de 27% par rapport au scénario de référence.

6.3.3 Limites de la politique de remplacement considérée

Les remarques sur l'efficacité de la politique Hawkeye pour un système monocœur se retrouvent également pour un système multicœurs. Sur la Figure 6.10, chaque configuration utilisant la politique Hawkeye et normalisée à son équivalent avec la politique LRU.

On observe sur cette figure que le gain moyen en IPC est plus important avec des configurations de cache de $8Mo$ que $16Mo$. En moyenne, la configuration *M_stt_hwk* apporte un gain de 10%, tandis que celui de la configuration *B_stt_hwk* est de 6%. Le *mix* 4 illustre bien ce comportement. Dans ce cas, un cache de $8Mo$ avec la politique Hawkeye apportent un gain en IPC de plus de 30%. Lorsque le cache est agrandi à $16Mo$, ce gain est similaire à un cache de $4Mo$, soit 12%. Pourtant, le cache de $16Mo$ réduit plus significativement le MPKI (voir Figure 6.8).

Néanmoins, les meilleurs résultats d'IPC et d'EDP sont fournis par la configuration *B_stt_hwk*, comme observés respectivement sur les Figures 6.8 et 6.9. Ces observations montrent que l'augmentation de la capacité de stockage d'une mémoire cache est efficace dans un contexte multicœurs ou ce dernier doit contenir des données différentes pour plusieurs cœurs. En contrepartie, cela laisse moins de possibilités d'action à la politique de remplacement Hawkeye.

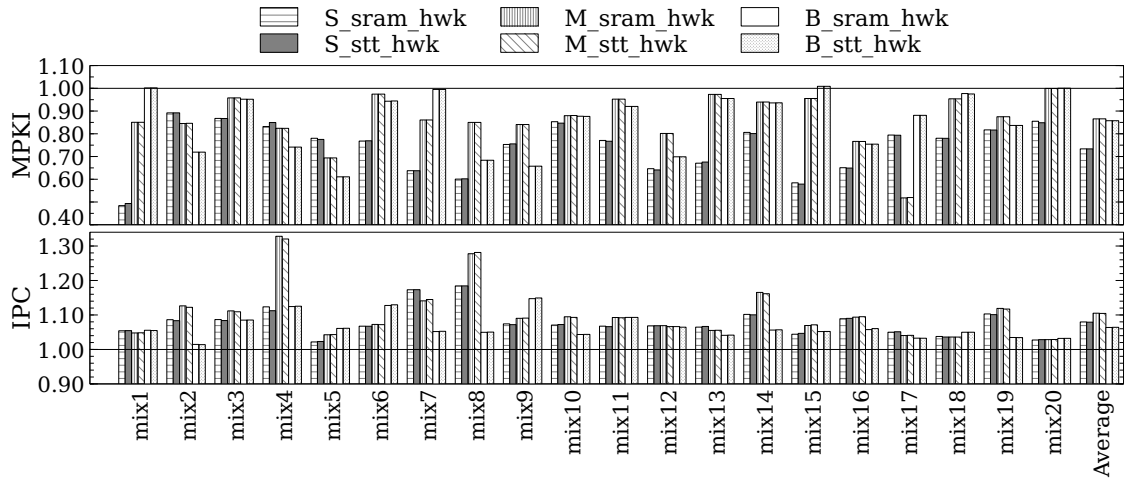


FIGURE 6.10 – Impact de Hawkeye sur les performances normalisé à la LRU

6.4 Impact des choix de conception de la mémoire cache sur les optimisations architecturales proposées pour le LLC

La section précédente a servi à montrer la validité des optimisations proposées. A présent, nous pouvons explorer différentes optimisations de conception de la mémoire cache sur lesquelles appliquer ces stratégies.

6.4.1 Choix des optimisations de conception

Notre étude porte sur deux designs de caches différents pour la STT-MRAM. On considère le design STT-MRAM utilisé dans les sections 6.2 et 6.3 comme le design C_{orig}. Un premier design, C_{lat}, optimise les latences par rapport à C_{orig}. Un second, C_{area} optimise la surface occupée. L'optimisation de la mémoire cache SRAM est un compromis entre les latences et la surface. Les caractéristiques sont données dans le tableau 6.5. La Figure 6.11 donne un aperçu de la latence de lecture en fonction de la surface pour les trois designs. Les points gris représentent chacun une configuration de cache possible, les points noirs représentent les points Pareto.

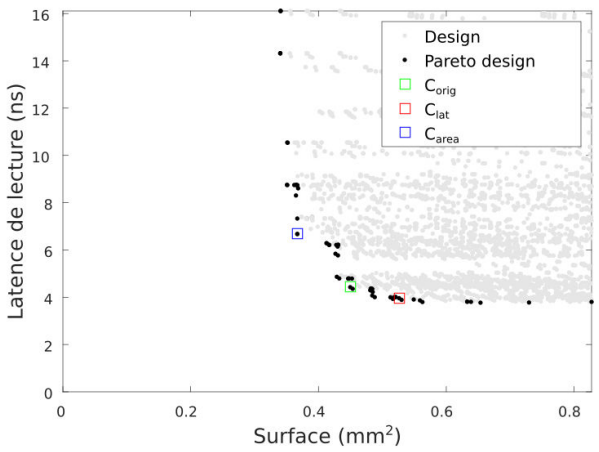


FIGURE 6.11 – Latence d'accès en lecture en fonction de la surface pour les configurations STT-MRAM.

La configuration C_{area} a été choisie pour minimiser la surface d'au moins 10% par rapport à C_{orig} tout en n'augmentant pas de manière significative les latences. A l'inverse, l'optimisation C_{lat} est sélectionnée pour ne pas trop augmenter la surface de C_{orig} (15% maximum).

Pour cette analyse, nous reprenons les mêmes nœuds de calcul qu'aux sections 6.2 et 6.3. Le LLC de référence a une capacité de $2Mo$ et $4Mo$ respectivement pour les systèmes

	SRAM		STT-MRAM C_{lat}				STT-MRAM C_{area}			
	2Mo	4Mo	2Mo	4Mo	8Mo	16Mo	2Mo	4Mo	8Mo	16Mo
Latence L. [ns]	1.43	2.10	3.96	4.01	4.71	5.26	6.67	7.33	9.16	12.89
Latence E. [ns]			5.53	5.60	5.91	6.14	7.56	8.05	10.85	13.63
Puissance [mW]	62	124	8	16	31	61	3	5	7	12
Surface [mm^2]	1.52	3.02	0.53	0.96	1.77	3.47	0.37	0.72	1.37	2.66

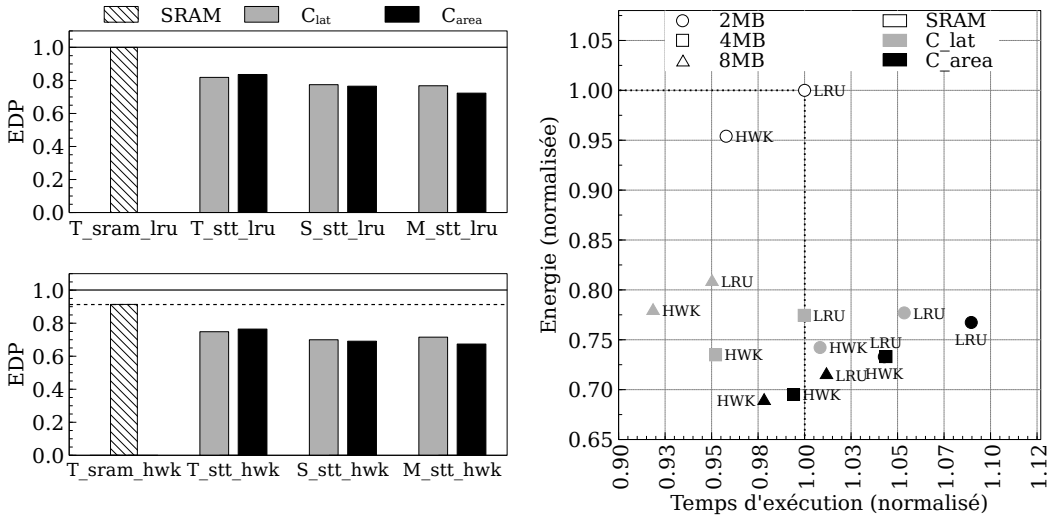
Tableau 6.5 – Caractéristiques des configurations C_{lat} et C_{area}

monocœur et multicœurs. L'environnement de simulation utilisé est basé sur ChampSim. Les applications exécutées sont extraites de la suite SPEC CPU2006.

6.4.2 Résultats expérimentaux

Plateforme monocœur

Pour un système monocœur, on observe sur la Figure 6.12a que les deux designs C_{lat} et C_{area} améliorent l'efficacité énergétique par rapport à la SRAM, et ce quelque soit la stratégie de remplacement utilisée. Les configurations *T_stt_lru* et *T_stt_hwk* du design C_{area} montrent un EDP plus élevé leur équivalent C_{lat}. Cette tendance s'inverse lorsque la taille des caches est augmentée. Ainsi, le design fournissant la meilleure efficacité énergétique est le design C_{area} et sa configuration *M_stt_hwk*.

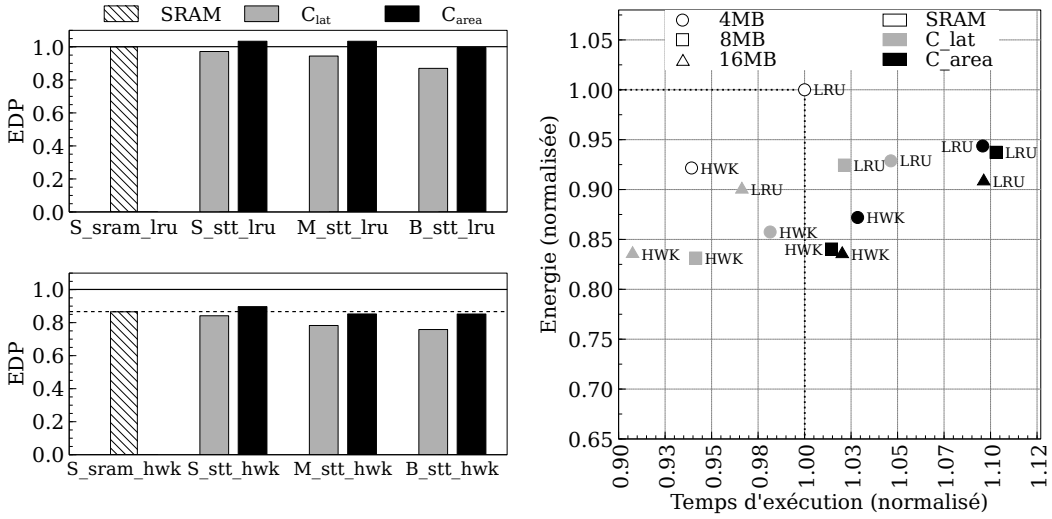


(a) Energy-Delay Product (b) Temps VS énergie

FIGURE 6.12 – Efficacité énergétique avec les politiques de remplacement LRU (haut) et Hawkeye (bas) et résultats de temps d'exécution et d'énergie pour une plateforme monocœur

On observe également deux comportements différents en fonction du design considérés. Les résultats d'EDP des configurations *S_stt_hwk* pour C_{lat} et C_{area} sont quasiment identiques mais sont obtenus de deux manières différentes. Sur la Figure 6.12b, on observe que le design C_{lat} améliore de 5% le temps d'exécution pour une diminution de la consommation énergétique de 26%. Le design C_{area} diminue seulement de 1% le temps d'exécution, mais la consommation énergétique diminue elle de plus de 30%. Ainsi, ces deux designs pour la configuration *S_stt_hwk* fournissent la même EDP en diminuant majoritairement la latence pour C_{lat} et l'énergie pour C_{area}.

La Figure 6.12 montre également que quatre configurations du design C_{lat} et deux configurations du design C_{area} égalent ou améliorent le temps d'exécution comparé à la SRAM. Deux configurations du design C_{lat} ne nécessitent pas d'utiliser la stratégie Hawkeye pour ce résultat, ce qui est le cas pour les deux configurations du design C_{area}.



(a) Energy-Delay Product

(b) Temps VS énergie

FIGURE 6.13 – Efficacité énergétique avec les politiques de remplacement LRU (haut) et Hawkeye (bas) et résultats de temps d'exécution et d'énergie pour une plateforme multicœur

Plateforme multicœur

Pour le système multicœur utilisant la LRU, on voit que le design C_{area} dégrade légèrement l'efficacité énergétique pour les configurations *S_stt_lru* et *M_stt_lru* (Figure 6.13a). Seule la configuration de cache *B_stt_lru* parvient à égaler le scénario de référence *S_sram_lru*. Le design C_{lat} quant à lui obtient de meilleurs résultats que la SRAM dans tous les cas. Lorsque l'on ajoute la stratégie Hawkeye, les résultats d'EDP du design C_{area} montrent

une légère amélioration par rapport à une configuration SRAM utilisant cette politique de remplacement. On voit ici un nouvel exemple de l'intérêt de Hawkeye et de la densité de la STT-MRAM. Le design C_{lat} montre cependant de meilleurs résultats pour l'EDP.

Contrairement à une plateforme monocœur, le design C_{area} a plus de difficultés à fournir une performance énergétique satisfaisante. En effet, ses latences plus élevées et l'augmentation du trafic en direction du LLC l'empêche de traiter toutes les requêtes sans augmenter le temps d'exécution. On observe sur la Figure 6.13b qu'aucune configuration du design C_{area} n'est capable d'améliorer le temps d'exécution moyen des applications comparé au scénario SRAM. De plus, toutes les configurations utilisant la politique LRU dégradent en moyenne le temps d'exécution de 10%. Sur un système monocœur, seule la configuration *S_stt_lru* produisait un résultat similaire.

Ces résultats montrent qu'un design comme C_{lat} privilégiant les latences en dépit de coûts énergétiques supérieurs permet dans un contexte multicœur de fournir une efficacité énergétique supérieure à la SRAM, là où un design comme C_{area} souffre de ses latences élevées et du nombre de requêtes à traiter.

Angle de lecture des résultats

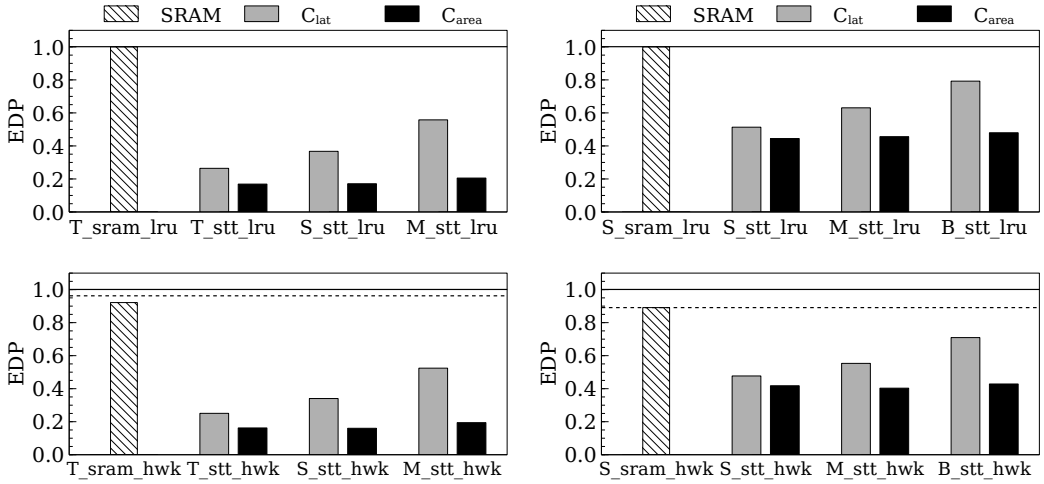
Comme nous l'avons mentionné au chapitre 3, l'analyse énergétique de plusieurs travaux de la littérature se focalise sur le niveau de cache où est intégré la STT-MRAM (ou tout autre technologie de mémoire non volatile). Nous montrons ici que lorsque l'on adopte ce point de vue, les observations peuvent être différentes qu'avec un point de vue plus global comprenant toute la hiérarchie mémoire.

La Figure 6.14 montre les résultats d'efficacité énergétique pour les plateformes monocœur et multicœurs évaluées précédemment.

Sur la Figure 6.14a (système monocœur), on voit que le design C_{area} produit une meilleure amélioration de l'EDP par rapport au design C_{lat}, et ce sur toutes les tailles de cache explorées et même avec une politique de remplacement non standard. Cette observation est identique à ce que montre la Figure 6.12a.

Pour un système multicœur en revanche (Figure 6.14b), plusieurs changements sont observés. Premièrement, aucun design ne dégrade l'efficacité énergétique comme c'est le cas sur la Figure 6.13a. Deuxièmement, le design de cache C_{lat}, initialement meilleur que le design C_{area} pour toutes les configurations, à un EDP supérieur au design C_{area}, ce qui n'était pas le cas précédemment. Enfin, la configuration *B_stt_hwk* du design C_{area} sur la Figure 6.13a égale la configuration *S_sram_hwk* en terme d'EDP. Ici, la configuration *S_sram_hwk* obtient un score d'EDP largement supérieur à la configuration *B_stt_hwk*. Le design C_{area} qui précédemment égalait difficilement la configuration *S_sram_hwk* est

maintenant meilleure de plus de 50%.



(a) Système monocœur

(b) Système multicœurs

FIGURE 6.14 – Efficacité énergétique du LLC avec les politiques de remplacement LRU (haut) et Hawkeye (bas) pour des systèmes monocœur et multicœurs

Cette illustration de la lecture des résultats montre qu'il est important de définir le niveau de perspective sur lequel se base une analyse de l'efficacité énergétique. Ici, on a vu que deux perspectives différentes montrent deux conclusions différentes dans le cas d'un système multicœur.

6.5 Résumé

Dans ce chapitre, on a cherché à valider deux optimisations architecturales pour une mémoire cache de dernier niveau utilisant la technologie STT-MRAM. Ces deux optimisations, proposées au chapitre 5, sont *i*) l'augmentation de la capacité de stockage du LLC en utilisant une organisation interne différente avec la STT-MRAM qu'avec la SRAM et *ii*) l'utilisation de Hawkeye, une politique de remplacement avancée. Les écritures de type *Write-Fill* étant les plus pénalisantes, ces deux optimisations visent à réduire le nombre de *miss* sur le LLC pour diminuer le nombre de *Write-Fill*.

Dans un contexte monocœur, la combinaison de ces optimisations permet d'améliorer l'efficacité énergétique face à un scénario « full SRAM » jusqu'à 26% en moyenne. On observe que la technologie STT-MRAM peut offrir de meilleures performances en matière d'IPC que la SRAM, jusqu'à 10% en moyenne. De plus, la politique Hawkeye seule permet d'obtenir en moyenne des performances similaires à la SRAM tout en profitant de la faible

énergie statique de la STT-MRAM, améliorant ainsi l'efficacité énergétique de 13%. On voit ici l'importance des écritures de type *Write-Fill* qu'il convient de diminuer un maximum.

Dans une configuration multicœurs où chaque cœur exécute une application différente, le cache de dernier niveau est soumis à une plus grosse quantité de trafic. En effet, il n'y a aucun partage d'informations et chaque cœur accède a des données différentes. Néanmoins, l'augmentation de la capacité du LLC permet de diminuer les conflits entre les données et d'améliorer la performance par rapport à la SRAM de 12% malgré des latences de lecture et d'écriture plus importantes. L'ajout de la stratégie Hawkeye permet une meilleure gestion de la mémoire et améliore les performances jusqu'à 14%. De la même manière qu'avec un système monocœur, l'efficacité du changement de politique de remplacement est limitée par les opportunités d'action, celles-ci étant réduites par l'augmentation de la capacité de stockage de la mémoire cache de dernier niveau. Néanmoins, l'efficacité énergétique est tout de même améliorée jusqu'à 27% par rapport à un scénario « full SRAM ». Enfin, la politique Hawkeye seule permet d'améliorer les performances de 4% par rapport à un scénario SRAM, là où la politique LRU par défaut ne le permet pas.

Ces évaluations ont permis de montrer la validité des optimisations proposées au chapitre 5, qui s'appliquent à la fois sur des plateformes monocœur et multicœurs. L'utilisation de la densité de la STT-MRAM ou une stratégie de remplacement avancée montrent chacune des gains grâce à la diminution des transactions de type *Write-Fill*. La combinaison de ces deux techniques permet de maximiser les résultats obtenus.

Nous avons ensuite analysé les différents choix de conception possibles pour un cache basée sur la technologie STT-MRAM et utilisant les optimisations proposées. Nous avons observé qu'une conception réduisant la latence peux améliorer l'efficacité énergétique sans optimisation architecturale particulière. A l'inverse, un design optimisant la surface tire pleinement profit des optimisations architecturales proposées pour fournir une efficacité énergétique similaire à la SRAM tout en libérant de l'espace sur la puce.

Enfin, nous avons illustré le choix des perspectives d'analyse de l'efficacité énergétique. Nous avons montré qu'en fonction du point de vue, il est possible de tirer des conclusions inverses. Cela implique qu'une réflexion sur la caractérisation de l'efficacité énergétique et de ce que l'on désire mesurer doit être faite lors de l'étape de conception de la mémoire cache.

In every game and con there's always an opponent, and there's always a victim. The trick is to know when you're the latter, so you can become the former.

JACK GREEN - REVOLVER (2005)

Chapitre 7

Conclusion et perspectives

7.1 Travaux présentés dans ce manuscrit

La STT-MRAM est une technologie mémoire qui est étudiée de près depuis de nombreuses années. Chercheurs comme industriels voient dans ses propriétés une éventuelle solution à l'augmentation des besoins énergétiques de nos machines. Cependant, ses latences d'accès plus élevées que la SRAM nécessitent de comprendre les enjeux de ce changement de technologie dans une hiérarchie mémoire.

L'étude de l'impact d'un changement technologique nécessite une phase d'exploration architecturale. Une première difficulté quant à cette étape est le choix du niveau d'abstraction qui doit être fait. Dans le chapitre 2, nous avons vu qu'une approche à base de simulations matérielles permet une justesse d'évaluation très fine au prix d'une mise en place qui peut s'avérer complexe et coûteuse. Si l'on se place à un niveau plus élevé, le concepteur va perdre en précision et gagner en facilité d'utilisation. Malgré cet aspect négatif, c'est ce choix que nous avons retenu pour nos travaux. Le chapitre 4 de cette thèse a

été consacré à la définition d'un schéma générique de la manière d'aborder nos explorations architecturales, ainsi qu'à la présentation des deux implémentations que nous avons réalisées.

La première implémentation, MAGPIE, est basée sur un ensemble d'outils de la littérature dont nous avons automatisé le fonctionnement. En connectant un simulateur de mémoires non volatiles, un simulateur d'architectures et un outil d'estimation de consommation énergétique, MAGPIE permet des explorations à un niveau système et architectural. Cependant, l'analyse des spécificités de la STT-MRAM nous a poussé vers l'étude de la gestion mémoire par les politiques de remplacement de caches. Cette étude a notamment été présentée au chapitre 2. Nous avons alors repris le cadre générique d'exploration, et remplacé le simulateur gem5 par ChampSim, et développé un modèle analytique de consommation énergétique.

Le chapitre 3 de cette thèse a été consacré à l'étude des propositions faites pour améliorer la STT-MRAM. On a vu qu'il existe différents niveaux d'action, du niveau technologique au niveau architectural. Nous n'avons pas considéré les niveaux technologiques et circuits dans nos propositions et avons choisi d'utiliser la technologie STT-MRAM plutôt que de l'améliorer. Ainsi, la difficulté de nos travaux a résidé dans le fait que nous n'avons pas cherché à modifier la STT-MRAM mais à en intégrer les propriétés. Il nous a fallu trouver des solutions pour intégrer une technologie qui nativement propose des performances plus faibles que la technologie SRAM. Le chapitre 5 a été consacré à l'étude des ces opportunités.

Nous nous sommes intéressé à la propriété de densité de la STT-MRAM et avons observé qu'un choix judicieux dans la composition interne de la mémoire cache permet de profiter de cette densité sans augmenter drastiquement les latences d'accès. Nous nous sommes également intéressé aux optimisations de conception des caches en fonction du contexte et du niveau de cache. Enfin, nous avons mené une étude sur la hiérarchie mémoire et les transactions d'écriture qui existent. Nous avons montré qu'un seul type d'écriture est réellement pénalisant. Le nombre de ces écritures peut alors être réduit grâce à une gestion plus fine de la mémoire.

Le chapitre 6 a mis en application les résultats du chapitre 5 en montrant l'impact du grossissement de la mémoire cache et l'ajout d'une politique de remplacement avancée dans un système monocœur et multicœur. On a vu que la combinaison de ces deux optimisations permet d'obtenir de meilleures performances que la technologie SRAM, tant en matière de temps d'exécution que de consommation énergétique. L'efficacité énergétique, caractérisée dans nos travaux par l'Energy-Delay Product (EDP) est ainsi amélio-

rée jusqu'à 34% et 27% pour des système respectivement monocœur et multicœurs. De plus, la stratégie Hawkeye seule a montré des performances similaires ou supérieures à la SRAM, illustrant l'importance des écritures de type *Write-Fill* sur la STT-MRAM. Enfin, nous avons montré que la perspective d'étude de l'efficacité énergétique doit être réfléchie et définie en amont de l'intégration d'une nouvelle technologie de mémoire.

7.2 Perspectives de travail à court terme

Cette section propose des perspectives de travail sur les approches développées pendant la thèse.

7.2.1 Fiabilité de la technologie STT-MRAM

Inversion de bits

Dans nos travaux de thèse, mous avons négligé les aspects relatifs aux inversions de bits, ou *bit-flip* [10]. Une inversion de bits est un phénomène qui a lieu lors d'une lecture et où l'information présente est inversée ($0 \rightarrow 1$ ou $1 \rightarrow 0$). En effet, l'impulsion électrique nécessaire pour une lecture dans une cellule MTJ est plus faible que pour une écriture. Cependant, ces deux transactions partagent le même chemin d'accès à la cellule. Cette différence de courant génère une perturbation magnétique dans la MTJ, ce qui peut provoquer un changement d'orientation et inverser la valeur. Cela est d'autant plus vrai lorsque la finesse de gravure diminue. Les travaux sur l'évolution du nœud technologique de la STT-MRAM montrent que l'inversion de bit augmente de manière significative [31, 73]. Afin de tout de même valider nos propositions architecturales, nous avons alors fait le choix de négliger ce problème.

L'injection de fautes dans le cache devra alors être ajoutée aux flots automatiques que nous avons développé dans le but de tester la validité de nos approches. Cela nécessite néanmoins de trouver ou de définir un modèle de faute pour la technologie STT-MRAM.

Endurance

Dans cette thèse, nous n'avons pas tenu compte d'un aspect de la STT-MRAM qui est son endurance limitée [120]. L'endurance représente la durée de vie d'une cellule MTJ. Bien que très élevée ($\sim 10^{15}$), celle-ci reste actuellement limitée dans le temps contrairement à la SRAM [1]. En effet, le matériau utilisé pour stocker l'information se dégrade au

1. Les transistors SRAM se dégradent également mais on considère par abus de langage leur durée de vie comme illimitée

fur et à mesure des changements induits par les impulsions électriques. Passé ce temps, la cellule sera inutilisable.

Cette observation est particulièrement vraie dans un contexte multicœur. Durant la phase de validation, nous avons placé la STT-MRAM dans le cache de dernier niveau partagé par tous les cœurs. Celui-ci est potentiellement soumis à un plus grand nombre d'écritures que dans le cas d'un cache L2 privé. A titre d'exemple, les 20 applications de la suite SPEC CPU2006 utilisées pour nos travaux génèrent en moyenne 16×10^6 transactions pour 8×10^8 instructions et un IPC moyen de 0.41. Si l'on considère une endurance maximale de 1×10^{15} et une machine calculant en permanence, il suffirait de 90 jours pour que les premières cellules MTJ inutilisables apparaissent.

Plusieurs approches, présentées au chapitre 3, existent pour augmenter l'endurance de la STT-MRAM. On peut utiliser des caches hybrides, déplacer des données entre différentes lignes de cache, détecter les écritures redondantes etc. La politique Hawkeye utilisée réduit également le nombre d'écritures, et donc augmente l'endurance des cellules MTJ. Une étude sur le nombre d'écritures évitées peut être menée. Toujours dans le contexte des politiques de remplacement, il est envisageable de développer une nouvelle stratégie pour distribuer les écritures sur les différents blocs d'une ligne lorsque plusieurs blocs sont candidats à un évincement.

7.2.2 Politique de remplacement prenant en compte l'endurance

Selon la stratégie de remplacement considérée, il est possible lors d'un remplacement sur une ligne de cache que plusieurs blocs satisfassent le critère d'éviction. Typiquement, la stratégie Hawkeye donnera toujours la priorité d'évincement maximale aux *miss block*. Si plusieurs *miss block* se trouvent sur une même ligne, le bloc évincé sera toujours le premier selon son placement sur la ligne en partant de la gauche. Si l'on imagine un cas pathologique où plusieurs lignes de cache ne contiennent que des *miss block*, alors le bloc 0 de chacune de ces lignes sera sans cesse écrit, dégradant de manière significative sa durée de vie par rapport aux autres.

Une solution pourrait alors être d'affecter pour chaque bloc des bits d'information permettant de déterminer si un bloc sur le point d'être remplacé est un ancien *miss block*. Si oui, on prendra en priorité un ancien *hit block*. Cela permet de répartir sur les différents blocs de la ligne de cache les écritures, quelles soient de type *Write-Back* ou *Write-Fill*.

7.3 Perspectives à long terme sur les mémoires non volatiles

Cette section est consacrée aux perspectives générales sur le futur des mémoires non volatiles dans les architectures.

7.3.1 Modèles énergétiques des technologies émergentes

Un des problèmes majeurs rencontrés par la communauté scientifique autour des technologies de mémoires émergentes est leur modélisation. La fiabilité des outils existants est parfois remise en question. Il est possible de construire des modèles précis au niveau matériel avec des outils comme HSPICE, ou au niveau RTL, mais cela nécessite un investissement non négligeable en matière de temps de travail. De plus, il peut s'avérer nécessaire d'avoir un partenaire industriel pour obtenir des informations très précises sur la technologie en question. Dès lors, les modèles construits ne peuvent être rendus publics sans briser la propriété intellectuelle de l'entreprise.

La simulation au niveau logiciel telle que proposée par NVSim est basée sur des modèles comportementaux à gros grains et ne représente pas fidèlement la réalité des NVM. Bien que les tendances soient majoritairement respectées, cet outil donne parfois des résultats en désaccord avec l'état de l'art.

Pour explorer le potentiel des NVM, les architectes ont besoin de modèles validés expérimentalement. Cela implique un effort conséquent de conception d'une mémoire non volatile à un niveau très bas (matériel puis RTL). Une mise à disposition de la communauté permettrait alors d'envisager une progression plus rapide dans l'évolution de la technologie. Une autre possibilité pour accéder à des modèles validés serait une entreprise qui accepterait de rendre public une partie de sa technologie à des fins de recherche, comme l'a fait ARM en proposant les modèles RTL de ses processeurs [2] de la gamme Cortex-M™.

7.3.2 Intégration en trois dimensions

Depuis une dizaine d'années, chercheurs et ingénieurs tentent de construire des cellules mémoires en trois dimensions en empilant les couches de silicium et en les connectant verticalement. Une telle organisation permet, comparé à un schéma 2D, d'améliorer les performances des mémoires, d'augmenter la capacité de stockage pour une surface moindre et de limiter l'augmentation de la consommation. Cette organisation montre des résultats prometteurs pour les technologies CMOS classiques comme la SRAM et la DRAM [11, 67].

Aujourd'hui, la maturité de certaines mémoires non volatiles comme la STT-MRAM ou la ReRAM indique que ces technologies seront bientôt capables de rivaliser avec la SRAM et la DRAM. Cependant, les modèles de comparaison sont des modèles à deux dimensions. D'ici quelques années, le marché des technologies CMOS classiques va se tourner vers la trois dimension et les NVM deviendront de nouveau moins performantes. Afin d'anticiper cette évolution, l'effort de recherche sur les NVM doit se poursuivre dans

2. https://www.arm.com/resources/designstart/designstart-university

cette même direction [21, 27, 60, 111].

You ought to spend a little more time trying to make something of yourself and a little less time trying to impress people.

Richard Vernon - The Breakfast Club (1985)

Communications et valorisation

Communications en lien avec la thèse

Pierre-Yves Péneau, David Novo, Florent Bruguier, Lionel Torres, Gilles Sassatelli and Abdoulaye Gamatié. Improving the Performance of STT-MRAM LLC through Enhanced Cache Replacement Policy *International Conference on Architecture of Computing Systems (ARCS).* April 2018.

Pierre-Yves Péneau, David Novo, Florent Bruguier, Gilles Sassatelli and Abdoulaye Gamatié. Performance and Energy Assessment of Last-Level Cache Replacement Policies *International conference on Embedded & Distributed System (EDIS).* December 2017.

Ron Mertens A new European project aims to develop a system-level STT-MRAM exploration flow. *https://www.mram-info.com/eu-launches-project-develop-system-level-stt-mram-exploration-flow* October 2017

Pierre-Yves Péneau and Abdoulaye Gamatié. Half-Day Tutorial on MAGPIE tool *Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS).* June 2017

Official webpage for the MAGPIE tool with installation/user guides and slides from the tutorial at COMPAS'17 http://www.lirmm.fr/continuum-project/pages/magpie.html

Sophiane Senni, Thibaud Delobelle, Odilia Coi, **Pierre-Yves Péneau**, Lionel Torres, Abdoulaye Gamatié, Pascal Benoit and Gilles Sassatelli. Embedded Systems to High Performance Computing using STT-MRAM *Design, Automation and Test in Europe (DATE).* March 2017.

Pierre-Yves Péneau, Rabab Bouziane, Abdoulaye Gamatié, Erven Rohou, Florent Bruguier, Gilles Sassatelli, Lionel Torres and Sophiane Senni. Loop Optimization in Presence

of STT-MRAM Caches : a Study of Performance-Energy Tradeoffs *International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS).* September 2016.

Thibaud Delobelle, **Pierre-Yves Péneau**, Abdoulaye Gamatié, Florent Bruguier, Sophiane Senni, Gilles Sassatelli and Lionel Torres. MAGPIE : System-level Evaluation of Manycore Systems with Emerging Memory Technologies *International Workshop on Emerging Memory Solutions (EMS).* March 2017

Thibaud Delobelle, **Pierre-Yves Péneau**, Sophiane Senni, Florent Bruguier, Abdoulaye Gamatié, Gilles Sassatelli and Lionel Torres. Flot automatique d'évaluation pour l'exploration d'architectures à base de mémoires non volatiles *Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS).* July 2016

Communications non liées à la thèse

Mohamed Lamine Karaui, **Pierre-Yves Péneau**, Quentin Meunier, Franck Wasjbürt and Alain Greiner. Exploiting Large Memory Using 32-bit Energy-Efficient Manycore Architectures *IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip* September 2016

Pierre-Yves Péneau, Mohamed Lamine Karaoui and Franck Wajsbürt. Migration de Processus pour un Multi-Noyau Large Echelle *Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS).*, July 2016

Bibliographie

[1] Anant Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz. An Evaluation of Directory Schemes for Cache Coherence. In *ACM SIGARCH Computer Architecture News*, volume 16, pages 280–298. IEEE Computer Society Press, 1988.

[2] Junwhan Ahn and Kiyoung Choi. Lower-Bits Cache for Low Power STT-RAM Caches. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 480–483. IEEE, 2012.

[3] Hiroyuki Akinaga and Hisashi Shima. Resistive Random Access Memory (ReRAM) Based on Metal Oxides. *Proceedings of the IEEE*, 98(12) :2237–2251, 2010.

[4] Esteve Amat, E Amatllé, Sergio Gómez, Nivard Aymerich, Carmen G Almudéver, Francesc Moll, and Antonio Rubio. Systematic and Random Variability Analysis of Two Different 6T-SRAM Layout Topologies. *Microelectronics Journal*, 44(9) :787–793, 2013.

[5] Laszlo A. Belady. A Study of Replacement Algorithms for a Virtual-Storage Computer. *IBM Systems journal*, 5(2) :78–101, 1966.

[6] Laszlo A. Belady and Frank P. Palermo. On-line Measurement of Paging Behavior by the Multivalued MIN Algorithm. *IBM Journal of Research and Development*, 18(1) : 2–19, 1974.

[7] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, volume 41, page 46, 2005.

[8] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC Benchmark Suite : Characterization and Architectural Implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[9] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2) :1–7, 2011.

[10] Rajendra Bishnoi, Mojtaba Ebrahimi, Fabian Oboril, and Mehdi B Tahoori. Read Disturb Fault Detection in STT-MRAM. In *Test Conference (ITC), 2014 IEEE International*, pages 1–7. IEEE, 2014.

[11] Bryan Black, Murali Annavaram, Ned Brekelbaum, John DeVale, Lei Jiang, Gabriel H Loh, Don McCaule, Pat Morrow, Donald W Nelson, Daniel Pantuso, et al. Die Stacking (3D) Microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–479. IEEE Computer Society, 2006.

[12] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. Emerging NVM : A Survey on Architectural Integration and Research Challenges. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(2) :14, 2017.

[13] Katherine Bourzac. Has Intel Created a Universal Memory Technology ? [News]. *IEEE Spectrum*, 54(5) :9–10, 2017.

[14] Rabab Bouziane, Erven Rohou, and Abdoulaye Gamatié. Compile-Time Silent-Store Elimination for Energy Efficiency : an Analytic Evaluation for Non-Volatile Cache Memory. In *Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation : Methods and Tools*, page 5. ACM, 2018.

[15] Rabab Bouziane, Erven Rohou, and Abdoulaye Gamatié. Partial Worst-Case Execution Time Analysis. In *ComPAS : Conférence d'informatique en Parallélisme, Architecture et Système*, Jul 2018.

[16] Doug Burger and Todd M Austin. The SimpleScalar Tool Set, Version 2.0. *ACM SIGARCH computer architecture news*, 25(3) :13–25, 1997.

[17] Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. Accuracy Evaluation of gem5 Simulator System. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–7. IEEE, 2012.

[18] Anastasiia Butko, Florent Bruguier, Abdoulaye Gamatié, Gilles Sassatelli, David Novo, Lionel Torres, and Michel Robert. Full-System Simulation of big.LITTLE Multicore Architecture for Performance and Energy Exploration. In *Embedded*

Multicore/Many-core Systems-on-Chip (MCSoC), 2016 IEEE 10th International Symposium on, pages 201–208. IEEE, 2016.

[19] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. Sniper : Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 52. ACM, 2011.

[20] Aaron Carroll, Gernot Heiser, et al. An Analysis of Power Consumption in a Smartphone. In *USENIX annual technical conference*, volume 14, pages 21–21. Boston, MA, 2010.

[21] Meng-Fan Chang, Pi-Feng Chiu, Wei-Cheng Wu, Ching-Hao Chuang, and Shyh-Shyuan Sheu. Challenges and Trends in Low-Power 3D Die-Stacked IC Designs Using RAM, Memristor Logic, and Resistive Memory (ReRAM). In *ASIC (ASICON), 2011 IEEE 9th International Conference on*, pages 299–302. IEEE, 2011.

[22] Hui Chen, Shinan Wang, and Weisong Shi. Where Does the Power Go in a Computer System : Experimental Analysis and Implications. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–6. IEEE, 2011.

[23] Odilia Coi, Guillaume Patrigeon, Sophiane Senni, Lionel Torres, and Pascal Benoit. A Novel SRAM - STT-MRAM Hybrid Cache Implementation Improving Cache Performance. In *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 39–44. IEEE, 2017.

[24] Thibaud Delobelle, Pierre-Yves Péneau, Sophiane Senni, Florent Bruguier, Abdoulaye Gamatié, Gilles Sassatelli, and Lionel Torres. Flot automatique d'évaluation pour l'exploration d'architectures à base de mémoires non volatiles. In *ComPAS : Conférence en Parallélisme, Architecture et Système*, 2016.

[25] Thibaud Delobelle, Pierre-Yves Péneau, Abdoulaye Gamatié, Florent Bruguier, Sophiane Senni, Gilles Sassatelli, and Lionel Torres. MAGPIE : System-level evaluation of manycore systems with emerging memory technologies. In *EMS : Emerging Memory Solutions*, 2017.

[26] Jean-Philippe Diguet, Naoya Onizawa, Mostafa Rizk, Johanna Sepulveda, Amer Baghdadi, and Takahiro Hanyu. Networked Power-Gated MRAMs for Memory-Based Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, (99) :1–13, 2018.

[27] Xiangyu Dong, Naveen Muralimanohar, Norm Jouppi, Richard Kaufmann, and Yuan Xie. Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems. In *Proceedings of the conference on high performance computing networking, storage and analysis*, page 57. ACM, 2009.

[28] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. Nvsim : A circuit-level performance, energy, and area model for emerging nonvolatile memory. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(7) :994–1007, 2012.

[29] Xiangyu Dong, Cong Xu, Norm Jouppi, and Yuan Xie. NVSim : A circuit-level performance, energy, and area model for emerging non-volatile memory. In *Emerging Memory Technologies*, pages 15–50. Springer, 2014.

[30] Zhuohui Duan, Haikun Liu, Xiaofei Liao, and Hai Jin. HME : A Lightweight Emulator for Hybrid Memory. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*, pages 1375–1380. IEEE, 2018.

[31] Xuanyao Fong, Yusung Kim, Sri Harsha Choday, and Kaushik Roy. Failure Mitigation Techniques for 1T-1MTJ Spin-Transfer Torque MRAM Bit-Cells. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(2) :384–395, 2014.

[32] Davy Genbrugge, Stijn Eyerman, and Lieven Eeckhout. Interval Simulation : Raising the Level of Abstraction in Architectural Simulation. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12. IEEE, 2010.

[33] Manzur Gill, Tyler Lowrey, and John Park. Ovonic unified memory-a high-performance nonvolatile memory technology for stand-alone memory and embedded applications. In *Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International*, volume 1, pages 202–459. IEEE, 2002.

[34] Paul Gratz, Jinchun Kim, and Gino Chacon. ISCA 2017 Cache Replacement Championship, 2017. http://crc2.ece.tamu.edu.

[35] PLDA Group. Methods to Fine-Tune Power Consumption of PCIe Devices, 201X.

[36] Anthony Gutierrez, Joseph Pusdesris, Ronald G Dreslinski, Trevor Mudge, Chander Sudanthi, Christopher D Emmons, Mitchell Hayenga, and Nigel Paver. Sources of Error in Full-System Simulation. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 13–22. IEEE, 2014.

[37] Mark Halper. Supercomputing's Super Energy Needs, and What to Do About Them, September 2015. Communications of the ACM.

[38] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. Simpoint 3.0 : Faster and More Flexible Program Phase Analysis. *Journal of Instruction Level Parallelism*, 7 (4) :1–28, 2005.

[39] Hardkernel. Hardkernel co., ltd. http://www.hardkernel.com/.

[40] Jingtong Hu, Chun Jason Xue, Qingfeng Zhuge, Wei-Che Tseng, and Edwin H-M Sha. Data Allocation Optimization for Hybrid Scratch-Pad Memory with SRAM and Non-Volatile Memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(6) :1094–1102, 2013.

[41] Zhigang Hu, Alper Buyuktosunoglu, Viji Srinivasan, Victor Zyuban, Hans Jacobson, and Pradip Bose. Microarchitectural Techniques for Power Gating of Execution Units. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 32–37. ACM, 2004.

[42] K Ikegami, H Noguchi, C Kamata, M Amano, K Abe, K Kushida, E Kitagawa, T Ochiai, N Shimomura, A Kawasumi, et al. A 4ns, 0.9V Write Voltage Embedded Perpendicular STT-MRAM Fabricated by MTJ-Last Process. In *VLSI Technology, Systems and Application (VLSI-TSA), Proceedings of Technical Program-2014 International Symposium on*, pages 1–2. IEEE, 2014.

[43] Intel. Intel and micron produce breakthrough memory technology, 2015. https://newsroom.intel.com/news-releases/intel-and-micron-produce-breakthrough-memory-technology/.

[44] ITRS. International Technology Roadmap for Semiconductors, 2018. http://www.itrs2.net.

[45] Amin Jadidi, Mohammad Arjomand, and Hamid Sarbazi-Azad. High-Endurance and Performance-Efficient Design of Hybrid Cache Architectures Through Adaptive Line Replacement. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pages 79–84. IEEE Press, 2011.

[46] Akanksha Jain and Calvin Lin. Back to the Future : Leveraging Belady's Algorithm for Improved Cache Replacement. In *Int'l Symp. on Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual*, pages 78–89. IEEE, 2016.

[47] Aamer Jaleel, Kevin B Theobald, Simon C Steely Jr, and Joel Emer. High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP). In *ACM SIGARCH Comp. Arch. News*, volume 38, pages 60–71, 2010.

[48] Aamer Jaleel, Hashem H Najaf-Abadi, Samantika Subramaniam, Simon C Steely, and Joel Emer. CRUISE : Cache Replacement and Utility-Aware Scheduling. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 249–260. ACM, 2012.

[49] Song Jiang, Feng Chen, and Xiaodong Zhang. CLOCK-Pro : An Effective Improvement of the CLOCK Replacement. In *USENIX Annual Technical Conference, General Track*, pages 323–336, 2005.

[50] Adwait Jog, Asit K Mishra, Cong Xu, Yuan Xie, Vijaykrishnan Narayanan, Ravishankar Iyer, and Chita R Das. Cache Revive : Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs. In *Proceedings of the 49th Annual Design Automation Conference*, pages 243–252. ACM, 2012.

[51] Chankyung Kim, Keewon Kwon, Chulwoo Park, Sungjin Jang, and Joosun Choi. A Covalent-Bonded Cross-Coupled Current-Mode Sense Amplifier for STT-MRAM with 1T1MTJ Common Source-Line Structure Array. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[52] Jinchun Kim. The ChampSim simulator, 2017. https://github.com/ChampSim.

[53] Nam Sung Kim, Todd Austin, David Baauw, Trevor Mudge, Krisztián Flautner, Jie S Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage Current : Moore's Law Meets Static Power. *computer*, 36(12) :68–75, 2003.

[54] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator : A Fast and Extensible DRAM Simulator. *IEEE Computer architecture letters*, 15(1) :45–49, 2016.

[55] Manu Komalan, José Ignacio Gómez Pérez, Christian Tenllado, Praveen Raghavan, Matthias Hartmann, and Francky Catthoor. Feasibility Exploration of NVM based I-cache through MSHR Enhancements. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[56] Ananda Vardhan Kommaraju. *Designing Energy-Aware Optimization Techniques through Program Behavior Analysis*. PhD thesis, Indian Institute of Science BANGALORE, 2014.

[57] An-Chow Lai, Cem Fide, and Babak Falsafi. Dead-Block Prediction & Dead-Block Correlating Prefetchers. In *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on*, pages 144–154. IEEE, 2001.

[58] Stefan K Lai. Flash Memories : Successes and Challenges. *IBM Journal of Research and Development*, 52(4.5) :529–535, 2008.

[59] KM Lepak and MH Lipasti. On the Value Locality of Store Instructions. In *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, pages 182–191. IEEE, 2000.

[60] Dean L Lewis and Hsien-Hsin S Lee. Architectural Evaluation of 3D Stacked RRAM Caches. In *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, pages 1–4. IEEE, 2009.

[61] Jianhua Li, Liang Shi, Qingan Li, Chun Jason Xue, Yiran Chen, Yinlong Xu, and Wei Wang. Low-Energy Volatile STT-RAM Cache Design Using Cache-Coherence-Enabled Adaptive Refresh. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 19(1) :5, 2013.

[62] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. McPAT : an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480. IEEE, 2009.

[63] Yong Li, Yiran Chen, and Alex K Jones. A Software Approach for Combating Asymmetries of Non-Volatile Memories. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 191–196. ACM, 2012.

[64] Yong Li, Yaojun Zhang, Hai Li, Yiran Chen, and Alex K Jones. C1C : a Configurable, Compiler-Guided STT-RAM L1 Cache. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4) :52, 2013.

[65] Yiling Lin and Jessie Shen. Samsung Ready to Mass Produce MRAM Chips using 28nm FD-SOI Process, September 2017. Digitimes journal, https://digitimes.com/news/a20170925PD206.html.

[66] Mirko Loghi, Massimo Poncino, and Luca Benini. Cache Coherence Tradeoffs in Shared-Memory MPSoCs. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(2) :383–407, 2006.

[67] Gabriel H Loh, Yuan Xie, and Bryan Black. Processor Design in 3D Die-Stacking Technologies. *IEEE Micro*, 27(3), 2007.

[68] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin : Building Custo-

mized Program Analysis Tools With Dynamic Instrumentation. In *ACM SIGPLAN Notices*, volume 40, pages 190–200. ACM, 2005.

[69] Deepak M Mathew, Éder F Zulian, Subash Kannoth, Matthias Jung, Christian Weis, and Norbert Wehn. A Bank-Wise DRAM Power Model for System Simulations. In *Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation : Methods and Tools*, page 5. ACM, 2017.

[70] Jagan Singh Meena, Simon Min Sze, Umesh Chand, and Tseung-Yuen Tseng. Overview of Emerging Nonvolatile Memory Technologies. *Nanoscale Research Letters*, 9 (1) :526, Sep 2014. doi : 10.1186/1556-276X-9-526.

[71] Sparsh Mittal and Jeffrey S Vetter. A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(5) :1537–1550, 2016.

[72] Sparsh Mittal, Jeffrey S Vetter, and Dong Li. A Survey of Architectural Approaches for Managing Embedded DRAM and Non-Volatile On-Chip Caches. *IEEE Transactions on Parallel and Distributed Systems*, 26(6) :1524–1537, 2015.

[73] Helia Naeimi, Charles Augustine, Arijit Raychowdhury, Shih-Lien Lu, and James Tschanz. STTRAM Scaling and Retention Failure. *Intel Technology Journal*, 17(1), 2013.

[74] Nicholas Nethercote and Julian Seward. Valgrind : a Framework for Heavyweight Dynamic Binary Instrumentation. In *ACM Sigplan notices*, volume 42, pages 89–100. ACM, 2007.

[75] Alejandro Nocua, Florent Bruguier, Gilles Sassatelli, and Abdoulaye Gamatié. ElasticSimMATE : A Fast and Accurate gem5 Trace-Driven Simulator for Multicore Systems. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2017 12th International Symposium on*, pages 1–8. IEEE, 2017.

[76] Hiroki Noguchi, Kazutaka Ikegami, Keiichi Kushida, Keiko Abe, Shogo Itai, Satoshi Takaya, Naoharu Shimomura, Junichi Ito, Atsushi Kawasumi, Hiroyuki Hara, et al. A 3.3ns-Access-Time 71.2μW/Mhz 1Mb Embedded STT-MRAM Using Physically Eliminated Read-Disturb Scheme and Normally-Off Memory Architecture. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[77] Fabian Oboril, Rajendra Bishnoi, Mojtaba Ebrahimi, and Mehdi B Tahoori. Evaluation of hybrid memory technologies using SOT-MRAM for on-chip cache hierarchy.

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 34(3) : 367–380, 2015.

[78] Takashi Ohsawa, Hiroki Koike, Sadahiko Miura, Hiroaki Honjo, Keizo Kinoshita, Shoji Ikeda, Takahiro Hanyu, Hideo Ohno, and Tetsuo Endoh. A 1 MB Nonvolatile Embedded Memory Using 4T2MTJ Cell With 32 b Fine-Grained Power Gating Scheme. *IEEE Journal of Solid-State Circuits*, 48(6) :1511–1520, 2013.

[79] Jyunichi Oshita. STT-MRAM Cuts Cache Memory Power Consumption by 60%, Jun 2014. Nikkei Business Publications, https://tech.nikkeibp.co.jp/dm/english/NEWS_EN/20140611/357762.

[80] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. Marss : a full system simulator for multicore x86 cpus. In *Proceedings of the 48th Design Automation Conference*, pages 1050–1055. ACM, 2011.

[81] Anuj Pathania, Qing Jiao, Aravind Prakash, and Tulika Mitra. Integrated cpu-gpu power management for 3d mobile games. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6. IEEE, 2014.

[82] Pierre-Yves Péneau, Rabab Bouziane, Abdoulayse Gamatié, Erven Rohou, Florent Bruguier, Gilles Sassatelli, Lionel Torres, and Sophiane Senni. Loop Optimization in Presence of STT-MRAM caches : a Study of Performance-Energy Tradeoffs. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2016 26th International Workshop on*, pages 162–169. IEEE, 2016.

[83] Pierre-Yves Péneau, David Novo, Florent Bruguier, Gilles Sassatelli, and Abdoulaye Gamatié. Performance and Energy Assessment of Last-Level Cache Replacement Policies. In *Embedded & Distributed Systems (EDiS), 2017 First International Conference on*, pages 1–6. IEEE, 2017.

[84] Pierre-Yves Péneau, David Novo, Florent Bruguier, Lionel Torres, Gilles Sassatelli, and Abdoulaye Gamatié. Improving the Performance of STT-MRAM LLC Through Enhanced Cache Replacement Policy. In *International Conference on Architecture of Computing Systems*, pages 168–180. Springer, 2018.

[85] Fernando Magno Quintão Pereira, Guilherme Vieira Leobas, and Abdoulaye Gamatié. Static prediction of silent stores. *ACM Transactions on Architecture and Code Optimization (TACO)*, 15(4) :44, 2018.

[86] Matt Poremba and Yuan Xie. Nvmain : An Architectural-Level Main Memory Simulator for Emerging Non-Volatile Memories. In *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, pages 392–397. IEEE, 2012.

[87] Matt Poremba, Sparsh Mittal, Dong Li, Jeffrey S Vetter, and Yuan Xie. DESTINY : A Tool for Modeling Emerging 3D NVM and eDRAM Caches. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1543–1546. EDA Consortium, 2015.

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

[89] Moinuddin K Qureshi, Daniel N Lynch, Onur Mutlu, and Yale N Patt. A Case for MLP-Aware Cache Replacement. *ACM SIGARCH Computer Architecture News*, 34 (2) :167–178, 2006.

[90] Moinuddin K Qureshi, Aamer Jaleel, Yale N Patt, Simon C Steely, and Joel Emer. Adaptive Insertion Policies for High Performance Caching. In *ACM SIGARCH Computer Arch. News*, volume 35, pages 381–391, 2007.

[91] Mitchelle Rasquinha, Dhruv Choudhary, Subho Chatterjee, Saibal Mukhopadhyay, and Sudhakar Yalamanchili. An Energy Efficient Cache Design Using Spin Torque Transfer (STT) RAM. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, pages 389–394. ACM, 2010.

[92] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. DRAMSim2 : A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters*, 10(1) :16–19, 2011.

[93] James F Scott and Carlos A Paz De Araujo. Ferroelectric memories. *Science*, 246 (4936) :1400–1405, 1989.

[94] SEMICO. Semico research corporation. http://www.semico.com.

[95] Sophiane Senni, Lionel Torres, Gilles Sassatelli, Abdoulaye Gamatie, and Bruno Mussard. Emerging Non-Volatile Memory Technologies Exploration Flow for Processor Architecture. In *VLSI (ISVLSI), 2015 IEEE Computer Society Annual Symposium on*, pages 460–460. IEEE, 2015.

[96] Sophiane Senni, Lionel Torres, Gilles Sassatelli, Abdoulaye Gamatie, and Bruno Mussard. Exploring MRAM Technologies for Energy Efficient Systems-On-Chip. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(3) :279–292, 2016.

[97] Sophiane Senni, Thibaud Delobelle, Odilia Coi, Pierre-Yves Péneau, Lionel Torres, Abdoulaye Gamatié, Pascal Benoit, and Gilles Sassatelli. Embedded Systems to High Performance Computing Using STT-MRAM. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 536–541. European Design and Automation Association, 2017.

[98] Seokbo Shim, Sungho Kim, Jooyoung Bae, Keunsik Ko, Eunryeong Lee, Kwidong Kim, Kyeongtae Kim, Sangho Lee, Jinhoon Hyun, Insung Koh, et al. A 16Gb 1.2V 3.2Gb/s/pin DDR4 SDRAM with Improved Power Distribution and Repair Strategy. In *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, pages 212–214. IEEE, 2018.

[99] Clinton W Smullen, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R Stan. Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 50–61. IEEE, 2011.

[100] Guangyu Sun, Xiangyu Dong, Yuan Xie, Jian Li, and Yiran Chen. A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 239–249. IEEE, 2009.

[101] Zhenyu Sun, Xiuyuan Bi, Hai Li, Weng-Fai Wong, Zhong-Liang Ong, Xiaochun Zhu, and Wenqing Wu. Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme. In *Microarchitecture (MICRO), 2011 44th Annual IEEE/ACM International Symposium on*, pages 329–338. IEEE, 2011.

[102] Riichiro Takemura, Takayuki Kawahara, Katsuya Miura, Hiroyuki Yamamoto, Jun Hayakawa, Nozomu Matsuzaki, Kazuo Ono, Michihiko Yamanouchi, Kenchi Ito, Hiromasa Takahashi, et al. A 32-MB SPRAM with 2T1R Memory Cell, Localized Bi-Directional Write Driver and 1'/0' Dual-Array Equalized Reference Scheme. *IEEE Journal of Solid-State Circuits*, 45(4) :869–879, 2010.

[103] Micron Technology. DDR3 MT41K512M8DA-125 datasheet, October 2017. https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/4gb_ddr3l.pdf.

[104] Saied Tehrani, Jon M Slaughter, Mark Deherrera, Brad N Engel, Nicholas D Rizzo, Jonn Salter, Mark Durlam, Renu W Dave, Jason Janesky, Brian Butcher, et al. Magnetoresistive Random Access Memory Using Magnetic Tunnel Junctions. *Proceedings of the IEEE*, 91(5) :703–714, 2003.

[105] Scott Thornton. DRAM vs SRAM, June 2017. Microcontroller Tips, https://www.microcontrollertips.com/dram-vs-sram.

[106] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Compilation techniques for low energy : An overview. In *IEEE Symposium on Low Power Electronics*, pages 38–39. IEEE, 1994.

[107] Top500.org. Tianhe-2A supercomputer, 2014. https://top500.org/system/177999.

[108] Lluís Vilanova. gem5 to mcpat converter, feb 2015. https ://projects.gso.ac.upc.edu/projects/gem5-mcpat.

[109] Matthew J Walker, Stephan Diestelhorst, Andreas Hansson, Anup K Das, Sheng Yang, Bashir M Al-Hashimi, and Geoff V Merrett. Accurate and stable run-time power modeling for mobile and embedded cpus. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(1) :106–119, 2017.

[110] Jianxing Wang, Yenni Tim, Weng-Fai Wong, Zhong-Liang Ong, Zhenyu Sun, and Hai Li. A Coherent Hybrid SRAM and STT-RAM L1 Cache Architecture for Shared Memory Multicores. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 610–615. IEEE, 2014.

[111] KL Wang, JG Alzate, and P Khalili Amiri. Low-Power Non-Volatile Spintronic Memory : STT-RAM and Beyond. *Journal of Physics D : Applied Physics*, 46(7) :074003, 2013.

[112] Steven JE Wilton and Norman P Jouppi. Cacti : An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5) :677–688, 1996.

[113] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs : Characterization and Methodological Considerations. In *ACM SIGARCH computer architecture news*, volume 23, pages 24–36. ACM, 1995.

[114] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C Steely, and Joel Emer. SHiP : Signature-based Hit Predictor for High Performance Caching. In *Int'l Symp. on Microarch. (MICRO)*, pages 430–441, 2011.

[115] Qing Wu, Massound Pedram, and Xunwei Wu. Clock-Gating and Its Application to Low Power Design of Sequential Circuits. *IEEE Transactions on Circuits and Systems I : Fundamental Theory and Applications*, 47(3) :415–420, 2000.

[116] Sam Likun Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. Quantifying Sources of Error in McPAT and Potential Impacts on Architectural Studies. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 577–589. IEEE, 2015.

[117] Wei Xu, Hongbin Sun, Xiaobin Wang, Yiran Chen, and Tong Zhang. Design of Last-Level On-Chip Cache Using Spin-Torque Transfer RAM (STT RAM). *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(3) :483–493, 2011.

[118] Sadegh Yazdanshenas, Marzieh Ranjbar Pirbasti, Mahdi Fazeli, and Ahmad Patooghy. Coding Last Level STT-RAM Cache for High Endurance and Low Power. *IEEE computer architecture letters*, 13(2) :73–76, 2014.

[119] Matt T Yourst. Ptlsim : A cycle accurate full system x86-64 microarchitectural simulator. In *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 23–34. IEEE, 2007.

[120] WS Zhao, Yue Zhang, Thibaut Devolder, Jacques-Olivier Klein, Dafiné Ravelosona, Claude Chappert, and Pascale Mazoyer. Failure and Reliability Analysis of STT-MRAM. *Microelectronics Reliability*, 52(9-10) :1848–1852, 2012.

[121] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. Energy Reduction for STT-RAM USsing Early Write Termination. In *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 264–268. IEEE, 2009.