Exploration of reconfigurable tiles of computing-in-memory architecture for data-intensive applications

Roman Gauchi

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE GRENOBLE ALPES

Spécialité : NANO ELECTRONIQUE ET NANO TECHNOLOGIES

Arrêté ministériel : 25 mai 2016

Présentée par

Roman GAUCHI

Thèse dirigée par **Henri-Pierre CHARLES**, CEA, UGA
et codirigée par **Pascal VIVET**, CEA, UGA
et **Subhasish MITRA**, Stanford University

préparée au sein du **Laboratoire CEA LIST**
dans **l'École Doctorale Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)**

Exploration d'une architecture tuilée reconfigurable de mémoire calculante pour les applications gourmandes en données

Exploration of reconfigurable tiles of computing-in-memory architecture for data-intensive applications

Thèse soutenue publiquement le **22 mars 2021**,
devant le jury composé de :

Monsieur Ian O'CONNOR
PROFESSEUR, Ecole Centrale de Lyon, Rapporteur
Monsieur Gilles SASSATELLI
PROFESSEUR, LIRMM, CNRS, Rapporteur
Monsieur Frédéric ROUSSEAU
PROFESSEUR, TIMA, UGA, Président du jury
Madame Edith BEIGNE
INGENIEUR HDR, Facebook, Examinatrice
Monsieur Alexandre LEVISSE
INGENIEUR DOCTEUR, EPFL, Examinateur
Monsieur Subhasish MITRA
PROFESSEUR, Stanford University, Co-directeur de thèse
Monsieur Henri-Pierre CHARLES
INGENIEUR HDR, CEA, UGA, Co-directeur de thèse

Remerciements

Although this manuscript is written in English, I must thank the people who helped, surrounded, inspired and supported me during these three years of thesis.

Me voilà arrivé à la fin, ou presque, ou plutôt au début de quelque chose de nouveau. La thèse est un marathon de trois années, mais qui passe aussi vite qu'un sprint de quelques secondes. Heureusement, j'ai appris à en apprécier chaque instant et chaque rencontre qui se présentaient à moi. Plus qu'une expérience technique, j'ai vécu une expérience humaine. Une sorte de voyage introspectif, qui m'a changé profondément, sur ma vision de la recherche, mais aussi sur le monde qui m'entoure. Grâce à vos remarques, vos conseils et vos opinions, j'ai commencé à bâtir modestement ma propre vision. Aujourd'hui, je suis empli de doutes, mais je suis convaincu qu'on trouvera des solutions, je suis confiant. Pour tout ce que vous m'avez apporté, je me dois de vous remercier dans les paragraphes suivants. Je vous pris de m'excuser, si par mégarde, j'ai oublié de vous citer, sachez que je vous remercie.

Pour commencer, je tiens à remercier Monsieur **Ian O'Connor**, professeur à l'École Centrale de Lyon, et Monsieur **Gilles Sassatelli**, professeur à l'Université de Montpellier, pour avoir pris le temps de revoir mon travail. J'aimerais aussi remercier Monsieur **Frédéric Rousseau**, professeur à l'Université Grenoble Alpes, Madame **Edith Beigné**, ingénieure de recherche chez Facebook, ainsi que Monsieur **Alexandre Levisse**, docteur à l'École Polytechnique Fédérale de Lausanne, pour avoir accepté d'examiner attentivement mon manuscrit.

Pascal, je ne sais comment te remercier pour ton implication sincère et tes remarques toujours pertinentes. À nos discussions du vendredi soir et celles un peu plus tardives pour les soumissions de papier, désolé Sonia. À nos voyages (professionnels) à Stanford, à nos parties de bowling et nos ballades dans la nature américaine. Merci d'avoir pris de ton temps, merci pour ton attention et merci pour ton exigence infinie. Cette thèse n'aurait pas été la même sans toi, merci infiniment du fond du cœur. **Henri-Pierre**, merci pour ton approche de la recherche et de la vision logicielle. Merci pour avoir repris mon encadrement et de m'avoir conseillé tout en me laissant le dernier mot. Je me souviendrais de nos échanges cordiaux, parfois engagés, entre logiciel et matériel. De ces débats, j'en suis sorti grandi intellectuellement, pour cela, je t'en suis reconnaissant. **Edith**, je tiens tout particulièrement à te remercier, pour m'avoir fait confiance, aussi bien humainement que scientifiquement. Tu m'as apporté beaucoup en peu de temps et tu as su me guider et me faire réfléchir aux bonnes questions. Je me souviens de nos sympathiques discussions à Stanford, merci pour ta jovialité, ton ouverture d'esprit et à bientôt aux USA ! **Subhasish**, thank you for all your always relevant and thoughtful remarks that you have brought to our discussions and my scientific flourishing. Sorry for my often inaccurate English, and thank you for all the opportunities you were able to offer me during our collaboration.

Je remercie le CEA Grenoble pour m'avoir donné cette magnifique opportunité de thèse, merci à l'ex-LETI et au nouveau LIST, à l'ex-DACLE et au nouveau DSCIN, à l'ex-LISAN et aux nouveaux LFIM, LIIM et LSTA. Merci **Yvain T.** et **Jean-Fred C.** pour leur goût de la personnalisation et des dotfiles - Yvain, j'éteindrais la lumière en sortant ! Merci **David C.** pour tes explications claires et tes goûts musicaux atypiques, **Manu P.** pour tes conseils avisés et nos discussions de barbus à sandales (IPoAC), **César F. T.** pour ta sympathie et ton enseignement sur les plateformes de simulation, **Ivan M.-P.** et **Romain L.** pour vos conseils et vos visions architecturales, qui m'ont aidé à créer la mienne, **Simone M.** et **Marjorie G.** pour votre écoute attentive et humaine, et **Andrea B.** pour ces merveilleux moments passer avec toi au Pérou.

Merci à l'équipe mémoire pour m'avoir accueilli en cours de chemin, merci **Jean-Phi N.** et **Bastien G.** pour votre passion cinéphile d'OSS 117 et de Kaamelott, ainsi que des viennoiseries légères : *je suis prêt à vous dire au revoir un par un.* Merci **Maha K.** pour ta patience, ton écoute et ta gentillesse, **Lorenzo C.**, notre Leonardo Da Vinci du design mémoire, et **Valentin G.** pour nos avis échangés sur le monde de la recherche. Merci à tous mes co-bureaux pour m'avoir supporté pendant ces trois longues années, **Julie D.** et **Riyane S. L.** pour votre amicale compagnie et votre ouverture d'esprit (le code, c'est la vie !), et à l'inséparable équipe des Neurouxnes, **Capucine L. d. B.** (aka Clémentine), **François R.** et **Thomas M.** pour m'avoir accueilli comme l'un des vôtres et pour votre humour noir qui a su éclairer mes journées.

Un merci tout particulier pour **Maxence B.** et **Valentin E.**, mes copains thésards et futurs docteurs, avec qui j'ai partagé mes doutes et questionnements personnels tout au long de mon parcours de thèse. Allez, c'est bientôt la fin ! Bon courage aussi à tous mes amis thésards, **Miguel S.**, **Sota S.**, **Eduardo E.**, **Mona E.**, **Kevin M.**, **Stéphane B.** et **Manon D.**, la libération se rapproche de jour en jour ! Bonne continuation à notre alternant préféré, **Antoine P.**, et merci à tout les thésards du LGECA, **Nicolas G.**, **Loïck L. G.**, **Carlos A. B.**, **Housseim E. D.**, **Adrien M.** et **J-B**, merci pour toutes nos discussions plus ou moins scientifiques en salle café.

Un gros merci aux coupains, à la Kacoloc, **Antoine** et **Tanguy** (fraîchement docteur), autant majestueux dans le A-clique que dans les régimes diététiques (on le fera un jour ce marathon !), sans oublier notre quatrième coloc **Piat**, locataire à vie de la Belle Électrique. Merci **Adrien**, avec qui j'ai pu refaire le monde, au moins un bon milliard de fois, bon courage pour ta thèse, tu le mérites. Merci **Amaury** et **Philippine** pour cette colocation improvisée, vous êtes des amours. Merci **Maël**, ex-protoss et nouveau terran, là on est bien ! Merci et félicitation à la nouvelle petite famille **Zoé**, **Christophe** et **Astrid**, merci **Rémi**, **Margot**, **Axel**, **Maëva**, **Hochard S.**, **Vincent** et **Léo** pour votre soutien. Une dernière dédicace aux artistes du blindtest, **Amélie**, **Emilie**, **JR**, **Agathe**, **Adrien**, **Mateo**, **Alexandre**, nos petites soirées au métro me manquent.

Pour terminer, je tiens à remercier chaleureusement toute ma famille qui a su comprendre les enjeux et discerner les difficultés de la thèse. Je ne saurais comment remercier suffisamment, ma mère, pour son ouverture d'esprit, sa dextérité manuelle et sa pugnacité qu'elle m'a donnée, mon père, pour sa persévérance de réflexion et de questionnements qu'offre le métier de chercheur scientifique, ma sœur, pour sa sympathie altruiste, sa créativité et son humour, et mon frère, pour sa détermination et son goût de l'aventure, qui me pousse, aujourd'hui à partir vers d'autres contrées lointaines.

Abstract

Exploration of Reconfigurable Tiles of Computing-in-Memory Architecture for Data-intensive Applications

Current computing architectures for data-intensive applications are facing severe memory access limitations. Power-hungry caches are not efficient anymore, the memory available to the cores is more and more limited in both capacity and bandwidth. Unfortunately, the trend in memory technologies does not scale as fast as the computing performances, leading to the so called memory wall. To address these challenges, the main directions followed in the research community are both at the architecture level and technology level: new architecture with computation immersed in memory, coupled to on-chip Non-Volatile Memory (NVM) for increased density, and advanced 3D architectures for increased memory capacity while offering more tightly coupled computing and memory. As a first step towards these directions, this PhD thesis addresses the architectural study of computation immersed in memory architecture with an increased memory sizing, scalability and reconfigurability using standard CMOS technologies.

Recent techniques that bringing computing as close as possible to the memory array such as, In-Memory Computing (IMC), Near-Memory Computing (NMC), are expected to address these limitations, but are facing limitations such as, fixed vector size and total available memory capacity. To process data-intensive applications with larger datasets, in this thesis, I propose a scalable and reconfigurable tile-based architecture composed of SRAM-based NMC tiles, each enabling arithmetic and logic operations within the memory. The combination of a horizontal scalability scheme and a vertical data communication offers an adaptive vector size for maximum performance onto the NMC tiles. In terms of programming model, this architecture can be programmed as an accelerator and executes vector-based kernels available on existing SIMD engines. For architecture exploration, performance and energy of data-intensive kernels are quantified using an instruction-accurate simulation platform using SystemC/TLM, calibrated on existing NMC SRAM tile designed in 22 nm FDSOI technology. Compared to 512-bit SIMD architecture, the proposed NMC architecture achieves an Energy-Delay Product (EDP) reduction up to 52× and 71× for linear and quadratic computational complexity kernels, respectively.

Exploration d'une Architecture Tuilée Reconfigurable de Mémoire Calculante pour les Applications Gourmandes en Données

Les architectures de calcul actuelles dédiées aux applications gourmandes en données sont confrontées à de graves limitations d'accès à la mémoire. Les caches gourmands en énergie ne sont plus efficaces et la mémoire disponible pour les processeurs est de plus en plus limitée en termes de capacité et de latence. Malheureusement, l'évolution technologique des mémoires ne s'adapte pas aussi rapidement que les performances de calcul, ce qui conduit à ce que l'on appelle le "mur mémoire". Pour relever ces défis, les principaux axes de recherche suivies dans la communauté se situent à la fois au niveau de l'architecture et au niveau technologique : des nouvelles architectures avec du calcul immergé dans la mémoire, couplée à une mémoire non-volatile (NVM) sur puce pour une densité accrue, et des architectures 3D pour une capacité de mémoire accrue tout en offrant un couplage étroit entre le calcul et la mémoire. Afin d'avancer vers ces directions, cette thèse de doctorat porte sur l'étude architecturale du calcul immergé dans la mémoire, de son dimensionnement, son extensibilité et sa reconfigurabilité en utilisant des technologies CMOS standard.

Les techniques récentes qui rapprochent le plus possible le calcul de la matrice mémoire, telles que le calcul en-mémoire (IMC) et le calcul proche-mémoire (NMC), devraient permettre de résoudre ces problèmes, mais sont confrontées à des limitations telles que la taille fixe des vecteurs et la capacité totale de mémoire disponible. Pour traiter des applications avec des ensembles de données plus importants, je propose dans cette thèse, une architecture modulable et reconfigurable basée sur des tuiles NMC à base de SRAM, chacune permettant des opérations arithmétiques et logiques au sein de la mémoire. La combinaison d'un schéma d'extensibilité horizontale et d'une communication de données verticale offre une taille de vecteur adaptable pour des performances maximales sur les tuiles NMC. En termes de modèle de programmation, cette architecture peut être programmée comme un accélérateur et exécute les applications vectorisées disponibles sur les accélérateurs SIMD existants. Pour l'exploration architecturale, les performances et l'énergie des applications à forte intensité de données sont quantifiées à l'aide d'une plateforme de simulation précise à l'instruction, utilisant le langage SystemC/TLM et calibrée sur une implémentation de la tuile NMC en SRAM conçue avec la technologie FDSOI 22 nm. Par rapport à l'architecture SIMD 512 bits, l'architecture NMC proposée permet une réduction énergétique et des délais (EDP) allant jusqu'à 52× et 71× pour les applications à complexité de calcul linéaire et quadratique, respectivement.

Contents

List of Figures

List of Tables

List of Listings

Introduction

Flow of data are created every day by millions of people through computers, smart phones, autonomous cars or medical devices. The quantitative explosion of digital data, known as "Big Data", challenges us to find innovative approaches to capture, share, store and analyze such data. Data Science is an inter-disciplinary field that combines scientific methods, statistics and mathematics analysis to extract relevant and valuable information based on large amount of data rather than computation. Such applications require a strong data accessibility constraint compared to the data processing locality.

As shown in Figure 1, the combination of this data knowledge with Artificial Intelligence (AI) applications has improved the interpretation and accuracy of predictions but mainly the interaction with the physical world, closer to human behavior in robotics, expert systems or computer vision applications. Furthermore, the massive data processing by Machine Learning (ML) algorithms, provides the ability to make predictions or take decisions, difficult (or even impossible) to obtain with conventional algorithms, such as medical diagnosis, speech recognition or natural language processing applications. The ML methods of classification or clustering allow the prediction of data groupings based on the extraction of the main features by a massive database pre-processing. Differently from the ML, the Deep Learning (DL) applications directly exploit the raw data to learn these key features by successive training, without prefiltering or human interactions. To complete human-like tasks, artificial Neural Networks (NN)

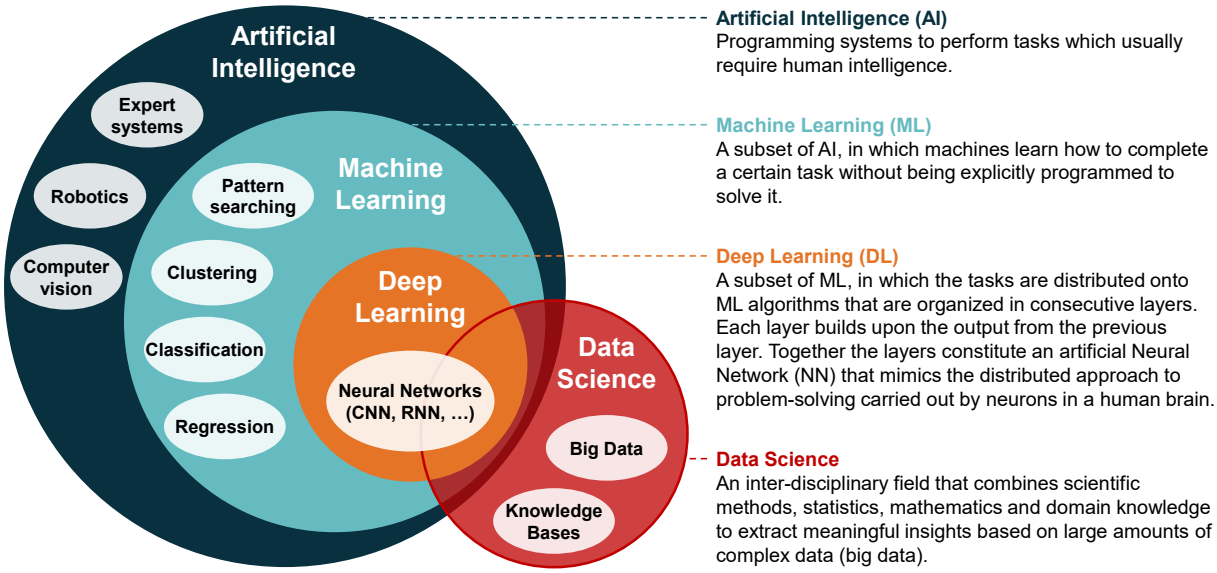


Figure 1: Artificial intelligence, data science, machine and deep learning overview.

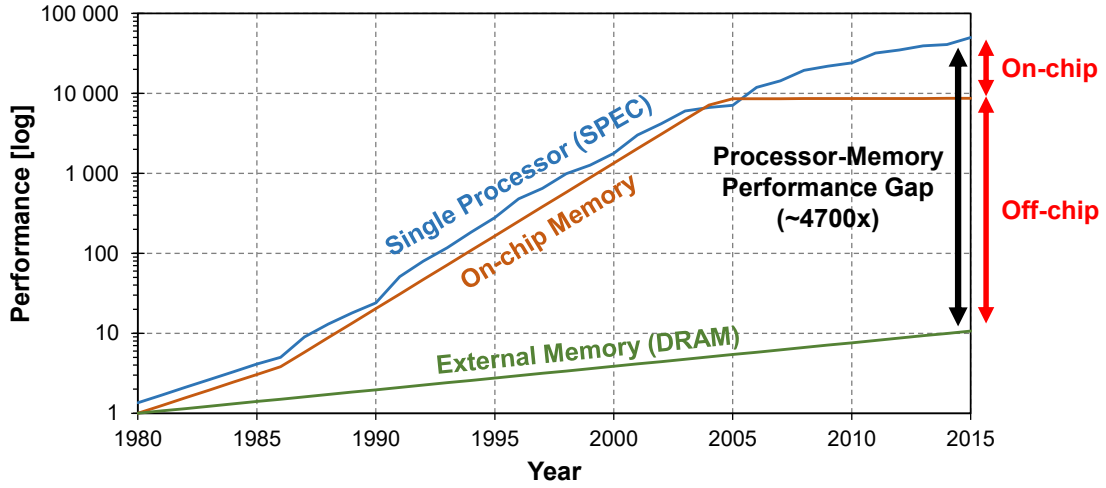


Figure 2: Growth in processor and memory performance over 35 years [2]. Processor performance is measured by the SPEC's benchmark suite and compared to the VAX 11/780 architecture (1980). On-chip memory performance depends on the processor memory requests and external memory performance depends on the DRAM access latency.

are used in a wide range of applications to perform contextual object recognition (e.g. Convolutional Neural Network (CNN)), natural language interpretation (e.g. Recurrent Neural Network (RNN)), medical image analysis, bio-informatics, Computer-Generated Imagery (CGI), and so on. Recently, a major breakthrough in protein structure prediction is achieved by *DeepMind* using the *AlphaFold* neural network to determine the three-dimensional shape of a protein from its amino acid sequence [1]. The scope of big data applications is wide and the impacted domains are diverse, from climate predictions to autonomous driving, but the cost of processing this large amount of data is becoming more and more challenging for computers.

Today's conventional computer architectures are struggling to satisfy the performance and energy requirements for these data-intensive applications. The evolution of modern architectures has moved towards High Performance Computing (HPC) and distributed systems in order to explore new ideas in the architecture of massively parallel machines and software, such as the *Blue Gene* project developed by *IBM* [3] or modern Graphics Processing Units (GPUs). However, the cost of transferring data from the external off-chip memory to the processor is still very high compared to the cost of the computation itself. This processor-memory performance gap is called the "Memory Wall" [4], as presented in Figure 2. Until 2005, architectural solutions such as DMA units and high-speed on-chip memory Cache close to the processor bridged the gap between external off-chip memory and processor speeds. Thus, two challenges must be addressed by modern architectures to reduce these gaps: by integrating new memory technologies to solve the off-chip memory wall, and computation immersed in memory to solve the on-chip memory wall.

Moreover, technological advances have improved the performance of the processor architecture. The trend of these evolutions is closely related to Moore's law [5] (1965) which, through observation of the silicon electronics industry, observed that the number of transistor devices on a chip doubled each year. This steady growth is driven by three factors: larger chips, smaller devices that allow more devices per unit area and design cleverness innovations that making better use of available space on a chip. In 2020, this law is still verified in the design of conventional architectures. Regarding the scaling aspects of the technology, Dennard's law [6]

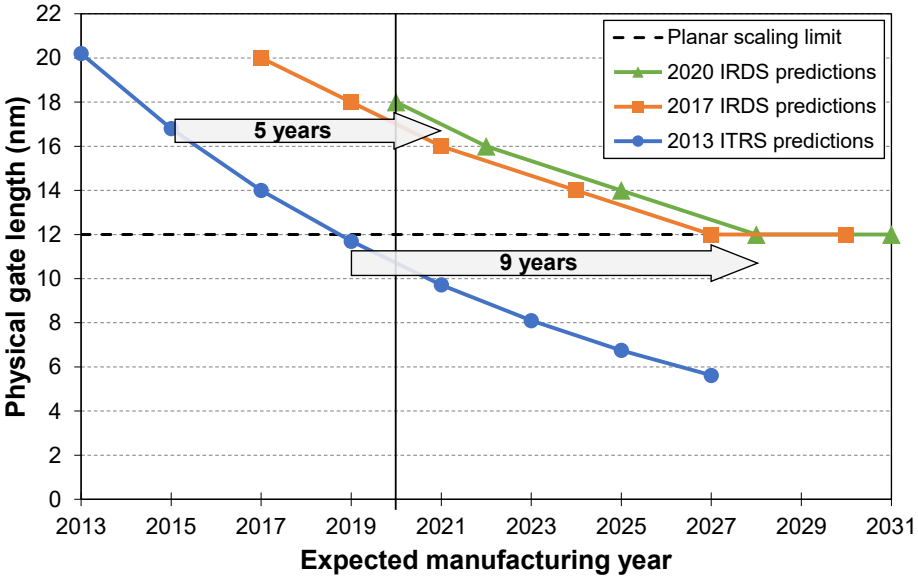


Figure 3: Slow down of the scaling era, based on transistor dimension predictions over time. These numbers are extracted from ITRS (*International Technology Roadmap for Semiconductors*) [7] and IRDS (*International Roadmap for Devices and Systems*) [8] reports.

(1974) refers as the proportional relationship between the area dimensions of the transistor in relation to its power density. As the transistor size shrunk and the voltage is reduced, the circuits could operate at a higher frequency with the same power. However, around 2005, this observation ends because the law ignores the impact of static energy (leakage) and transistor threshold voltage. Thus, the scaling of the transistor faces the so-called "Power Wall", implying the limitation of processor frequency to 4 GHz since 2005. As a side effect, the processor frequency limits the on-chip memory bandwidth, leading to an on-chip processor-memory performance gap as shown in Figure 2.

The trends of circuit technologies is followed up and forecasted by the ITRS (*International Technology Roadmap for Semiconductors*) [7] and IRDS (*International Roadmap for Devices and Systems*) [8] roadmaps. Due to the complexity of upcoming technologies, the roadmap predicts and inquires not only devices, but also circuits, systems, and differentiating technologies. In Figure 3, the optimistic predictions of 2013 are becoming harder and harder to achieve and the expectations of technology nodes are slowly approaching the physical limits of the transistor. Indeed, transistor shrinkage is due to the physical gate length of the transistor and the scaling slows down to a physical limit around 12 nm for planar transistors such as MOSFET (*Metal-Oxide-Semiconductor Field-Effect Transistor*). Indeed, this "Scaling Wall" is pushed back thanks to recent technological design breakthroughs, such as FinFET (*Fin Field-Effect Transistor*) process technology to build non-planar transistors.

Tomorrow's applications will become more and more data intensive, especially to solve problems by learning from a large knowledge base but also general-purpose applications. Unfortunately, due to the memory, the power and the scaling walls, conventional computer architectures are limited in performance and energy-efficiency, and technologies are more and more constrained by physical limitations. Many solutions have been proposed but emerging architectures and technologies must be considered to reverse these paradigms. Architectural challenges are expected to increase the connectivity density such as advanced 3D architectures

or bringing the computation closer to memory in order to limit memory transfers and increase the bandwidth. Application-specific and reconfigurable architectures are promising opportunity to increase energy-efficiency by dynamically reconfiguring the data flows. New memory breakthroughs are also being explored to develop dense on-chip memories, using non-planar transistors to store more and more data.

This PhD thesis addressed the memory wall problem by proposing an new architectural and technological vision. In this context, I developed a reconfigurable architecture composed of memory-based computing tiles to reduce the on-chip memory wall impact. Each tile can perform computations with data stored in the attached memory. To evaluate the performance and energy gains of this architecture, I studied and modeled the interconnect memory scalability and proposed transfer mechanisms to move data and computation between different tiles. At the software level, this memory architecture is configured by a processor that supports the instructions to be executed by each tile as well as its interconnect reconfiguration mechanisms. In addition, this architecture is simulated at the hardware and software levels through a in-house simulator to explore different data-intensive kernels used in the scope of image processing, database searching and neural networks applications.

Outline of the Thesis

This thesis manuscript is composed of eight chapters, not including the introduction and the conclusion:

- *Chapter 1 - Energy Efficient Computing Architecture Challenges*, presents the background and motivation of this work and the context of big data applications and computer trends. Moreover, this chapter provides an application-driven approach to size data-intensive architectures and identify conventional computer system bottlenecks.
- *Chapter 2 - State-of-the-Art on Energy Efficient Distributed Emerging Architectures*, discusses about the state-of-the-art in emerging architectures for data-intensive applications. Vector architectures, memory-based computing technologies and 3D implementation opportunities will be studied to assess the impact of these distributed architectures.
- *Chapter 3 - A Dream: a 3D Stacked Distributed Computing Architecture*, proposes a architectural vision in order to break the memory wall paradigm, enhanced by the technologies presented above. This chapter summarizes the contributions of this thesis and which questions this work intends to answer in the context of a more comprehensive 3D architecture that still an going research topic.
- *Chapter 4 - A Reconfigurable Memory-based Computing Architecture Proposal*, presents a scalable and reconfigurable and scalable memory-based computing architecture. To increase the available memory space and associated computing capability, we propose to assemble a set of memory-based computing tiles in a configurable fashion. This allows to extend computing vector in two manners: either a horizontal memory extension allowing larger vectors, or a vertical memory extension allowing more vectors.
- *Chapter 5 - Software Integration for Scalable Vector Computing*, presents software and system integration of specific instructions for near-memory operation accelerators' implementation. In order to address kernels with larger dataset and real applications, it is required to scale up the architecture with more memory, while proposing the adequate programming model allowing vector acceleration.
- *Chapter 6 - Design Space Exploration of the Memory Interconnect*, explores trade-off of the wiring interconnect cost in multiple memory tile designs in terms of power, performance and area. This design space exploration are performed with a standard digital integration flow to extract low-level performance numbers for accurate architectural explorations.
- *Chapter 7 - ArchSim: an IMC-NMC Software-Hardware Simulation Platform*, presents an hardware/software simulation platform to evaluate emerging technologies and architectures onto standard systems. All transaction events are considered between hardware components and its integration into a standard software tool chain.
- *Chapter 8 - Architectural Exploration Results*, discusses the results of the proposed architecture evaluated with the proposed simulation platform. The proposed scalable vectorization scheme allows to directly re-use existing vector application kernels to explore the execution speed up, the energy reduction and the energy delay product onto different architectures.

Chapter 1

Energy Efficient Computing Architecture Challenges

Contents

In this chapter, I discuss the application background and motivation of this thesis. Hence, I propose an overview of data-intensive applications, their algorithmic dimensioning constraints and their impact on conventional memory architecture. This application-driven approach provides key parameters to identify the conventional architecture bottlenecks and limitations. Finally, I detail the context of the "memory wall" problem caused by the growing disparity between the speed of the processor and the memory latency.

1.1 Introduction

Data-intensive applications, often referred to Big Data applications, involves a number of disciplines, including statistics, machine learning, neural networks, signal processing, pattern recognition, optimization methods and visualization approaches. These are classes of applications containing a large set of data (usually terabytes or petabytes) and various types of data, for which it becomes difficult to integrate into a conventional system. Today's architectures face many challenges and difficulties related to the capture, storage, sharing and analysis of this large amount of data [9]. Moreover, most architectures are agnostic to the data-intensive application requirements. In fact, processors are optimized in latency and throughput for a set of generic functions to execute all types of operations for the largest range of applications. Conventional architectures are computationally efficient but are limited due to data access, which is called the memory wall problem [4].

Thus, an application-driven approach should be proposed to identify the application requirements of these applications to evaluate the architecture data path flexibility [10]. It is appropriate to determine the computational precision (operation bit width), the computational complexity (type of operations) and the amount and data locality of the workload (data accesses). Figure 1.1 provides a high-level overview of the main data-intensive applications grouped by classes, specific to signal processing, optimization, machine learning and scientific computing.

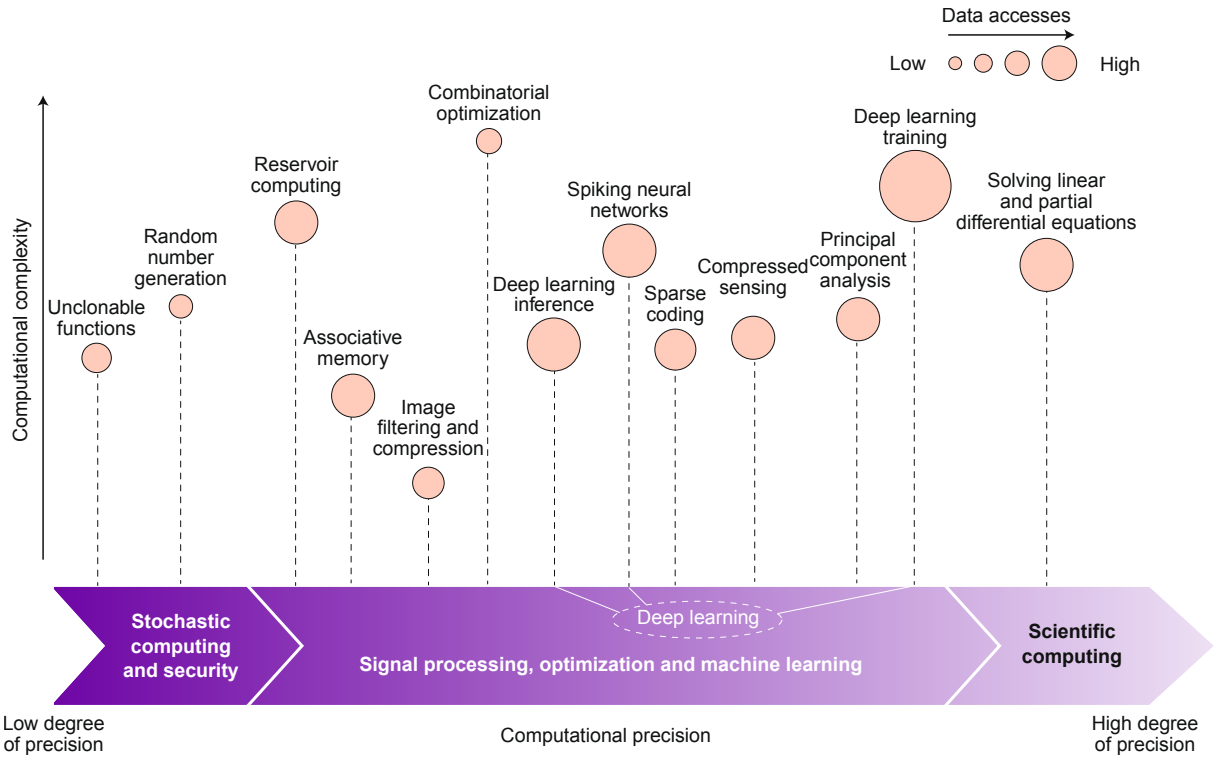


Figure 1.1: Landscape of data-intensive applications, according to computational complexity, precision and data accesses [10].

Signal processing applications are used to extract, modify and manipulate signals such as sounds, images or sensor measurements to improve analysis, transmission or storage efficiency. As a subset of this category, image processing is based on 8-bit precision arithmetic operators per colour channel for image filtering and compression, as detailed in Section 1.3.1.

Associative memory is used in several database search applications to save searching time. Also known as Content Addressable Memory (CAM), it is a special type of memory that is optimized to perform searches through data, as opposed to simply accessing data directly based on the address. Other search algorithms are used in bio-informatic applications for pattern matching in a DNA sequence, as detailed in Section 1.3.2. Deep learning is a subset of machine learning applications for neural network inference (low-complexity and low-precision) and training (high-complexity and high-precision), as detailed in Section 1.2. Quantized Neural Networks (QNN) is a promising alternative to reduce data movement and computational complexity, as detailed in Section 1.2.2.

Finally, scientific computing applications involve a compute-intensive approach where a massive amount of complex computations, usually high-precision floating-point operations, are executed by supercomputers on HPC platforms. These applications still have a rather high computation ratio compared to memory accesses, which reduces the memory wall impact but does not solve the problem. This type of compute-intensive applications has not been evaluated in this PhD thesis but the proposed long-term architectural vision would bring significant advantages in terms of performance and energy reduction.

1.2 Neural Network Applications

The purpose of artificial Neural Networks (NNs) is to mimic the human brain behavior in order to solve problems by learning without being explicitly programmed to solve them [11]. The model of the artificial neuron, initially called the perceptron (1958), is organized in successive layers, the output of one layer being the input of another, as shown in Figure 1.2. The inputs x_n of each neuron are weighted by weights w_n, called synapses, and then summed in y_n, this operation is called dot-product. From the result y_n, an activation function could be applied to each layer. The NN topology, defining the connectivity between the layers, allows applications such as recognition, detection, interpretation or classification of image, text or audio language. The main advantages of NNs compared to conventional applications relate to (1) the learning of new tasks by training and (2) no pre-processing steps are required to extract the main data components (features), which are analyzed by hand in conventional applications.

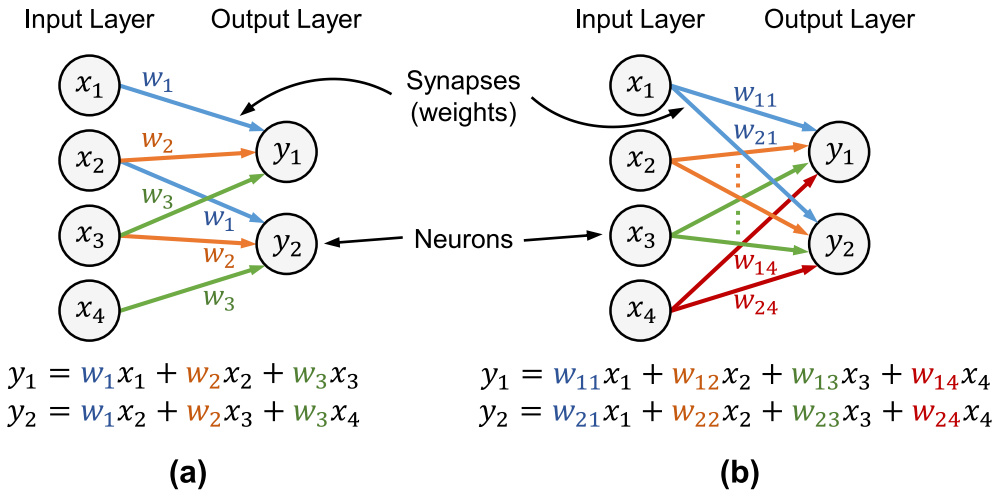


Figure 1.2: Simple neural network layer examples and terminology. **(a)** Convolution kernel representation. **(b)** Fully connected kernel representation.

In the topology of NNs, the data flow moves from layer to layer, from input to output, also called the forward propagation or inference. When learning new inputs, the training dataset is successively forward-propagated, then back-propagated in order to update the network weights according to the quality of the output score. To speed up the learning process on current architectures, the input data is stacked in a batch to process more than one input data in parallel, the so-called batch size. As the learning step requires a lot of energy, power, process time and memory resources, different architectures are used for training and inference, respectively. As both processes could use independent architectures, the updated weights of the training are transferred to the target architecture, in this thesis we will mainly focus on the inference stage.

A wide range of Deep Neural Network (DNN) (NN with more than three layers) topologies exist, using custom activation functions to solve specific tasks. CNNs are most often used to process and analyze images. They are mainly composed of a convolutional kernel layer, shown in Figure 1.2(a), and a fully-connected kernel layer, shown in 1.2(b), are detailed Section 1.2.1. For text or audio recognition, Recurrent Neural Network (RNN) topologies, of which Long Short-Term Memory (LSTM) is a popular variant, are more suitable for processing temporal information [12]. They use different connectivity at the layer level and have looped synapses on the neurons, allowing an intermediate state to be maintained to store past events. The main difference between CNN and RNN is in the specificity of the activation functions, while both NN use weighted sums between layers.

1.2.1 Image Classification

Computer vision competitions such as ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [13] evaluates algorithms for object detection and image classification by annotation. In 2012, a CNN model, named AlexNet [14], won this competition by a large margin. Since then, many DNN architectures based on this model have been studied to perform image classification, such as a typical representation is shown in Figure 1.3. Input layer data are images, representable by pixel arrays consisting of three 8-bit channels, red, green and blue. The pixel arrays go through many kernel convolution layers and activation functions such as non-linearity (e.g. sigmoid, Rectified Linear Unit (ReLU), ...), normalization and pool-

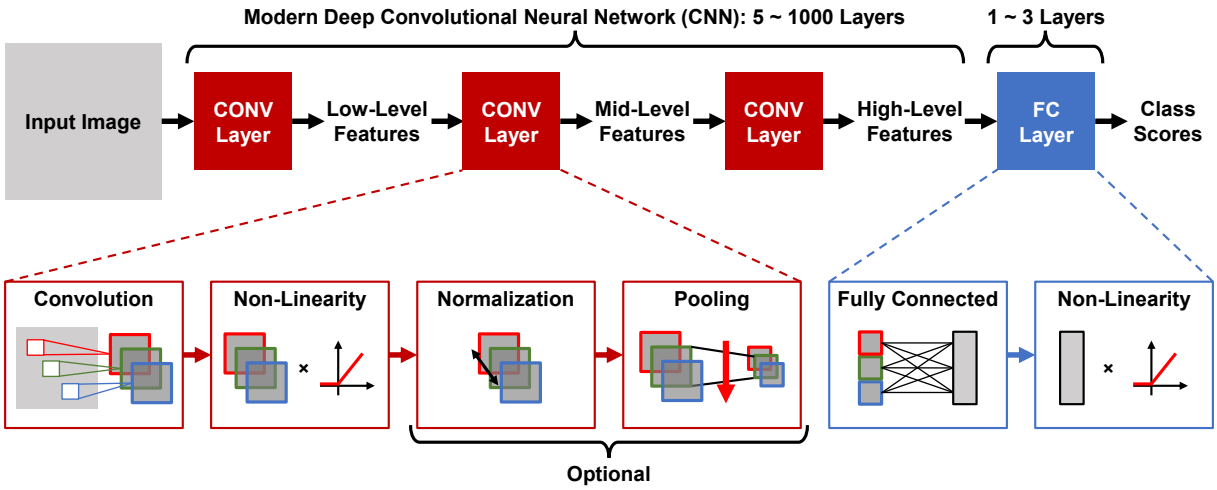


Figure 1.3: A typical Convolutional Neural Network (CNN) for image classification [11].

ing (down-sampling the activation maps), up to 1000 layers for some models. These layers extract the main features contained in the images, such as the distinctive shapes of objects, as far as identifying the object itself. Finally, the last Fully-Connected (FC) layers classify the object in the image by giving it a label and a score (accuracy). Main operation of the CNN consists in executing the matrix-vector Multiply-ACcumulate (MAC) operation during inference (for a single image) and matrix-matrix operation during training (for a set of images). Today, frameworks such as TensorFlow (*Google*), Torch (*Facebook*), Scikit-learn (*Inria*) integrate a set of tools and functions which optimize the program execution according to the hardware resources available on the Central Processing Unit (CPU) (vector units) and the GPU (tensor units), involving the MAC operation. To improve NN application execution, the basic MAC operation is integrated into the instruction set of our proposed architecture in Chapter 5.

Moreover, the principal metric for comparing NNs is the quality of the results, defined as the percentage of accuracy. At the architectural level, other metrics matter, such as (1) throughput, crucial for real-time systems, (2) latency is used to evaluate application interactivity, crucial for autonomous driving applications, (3) energy and power, crucial for embedded systems, (4) architecture flexibility, to maintain efficiency whatever the type of DNN models, and (5) scalability, to maintain the performance scale with increasing memory resources [11]. All these metrics depend on the number of Processing Element (PE), the number of MACs executed in parallel, the distribution of the PE but also on the amount of memory reusable in the computations according to the memory hierarchy of the architecture. Although optimizations to reuse data exist on GPUs [15], they require a huge amount of data compared to standard applications, and still present a memory wall inside the memory hierarchy (see Section 1.4).

1.2.2 Quantized Neural Network

Quantized Neural Networks (QNNs) use a reduced numerical data representation to encode weight values, activations and partial sums in order to reduce data movement and thus minimize the impact of the memory wall. For the same architecture, the memory bandwidth is increased and the memory size used to store the weights is reduced according to the quantification step. Obviously, there are trade-offs between the computation precision and the accuracy of the result in relation to the bit width of the data. Many challenges and research topics exist around the hardware and the architecture because the systems developed are custom and for a dedicated problem [16]. The observation of the weight values in the different neuronal layers shows a non-uniform distribution (close to a Gaussian) in relation to the depth of the layer in the network. A non-uniform quantization (logarithmic or with lookup tables) may be used to represent the weight values [17].

Moreover, system power consumption is reduced due to reduced memory resources and low computational precision to perform MAC operations. As shown in Figure 1.4, the energy cost of data access is 50× to 6,400× more costly than a 32-bit addition for on-chip and off-chip access, respectively. Thus, operation bit width is a critical factor to reduce the energy consumption of the architecture. By changing from a 32-bit floating addition to an 8-bit fixed addition, the energy consumption of the operation is reduced by 30×, and the transfer of this data is reduced by 4×. Binary Neural Networks (BNNs) reduce computational precision up to 1 bit resolution, enabling bitwise operators to be used instead of arithmetic operators, further reducing data transfer by 32×, such as XNOR-Net topology [18]. These Quantized Neural Networks (QNNs) are slower to train and could have additional hardware cost, but they are

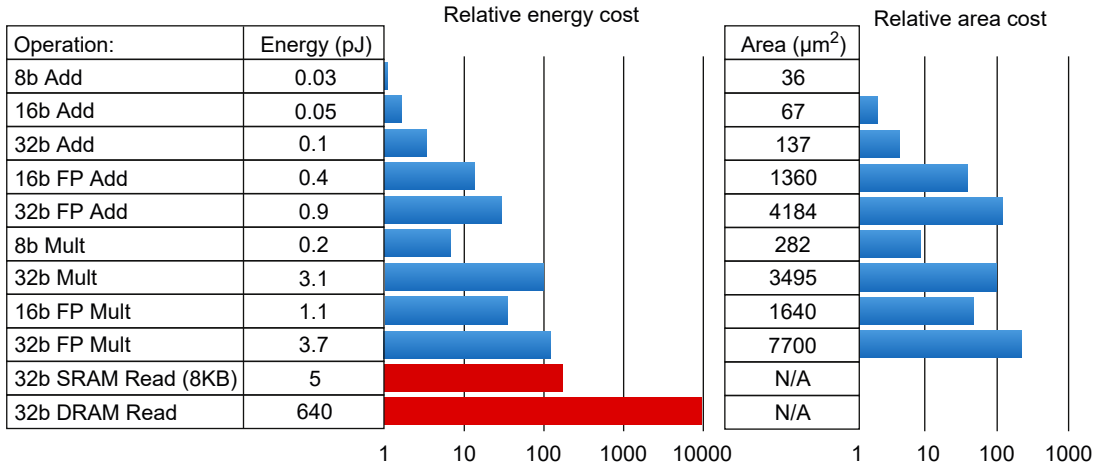


Figure 1.4: Energy and area costs of arithmetic operations and accesses to SRAM and DRAM [2]. Area is for TSMC (*Taiwan Semiconductor Manufacturing Company*) 45 nm technology node.

nearly as accurate as CNNs related to the quantification step. They demonstrated a 5% loss of accuracy with a 5-bit fixed-point resolution compared to a 32-bit floating-point resolution on the ImageNet dataset [16], by using a smaller memory footprint to store weights, reducing the impact of the memory wall.

1.3 Other Data-intensive Applications

1.3.1 Image Processing

Digital image processing is the extraction and the manipulation of digitized images through an algorithm to obtain enhanced images or to extract meaningful information from them. It involves a wide range of algorithms, including edge detection, noise reduction, geometric transform, filtering (interpolation), segmentation, compression and restoration. Images can be defined as a two-dimensional matrix of pixels arranged in row and columns. Pixels are whole elements defined by three 8-bit RGB (Red, Blue, Green) channels for colour images, a single 8-bit channel for black and white images or a single binary value (0 and 1) for binary images. Operation type varies according to the images properties or the algorithm, such as divisions, multiplications, arithmetic (addition, subtraction), logarithmic or cosine.

Lookup tables are used in image filtering, such as Fast Fourier Transform (FFT), and data compression, such as JPEG's lossy compression [19], to replace the cosine function with a table of pre-calculated values stored in a static program to save computing time or when there is no dedicated hardware mathematical unit (e.g. FPU). Compression algorithms apply matrix transforms and quantizations to take advantage of the sparsity of the output matrix and thus compress the null values. Thus, to speed up image processing applications, a reduced set of arithmetic instructions can be used to benefit from the vector acceleration available in conventional architectures.

To facilitate image manipulation, the applications include dedicated software libraries such as OpenCV (Computer Vision) or OpenBLAS (Basic Linear Algebra Subroutines) for CPU or cuBLAS for GPU. By studying the library's contents, the most relevant processing methods are:

vector addition, scalar-vector multiplication, vector-matrix multiplication and matrix multiplications. These libraries are optimized to speed up processing through vectorization and parallelization (e.g. loop unrooling, instruction reordering, ...). Instruction scheduling and data accesses are optimized to minimize the number of memory requests. Despite this, the scalability and the flexibility of digital image processing are constrained by the hardware resources and the memory hierarchy. Consequently, there are many specific software implementations (Strassen, Winograd, im2col [20], ...) and compilation challenges to reduce this memory-overhead depending on the structure of the matrix and data access patterns.

1.3.2 Database Searching

The amount of genetic DNA data obtained by next-generation sequencers doubles almost every year, leading to more sophisticated data structures for big data. Database searching is a fundamental problem in computational biology to find patterns in DNA sequences, but also in data servers to return the address location of matching words. Specific hardware architectures, such as associative memories (e.g. CAM) [21], are used to improve access latency to data servers but lack flexibility for pattern matching applications in bio-informatic.

Search algorithms, also known as exact pattern matching [22], return match for all occurrences found in a DNA sequence, composed of a finite alphabet (e.g. A, T, G, C) that could be changed according to the applications. Other algorithms, such as Hamming Distance, return a distance value corresponding to the number of different symbols present between two strings having the same length. Used to compare two DNA sequences, this algorithm is also used for error detecting and error correcting codes in coding theory. These algorithms use bitwise and logical operators and benefits from the bit-parallelism of the architecture to improve searching performances [23]. Hardware vector processor accelerators, such as SIMD (Single Instruction Multiple Data), have specific instructions to speed up the long pattern searching, but are physically limited by the maximum SIMD vector size (128-bit up to 512-bit on Xeon processors) and the data bandwidth between the on-chip memory and the SIMD units.

1.4 Conventional Architectures

Performance limitations of modern computer architectures mostly come from memory accesses, commonly named as the memory wall. One of the architectural solutions is to define a memory hierarchy using smaller high-speed on-chip memory closer to the processor to bridge the gap between external off-chip memory and processor speeds.

1.4.1 Memory Hierarchy

The aim of computer circuit architects is to create a design trade-off between memory capacity, memory performance and energy consumption whatever the application's hardware resources, according to technological constraints. As an introduction, we recall the main principle of memory hierarchy in computers, in order to separate the data and program memory into layers in order to minimize the access time and power activity and maximize the capacity. As shown in Figure 1.5, the memory hierarchy is composed of high-speed on-chip memories and high-capacity off-chip memories. Memories are becoming larger and larger in terms of capacity, but slower and slower in terms of performance relative to the proximity of the pro-

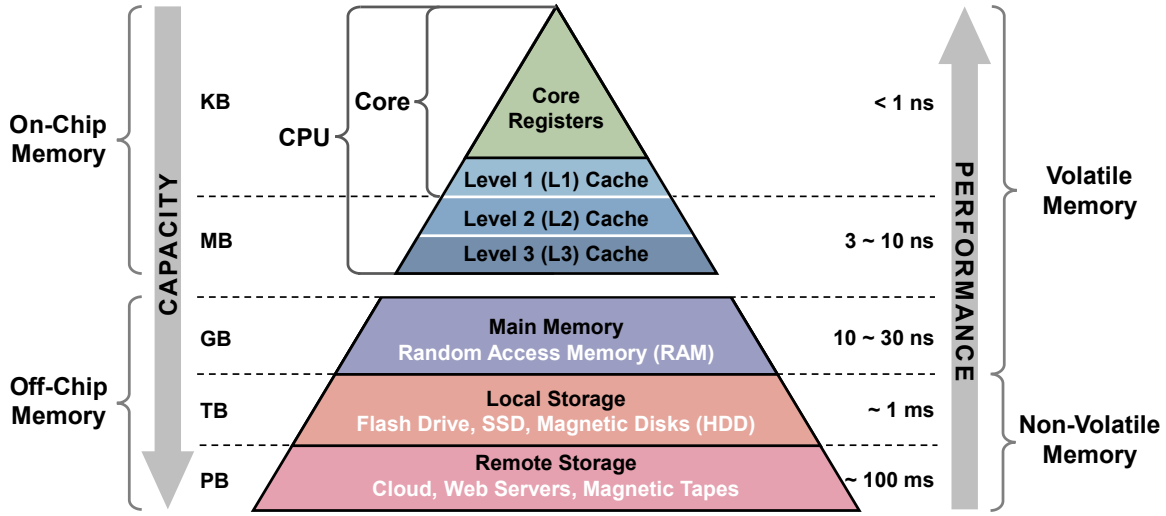


Figure 1.5: Memory hierarchy of a conventional computer architecture.

cessor. In this context, the chip refers to the processor or the CPU, which includes the cache memory hierarchy.

On-chip memories are composed of: core registers and the cache memory, divided into three levels: L1, L2 and L3/LLC (Last Level Cache), up to four levels in some multi-core architectures. The core is defined by its registers and L1 memory, which operate at the same frequency and store intermediate results, program loop counters or memory addresses. Additionally, the L1 cache is separated into two parts for instructions and program data. The Cache memory is a very high-speed memory which prevent from redundant data transfers. It is an extremely fast memory type that acts as a buffer between the main memory (or RAM) and the processor in order to reduce the average time to access off-chip data. It stores copies of data and holds frequently requested data and instructions so that they are immediately available to the CPU when needed. However, the sizing of these on-chip memories constrains the performance of the processor, thus limiting the maximum size of these memories. Current technologies no longer provide enough performance to minimize access times, mainly due to the scaling wall.

Off-chip memories are composed of: the main memory, also known as RAM (Random-Access Memory), the local Storage Class Memory (SCM), such as Hard Disk Drive (HDD) or Solid-State Drive (SSD), and the remote storage memory, often decentralized from the computer (Cloud). The main memory is used as a large buffer (today a few gigabytes) to transfer data from the external off-board storage memories to the processor. When a file is processed by the processor, this buffer contains the whole contents of the file. Storage memories (local and remote) are high-capacity non-volatile memories that are much slower than the processor, but whose non-volatility keeps the information even when the system is switched off. The memory wall problem occurs between the main memory and the processor [4]. As data-intensive applications can use files larger than the main memory, a memory wall between the external storage and the main memory remains.

Besides the data access latency, the bandwidth between these memories also has a significant impact. The transfer bus width between on-chip memories are 512-bit large, while the main memory is 64-bit large. In modern computer architectures, the evolution of SSD, whose bandwidth reaches speeds in the GB/s range. Most recently, new generations of SSD, mounted di-

rectly on wide ports, such as PCI (Peripheral Component Interconnect), are greatly improving the overall performance of applications, even non-data-intensive applications. After recalling the main principle of memory hierarchy, offering large memory capacity to local and fast cores, one can observe two main memory challenges : how to bring more memory on chip, and how to bring more memory to the local cores.

1.4.2 The Memory Wall in Micro-Architectures

The Cache memory is designed to hide the memory latency from the processor in order to minimize the impact of the on-chip memory wall. Hardware mechanisms of the Cache partly satisfy the core data requests. During a 32-bit data access, the Cache transfers continuous amounts of data per 512-bit line, to reduce the number of successive memory accesses when working on local data. Depending on the available data, memory requests are propagated to the upper layers of the Cache if the data is not present in the lower layers. Some data may be duplicated or updated between layers depending on the program's data workflow.

Furthermore, the cache placement policies determines where a memory line can be placed to prevent the overwriting of recent or reusable data. The hardware implementation and dimensioning of the cache is decisive for the execution of the application, as well as the writing of the application program. An application that performs irregular memory accesses will not benefit from the hardware mechanisms of the Cache [24]. Cache mechanisms satisfy general-purpose applications, but data-intensive applications suffer from the lack of connectivity between cache levels due to their small memory size and replacement policy.

1.5 Conclusion

Conventional architectures, such as CPU, are designed for general purpose applications but not overcome massive data movements in their memory hierarchy for data-intensive applications. These applications, such as NN and image processing, mostly involve MAC and vector-matrix multiplication and addition operations and specific data access could be optimized thanks to dedicated software libraries. By reducing the computational precision, QNN and database searching applications can benefit from bitwise and logical computing to increase the system energy-efficient and speed up their processing time. Moreover, all these applications presents little locality in computing, and reduced data reuse with very large datasets, for which standard memory hierarchy are inefficient. Cache memory aims to break down the on-chip memory wall but lack of flexibility and scalability caused by the placement policies and internal interconnect connectivity. Since the on-chip memory is designed for high-performance computing and the off-chip memory is designed for high-capacity storage, the memory wall problem remains. To solve this problem, new energy-efficient architectures must be designed by analyzing computing application requirements and optimizing data access resources. Emerging memory technologies and distributed computing are new opportunities to continue to break down the memory wall.

Chapter 2

State-of-the-Art on Energy Efficient Distributed Emerging Architectures

Contents

Performance limitations of conventional architectures mostly come from memory accesses, power and delay, commonly named as the memory wall problem. Hardware cache mechanisms and traditional memory hierarchy are obsolete to efficiently address modern data-intensive applications. Moreover, transistor scaling and technology trends are slowed due to the power wall, which limits the frequency of the system and thus the memory performance. Instead of increasing processor frequency, more and more processors are integrating domain-specific accelerators to improve the energy efficiency of redundant task processing. For instance, vector accelerators provide massive data parallelism that reduce the processing time. Another architectural solution to break the memory wall consists of pushing the processing units as close as possible to the memory to optimize data movement, also known as data-centric architectures. In this chapter, I present the state-of-the-art of vector accelerators, emerging memory technologies capable of computation, and dense 3D memory architectures in order to reduce the off-chip memory wall and increase the on-chip memory connectivity.

2.1 Optimized Accelerators using Standard Computing Paradigm

In the search for higher performance and energy efficiency, computing architectures are moving towards the use of specialized accelerators, to improve complex and repetitive tasks. Conventional architectures integrate many domain-specific accelerators, such as co-processors like SIMD, FPU, Digital Signal Processing (DSP) or dedicated NN accelerators, to perform operations optimized in performance and energy. Specific instructions from these co-processors are integrated into the core's extended instruction set and executed directly in the core's instruction flow. Computer systems also have accelerators external to the processor, such as GPU, which become processor-independent chip when hardware resources become more substantial. Thus, image processing and database searching applications would benefit from these vector accelerators (e.g. SIMD, GPU, ...) that support redundant tasks on massive amounts of data, contributing to improve energy efficiency, as detailed in Section 2.1.1.

Furthermore, the evolution of data-intensive applications exceeds the hardware performance of current architecture. As hardware struggles to adapt to new software challenges, programming flexibility become an important consideration in integrated circuit design. Thanks to custom hardware design, Application-Specific Integrated Circuits (ASICs) achieve the best energy efficiency for a dedicated task compared to CPU or GPU architectures. However, the lack of programming flexibility of ASIC leads to a short lifetime and expensive design costs. Reconfigurable architectures, such as Field-Programmable Gate Array (FPGA) or Coarse-Grained Reconfigurable Architecture (CGRA), provide a trade-off between programming flexibility and system energy efficiency, as detailed in Section 2.1.2. Thus, NN topologies are better suited to reconfigurable architectures as compared to conventional architectures that lack of memory communication flexibility due to the static Cache architecture.

2.1.1 Software Paradigm for Vector Architectures

Parallel computing techniques, such as data, instruction and task parallelism, are used to improve computing performance and energy efficiency of conventional architectures. Data-Level Parallelism (DLP) consists of executing the same instruction on a set of data at the same time, which is common for vector accelerators (e.g. SIMD). Instruction-Level Parallelism (ILP) arises because multiple instructions could be executed simultaneously during the program execution. It exploits DLP at hardware level, where the processor decides to execute instruction in parallel (pipelining, superscalar, out-of-order execution, ...), and at software level, where the compiler decides to execute instruction in parallel (speculative execution, branch prediction, ...). For example, Very Long Instruction Word (VLIW) processors exploit ILP by using one long instruction to explicitly execute several operations in parallel.

Finally, Thread or Task-Level Parallelism (TLP) refers to executing many instructions at the same time on the same or different set of data. General purpose GPU architectures exploit TLP and DLP through multiple stream processing unit integrating multiple SIMD engines. Several GPUs are implemented with this Single Instruction Multiple Threads (SIMT) execution model, where SIMD is combined with multi-threading. With this distributed vector computing approach, GPUs optimize data throughput at the cost of data latency, suitable for applications that tolerate high latencies, such as deep learning training applications.

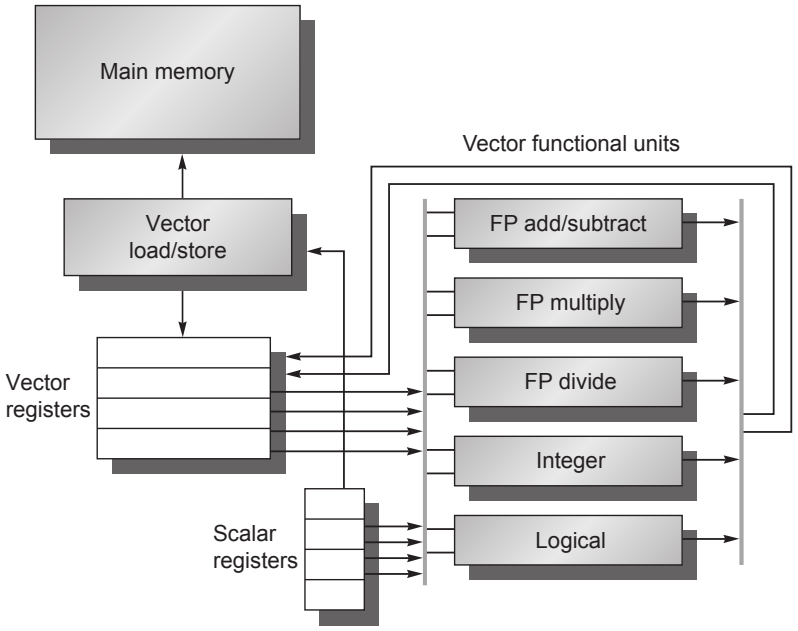


Figure 2.1: The basic structure of a vector architecture. A set of crossbar switches (*vertical thick gray lines*) connects these ports to the inputs and outputs of the vector functional units. Reprinted from "*Computer Architecture: A Quantitative Approach*" (p. 284), by J. Hennessy, D. Patterson, 2017 [2].

Vector architectures, GPUs and multimedia instruction sets exploit DLP by applying a single instruction to large dataset in parallel. As shown in Figure 2.1, vector architecture capture sets of data elements scattered in the main memory with its load/store unit, place them into large vector registers, operate on data with vector functional units, and then write back to their registers or scatter the results back into memory. A single instruction operates on data vectors, resulting in multiple register-to-register operations on independent data elements. The load/store unit is a key element of the vector architecture to fill data vector and scalar registers in order to hide memory latency and to leverage memory bandwidth. Current research topic are addressed at the compilation level to optimize access to the adequate data-structure by considering the application memory-access pattern and the hardware memory hierarchy [25].

In conventional architectures, vector programs strive to keep the memory busy but cache mechanisms stall the load/store unit when data is missing. GPU architectures achieve a high TLP and high throughput but exhibit high latency due to shared memory resources. Such as stream or systolic architectures provide high throughput but low execution latency, by transferring data between neighbours. Besides, distributed vector architectures provide new programming models to reduce latency impact between SIMD vector units by integrating specific vector memory instructions [26]. However, these architectures under exploits data structures with irregular patterns, such as strided-vector or stencil, due to the continuous data accesses of vector units [27], and the communication network between vector units does not scale according to the number of units [28, 29]. Vector architectures provide an efficient memory throughput, but remain limited by the latency of their communication network between neighbors and struggle to access specific patterns due to their vector register semantics, which are not suitable for irregular patterns. This lack of flexibility in communication networks differentiates vector architectures from reconfigurable architectures, whose interconnect resolves this lack of communication between neighbors.

2.1.2 Reconfigurable Architectures

Reconfigurable architectures are bridging the gap in performance and flexibility between ASICs and conventional processor-based architectures (CPU and GPU). Early FPGA architectures introduced a high degree of architectural flexibility through a fine-grained hardware reconfigurability [30]. A generic FPGA design is mainly composed of Configurable Logic Block (CLB), block of memory (BRAM), DSP block for highly specific computation and Input/Output (IO) blocks (IOB), as presented in Figure 2.2. The architectural behavior is re-programmable after manufacturing using a Hardware Description Language (HDL), thus configuring the interconnect hierarchy and the spatial layout of the components by configuring the Switch Box (SB) and Connection Box (CB). Therefore, the application is executed "physically", whereas conventional architectures have fixed hardware functions.

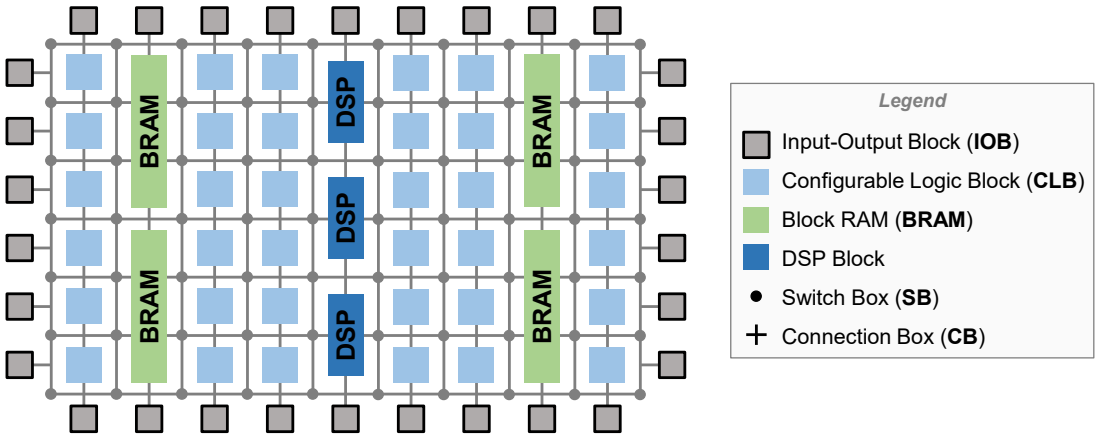


Figure 2.2: FPGA architecture overview.

Coarse-Grained Reconfigurable Architectures (CGRAs) were invented to solve FPGA limitations by trading some of their flexibility [31]. They increased the level of granularity, for larger and more complex units, including Configurable Function Block (CFB) and word-level interconnect reconfigurability. To improve the CGRA performance, the interconnect topologies are flexible but pre-defined [32] according to: (1) mesh-based architecture, such as 2D array with horizontal and vertical connections (e.g. Network-on-Chip (NoC)), (2) architecture based on linear arrays, using reconfigurable pipelined data paths to connect to the nearest neighbour, and (3) crossbar-based architecture, using a reduced switch network at word-level. There is a strong revival for CGRAs, especially for NN applications that benefit from their regularity properties and the reconfiguration of connections according to NN topologies.

CGRAs increase the performance and energy efficiency of data-intensive applications through dynamic reconfiguration of data paths and better management of memory resources [33]. The on-chip memory hierarchy is directly controlled by the application, providing more memory scalability and flexibility compared to the traditional cache memory. However, these architectures lack a general-purpose programming model to effectively exploit inter TLP instead of GPU architectures. Modern CGRAs address this problem by using simplified programming models to target specific applications [34], coarse-grained dataflow [35], or use a Domain-Specific Language (DSL) to capture high-level parallel patterns (*Map*, *Reduce*, *FlatMap*, *Fold*, ...), data locality and parallelism across applications at software level [36]. For example, Plasticine [37] achieves performance gains of up to 95× compared to an FPGA architecture, by dedicating coarse-grain logic functions (MAC, DSP, FPU and DRAM controllers).

To conclude, CGRAs are among the promising architectures to address the challenges of NN applications. However, they cannot easily manage vectors of configurable sizes and are specific for pre-determined vector sizes. CGRAs allow solving the computing configuration of such applications, but do not solve the memory wall, either on-chip nor off-chip.

2.2 Emerging Memory-based Computing Technologies

To break the memory wall, one architectural solution consists in bringing the processing units as close as possible to the memory to reduce data movement between memory and computing, also known as data-centric architectures [38]. In 1969, early work [39], propose a "cellular Logic-in-Memory" as a special programmable array to perform logic functions, suitable for associative memories (e.g. CAM), to speed up database searches. Other architectural work [40], proposed to integrate this kind of computing memory as a "logic-enhanced cache memory array" between the processor and the main memory, to perform logical and arithmetic operations between two data sectors. Since, many approaches to computation immersed in memory have been proposed using different memory technologies, such as Non-Volatile Memory (NVM), and Dynamic Random Access Memory (DRAM) and Static Random Access Memory (SRAM), both volatile memories.

2.2.1 Classification of Emerging Memories

In this section, emerging memories are studied under two aspects: at the technological level, due to the advances on emerging Non-Volatile Memories (eNVMs), and at the architectural level, by enabling memory computing architecture paradigm based on eNVM but also SRAM using current technologies. Emerging NVM focus on two kinds of applications: to increase memory density versus SRAM to solve the on-chip memory wall by changing the memory hierarchy paradigm, or to bring new computing paradigm with analog computing. According to the different memory properties, it is appropriate to classify these memories in relation to the computation and integration potentials of each technology.

NVMs are among the high-density, low-power consumption memories used for long-term Storage Class Memory (SCM), such as Flash memory (NAND, NOR), SSD or HDD. Thanks to many technological breakthroughs, emerging Non-Volatile Memory (eNVM) provide an adequate structure to enable computations within memory in various technology, such as the Spin-Transfer Torque Random-Access Memory (STT-RAM), the Resistive Random-Access Memory (RRAM), the Phase-Change Memory (PCM) and the Ferroelectric Random-Access Memory (FRAM) [41]. Although the activation mechanisms vary, all of these memories operate on the same principle: data are stored in a bit-cell array (crossbar), composed of a physically or magnetically alterable material, accessible vertically via Bit-Lines (BLs) and horizontally via Word-Lines (WLs). The bit-cell information is encoded with a specific current, voltage and duration pattern in order to write or read the data value. Values can be analog, binary (0 or 1) or discrete with Multi-Level Cell (MLC). There are two methods to perform computation within the bit-cell array between two memory words composed of horizontal lines of bit-cells. The first is based on Kirchoff's law to perform analog dot-product between words by summing the currents [42] and the second is to modify the memory array periphery according to the bitwise logical operation to execute [43]. Other works have demonstrated the interest of these emerging memories for image processing [44] or NN [45] applications, mostly by reducing the

Table 2.1: Characteristics of NVMs according to state-of-the-art studies [41].

	SRAM	DRAM	NAND Flash	STT-RAM	ReRAM	PCM	FeRAM	HDD
Cell size (F^2)	120-200	60-100	4-6	6-50	4-10	4-12	6-40	N/A
Write Endurance	10^{16}	$>10^{15}$	10^4-10^5	10^{12}-10^{15}	10^8-10^{11}	10^8-10^9	10^{14}-10^{15}	$>10^{15}$
Read Latency	∼0.2-2 ns	∼10 ns	15-35 µs	2-35 µs	∼10 ns	20-60 ns	20-80 ns	3-5 ms
Write Latency	∼0.2-2 ns	∼10 ns	200-500 µs	3-50 ns	∼50 ns	20-150 ns	50-75 ns	3-5 ms
Leakage Power	High	Medium	Low	Low	Low	Low	Low	N/A
Dynamic Energy (R/W)	Low	Medium	Low	Low/High	Low/High	Medium/High	Low/High	N/A
Technology Maturity	Mature	Mature	Mature	Test chips	Test chips	Test chips	Manufactured	Mature

memory traffic between memory and computation. Today, the number of operations available with these memories is limited, without the need for additional peripheral hardware. Although these memories have significant energy and density advantages, the memory access time and endurance are not sufficient to replace other memories, as presented in Table 2.1. With actual characteristics, these memories cannot be integrated close to the processor because of the performance gap in memory access time.

SRAM volatile memories, are appropriate in terms of performance to align with the processor speed, as shown in Table 2.1. Recent works [10] have proved with minor changes of the SRAM design, computing within the memory is possible to relax memory traffic and energy consumption to increase overall architecture performance. By changing the structure of the bit-cell array, where data are stored, pre-computing abilities between memory words is possible to perform logical and arithmetical operations when the system accesses a data [46–49]. This concept in commonly called In-Memory Computing (IMC). Additional post-computations are performed inside the memory periphery (e.g. arithmetic carry propagation) and then saved without going through the processor. Those works show that IMC is commonly used as a bit-wise vector-based accelerator and is a good pretender to mimic vector processing (e.g. SIMD). Among all these technologies, trade-offs exist between the area and energy cost of the modified memory and the execution speed up according to the application.

To summarize, these emerging memory-based computing solutions could be classified from the computation locality in the memory and the instruction complexity point of view, as shown in Figure 2.3. As a first level of integration, with the highest possible coupling, computations benefit from large vector width, as long as the operands are locally accessible inside the memory array. This first level is called IMC, or also compute before the memory sense amplifiers. More complex computation can be achieved at large vector width by integrating additional logic in the memory periphery after the sense amplifiers, which are often more power-hungry. Usually this principle is called Near-Memory Computing (NMC) in the literature. Any hardware accelerator (HW), such as CGRA or FPGA, bring specific-function to allow distributed parallelism, but suffer from reduced interconnect communication between the memory tiles and the processing elements. Finally, the CPU can perform any other complex operation by scattering data from the whole memory hierarchy (caches or dedicated Scratch-Pad Memory (SPM)). Two problems arise from this classification, which should be addressed to evaluate architec-

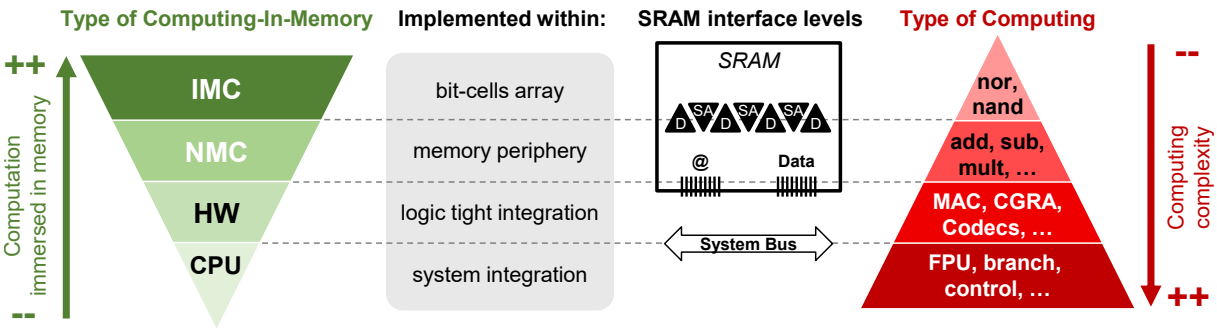


Figure 2.3: On-chip data-centric memory-based computing solutions classification. *Legend: IMC (In-Memory Computing), NMC (Near-Memory Computing), HW (Hardware accelerator integrating SRAM).*

tures using emerging memories: (1) hardware modifications and improvements are required to bring computation closer to the memory, increasing the current system energy consumption, (2) thus, an analysis of the type of operations to support across the memory hierarchy is necessary to propose trade-offs between Power, Performance and Area (PPA) design and the application [50]. After this presentation of the different levels of coupling between the computing part and the memory part, the IMC and NMC are presented in the following sections in more details.

2.2.2 In-Memory-Computing

In this section, we present in more details digital type of IMC based on SRAM, using existing technologies. Today, standard design tools allow to integrate the modified SRAM into a conventional architecture [46–49], in order to estimate the IMC performance gains with data-intensive applications. Due to the initial lack of an abstract programming model, these first approaches provide a custom IMC instruction set, to support basic logic operators (XOR, AND, OR, NOT). IMC architecture [47] has achieved performance gains of 1.9× and energy gains of 2.4× compared to a cache memory hierarchy architecture on a data-intensive kernel applications, for only 8% area overhead. Another work [49] reports a reduction of 75% of the number of memory accesses between the memory and the processor on cryptographic applications, such as Advanced Encryption Standard (AES), by executing IMC instructions over multiple cycles (*Read-Compute-Store*).

The Figure 2.4 presents how a logic IMC operation is performed inside the modified SRAM between two memory operands (A and B). First, the control logic unit give the Word-Lines (WLs) operand addresses to the row decoder unit, then additional precharge read Bit-Lines (BLs) drive the voltage according to the operand values up to the selected operation in the column decoder unit (OR, XOR, AND). Finally, the result can be stored in the memory array, often requiring an additional clock cycle. Design changes involve two additional pre-charge read BLs, the row decoder unit, for multiple WL selection, the control logic unit, including a Finite State Machine (FSM) to select the operand addresses, and the column decoder unit to select the logic operation. The pre-computation is performed inside the bit-cell array and the result is post-computed in the column decoder unit.

The IMC approach is close to the CAM memory technology, already used and proven to speed up the label search in databases [21], by selecting several WL (or all) in a modified bit-cell array. The standard memory interface (Address, Data) is maintained, thus data movements

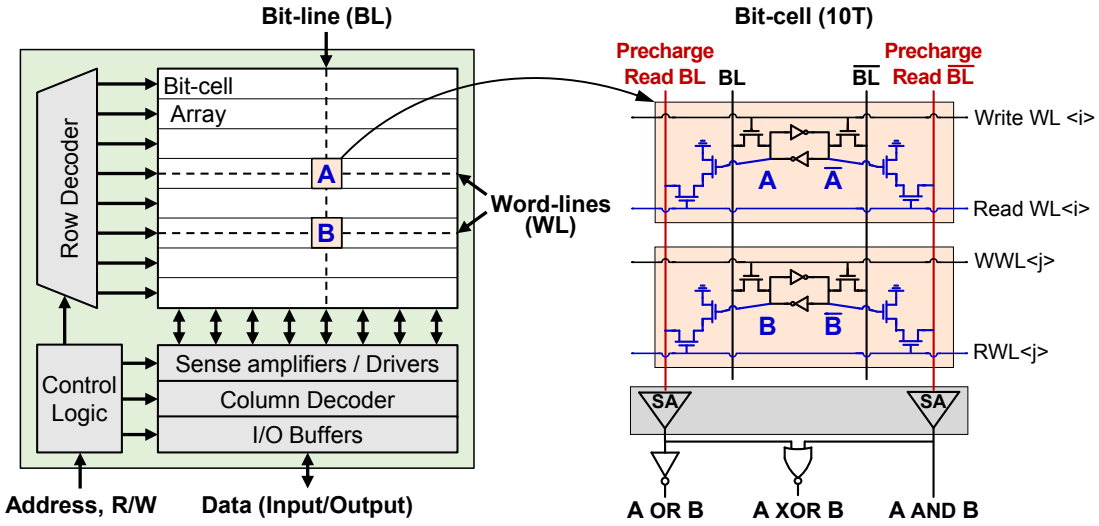


Figure 2.4: Modified SRAM architecture to enable In-Memory Computing (IMC) [52].

with the processor are reduced to only the control part, the computing being done directly within the memory array. Nevertheless, the memory operands must be located in the same bit-cell array, also known as memory bank, and operands must be aligned to perform vector processing. As the scope of type of operation is small, other works [46, 51] propose to implement an additional logic within the column decoder unit to extend the IMC instruction set in order to support arithmetic instructions (ADD, SUB). However, these operations have higher integration complexity (carry propagation, bit sign, ...) which impacts the energy consumption and the memory tile surface. Other issues occurs when an application is scaled up and the amount of memory is increased. In order to design larger memories, it is necessary to use multiple memory banks, or multiple tiles, making IMC between tiles impossible.

2.2.3 Near-Memory-Computing

IMC architectures will never perform all type of computations within the memory array, because of its memory structure, and control data movements between multiple memory tiles. The Near-Memory Computing (NMC) approach solves both of these problems by adding an Arithmetic Logic Unit (ALU) and interconnect features in the memory periphery for data manipulation between the bit-cell array and the outside interface. Contrary to IMC, only the periphery of the SRAM tile is modified, the bit-cell array is conserved, thus limiting the impact of the modified memory area overhead. Thus, NMC address a larger scope of data-intensive applications, such as database searching [47] and neural networks, vision and graph processing [53]. Combining IMC and NMC specific operations for security and cryptographic applications [52, 54] achieve gains up to 6.8× in performances and 12.8× in energy saving versus state-of-the-art cryptographic benchmarks running on hardware-accelerated implementations.

A specific IMC/NMC type, using the bit-serial scheme to perform a set of advanced instruction (arithmetic and floating-point operations) by computing the operation bit by bit at each clock ticking. Similarly to the IMC, the structure of the bit-cell array is modified, to shift horizontally the data to the ALU at the periphery. The bit-serial architecture can improve NN inference latency by 18.3× over state-of-art multi-core CPU and 7.7× over server class GPU [55].

Nevertheless, this scheme requires a high DLP (all words active) to achieve high-performance and the overall execution throughput depends on the type of operation. For example, it achieves an energy efficiency of 0.56 TOPS/W for 8-bit multiplication and 5.27 TOPS/W for 8-bit addition [51, 56].

Further system and architecture explore local data movements by integrating the NMC approach within a standard cache architecture [47, 57, 58]. The improved cache architecture achieves gains of 6× and 3× in performance with image processing and NN applications, respectively, compared an identical embedded processor including a SIMD co-processor [57]. However, the programming model is not explicit, partly because the cache involves hardware mechanisms and the interconnect between memory tiles is limited to a width of 512 bits. For larger memory, integration of multiple tiles of IMC or NMC has not been explored in a systematic way in terms of memory organization, local interconnect and performance trade-offs.

To conclude, the IMC and NMC approaches enable complementary design and architecture improvements to reduce memory traffic on the conventional CPU system bus. Although these architectures lack a general-purpose programming model, they offer vector processing opportunities, such as SIMD units, but may work on larger, scalable and configurable vector sizes by involving a reconfigurable memory interconnect. They provide a superior throughput by exploiting the DLP of data-intensive applications and should be compared to GPU architectures that use a comparable model. The combination of these two schemes will allow flexibility of the architecture according to application requirements. However, many software challenges remains, such as cache coherency and dynamic data remapping support [59]. In addition, the IMC/NMC scalability for data-intensive applications has not been studied except for cache replacement. Thus, the existing programming model does not consider memory transfers between NMC tiles, it would be relevant to propose an evaluation of the internal data movements through the memory tiling. Each additional functionality will necessarily impact the performance of the complete system and determine the trade-offs between energy consumption at the application level, allowing a more accurate architectural assessment.

2.3 3D Implementation Opportunities

Emerging 3D-stacked memory technologies are among the solutions to break the memory wall problem of high-performance computing systems [60]. These architectures provide high bandwidth, low latency, high memory density and capacity through different level of memory coupling within the computing architecture, depending on the level of 3D integration. Stacking memories on top of processing units increases interconnect connectivity and reduces data transmission delay through shorter wire length with less capacitance. Recent non-volatile 3D-stacked memory technologies improve memory capacity and cost of storage class memory (SCM), such as SSD. With these three-dimensional structures, new challenges of heterogeneous architectures aim to mitigate the memory bottleneck of off-chip memories.

2.3.1 3D Stacked Memories

Due to the growing memory demands of data-intensive applications, the main memory is pushed to its physical limits. It has become more difficult to increase memory density, reduce latency, and reduce power consumption of DRAM memory. 3D-stacked memory technologies, such as Hybrid Memory Cube (HMC) [61] and High Memory Bandwidth (HBM) [62], help

overcome these limitations by integrating larger DRAM memory and thus within the package, instead of having off-chip DRAM. They consist of multiple layers of DRAM memory stacked on top of each other and a logic layer at the bottom, in the same chip, as presented in Figure 2.5. These layers are connected vertically together with a wide Through-Silicon Via (TSV) bus, to provide high bandwidth and low latency data transfers. Moreover, the logic base provides a surface to implement hardware accelerators that interact with the DRAM memory via TSVs and the host CPU via a high-speed communication links. Manufacturers leave this slot available to allow new opportunities for architects to implement design, such as Processing-In-Memory (PIM) accelerators, as detailed in 2.3.2. A wide range of processing systems can be added in the logic layer, such as hardware accelerators, general-purpose cores, reconfigurable architectures, as long as the added logic complies with area, energy and thermal dissipation constraints.

2.3.2 Processing-In-Memory

Although 3D-stacked architectures provide higher capacity and bandwidth compared to off-chip memories, they are still limited compared to the maximum internal bandwidth available inside a DRAM chip [63]. Similarly to the IMC, the processing-using-memory approach can perform bulk bitwise operations (AND, OR, NOT) using the intrinsic properties and operational principles of the memory bit-cell array exploiting analog DRAM operations [64]. However, bulk bitwise operation cannot be performed on less than one row at a time and operand values could be modified according to the computation. Due to limits in the amount of instruction set that can be implemented inside the memory array, other works tend to implement processing-near-memory approach by adding hardware accelerator as close as to the DRAM chip [65]. In the rest of the thesis, the PIM refers to both processing-using-memory and processing-near-memory approaches [66].

The HMC architecture is developed jointly by *Samsung Electronics* and *Micron Technology* in 2011 [61]. As described in the previous section, the HMC architecture supports dedicated accelerators within the bottom logic layer, as shown in Figure 2.5. Recent works integrate PIM-based processing engines for neural network (NN) [53, 67, 68] and common scientific applications [69]. These processing engines access the stacked DRAM, containing the network weights, through a distribution interconnect (e.g. NoC, logarithmic interconnection, ...). 3D-stacked DRAM is separated vertically by 3D vaults, connected to the distribution interconnect by TSVs. Thus, all processing engines can address convolution computations on the whole memory in parallel. The integration of the state-of-the-art CNN achieves a performance of 240 GFLOPS (Giga Floating-point Operations Per Second) which is 3.5× better than a GPU architecture [70]. However, since the HMC architecture has limited space in the logic base, trade-offs exist between different NN topologies, the number of processing engines, and energy efficiency, as discussed in [53]. All these studies have show the interest of having local computing within and below the DRAM memory cube for large datasets, such as NN.

Although, HMC and HBM technologies are becoming a manufacturing standard with 3D TSV integration, many software and hardware challenges remains, such as a general-purpose programming model, virtual memory support and thermal solutions to enhanced computing functionality [59]. Also, due to the physical behavior of DRAM using a single transistor and capacitor (1T1C), the refresh rate of the memory slow down the performance of the overall architecture. Furthermore, the processing-using-memory is rarely exploited in 3D technologies

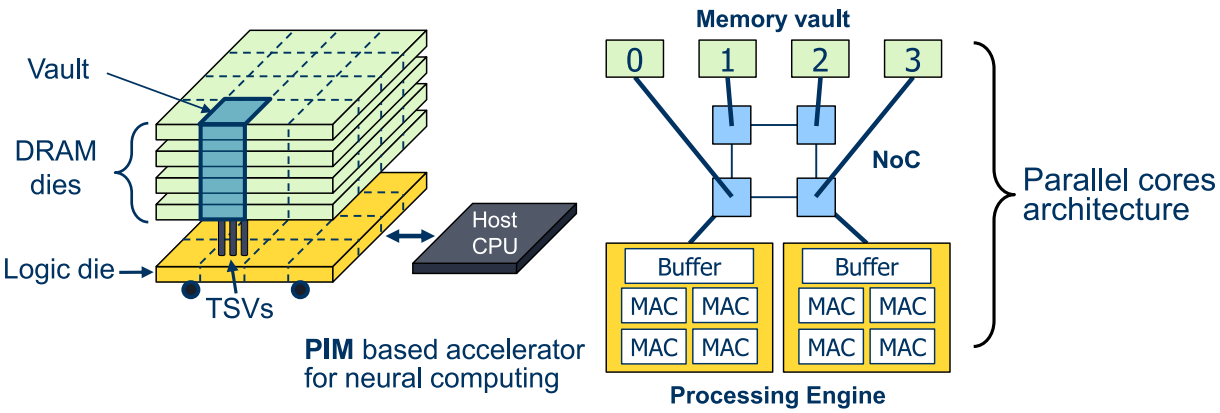


Figure 2.5: Hybrid Memory Cube (HMC) adapted for Processing-In-Memory (PIM) [72].

due to design modification impacts and the lack of programming models, limiting the maximum performance that such architectures could offer. Today's promising PIM architectures with industrial maturity focus on the 2D version of the proposed near-processing approach, as proposed by the *UPMEM* company [71].

2.3.3 Coarse Grain to Fine Grain

3D-stacked DRAM architectures provide high memory density available on a single chip and reduce production costs by replicating memory masks. However, DRAM requires periodic refreshes to maintain the stored data, which increases access latency and reduces performance of the architecture. Moreover, the strong connectivity between the memory and computing layers is a key element to improve the performance of data-intensive applications. Today, the size of TSVs limits the connectivity of 3D architectures, due to the standard cell congestion and wire lengths [73]. To solve these problems, heterogeneous 3D architectures propose to jointly combine several types of memory (DRAM, SRAM or NVM) using Monolithic 3D (M3D) integration [74]. It provides smaller vias and denser vertical integration, called Monolithic Inter-tier Vias (MIVs) (around 50 nm), compared to the TSV (1-5 µm), allowing to increase the number of connections between the different layers. Designing the same circuit using M3D integration reduces circuit area by 50% and power consumption by 22.3%, thanks to better wire connectivity [75].

For very tight 3D integration, instead of using TSV with pitches in the order of 50 µm pitch, two other kind of 3D technologies are possible. The first one, which is already mature, integrating multi-layer with aggressive pitch (1 µm range) using hybrid bonding technology, as used currently for imagers [76]. A second technology, called M3D, builds transistors onto multiple tiers, and establishing vertical cross connections between the tiers with nano-scale MIVs, as presented in Figure 2.6. As an example of such ultimate technology integration, the heterogeneous 3D-stacked *N3XT* architecture [77] integrates several technology breakthroughs to enable the best energy-efficiency systems for data-intensive applications. It is mainly composed of: (1) high-performance and energy-efficient Field-Effect Transistors (FETs), (2) high-density NVM to avoid off-chip memory wall, (3) fine-grained integration M3D for high connectivity between memory and logic tiers, (4) thermal dissipation solutions and (5) computation immersed in memory to exploit large data parallelism (e.g. IMC). Combining all these improvements, architecture achieves gains of 37× in performance and 23× in execution speed

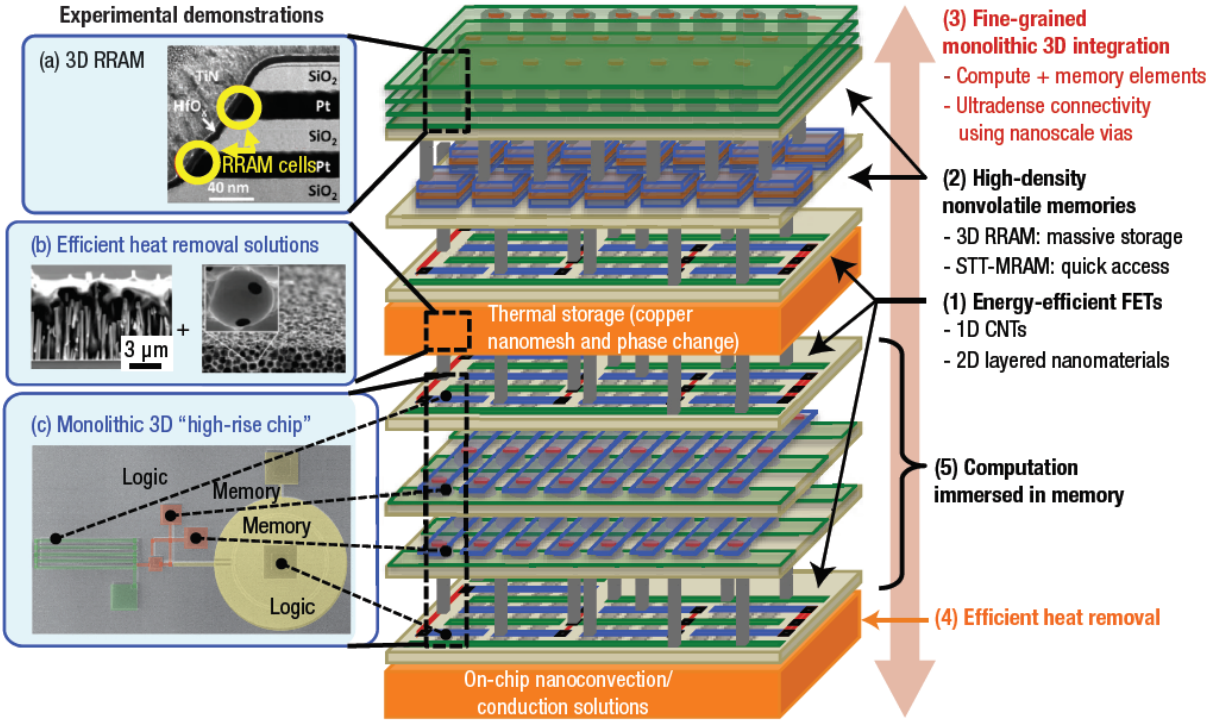


Figure 2.6: Monolithically integrated 3D system enabled by Nano-Engineered Computing Systems Technology (N3XT) [77]. On the right are the five key N3XT components. On the left are images of experimental technology demonstrations: (a) transmission electron microscopy (TEM) of a 3D resistive RAM (RRAM) for massive storage, (b) scanning electron microscopy (SEM) of nanostructured materials for efficient heat removal (left: microscale capillary advection; right: copper nanomesh with phase-change thermal storage), and (c) SEM of a monolithic 3D chip for high-performance and energy-efficient computation. CNTs: carbon nanotubes, FETs: field-effect transistors, and STT-MRAM: spin-transfer torque magnetic RAM.

up compared to the baseline of *PageRank* benchmark. While stand-alone hardware accelerators improve energy efficiency of sub-parts of the application, a better data connectivity is required to ensure that accelerators with low memory capacity do not slow down the overall system. The *N3XT* architecture fine-grained access to many memory arrays helps to overcome these limitations while reducing the impact of the memory wall.

2.4 Conclusion

Modern data-intensive applications are limited due to the memory wall problem of conventional architectures. On the one hand, the traditional cache is no longer energy-efficient enough to satisfy the requirements of vector architectures, on the other hand, dedicated ASIC architectures lack architectural flexibility and generic programming models. As a result, neither of these architectures can achieve a satisfactory balance between the two criteria, which creates an urgent demand for new architecture and technologies. At the architectural level, CGRA architectures provide the needed flexibility and provide a wide reconfigurable design to bring the calculation closer to the memories. Different memory-based computing approaches (IMC, NMC and PIM) can reduce memory traffic of the system bus in order to reach high connectivity, flexibility and scalability between memory tiles by computing on larger vectors. At the technological level, 3D-stacked approaches can bridge the connectivity gap between mem-

ory and computing, furthering the memory and computing coupling as introduced by the IMC and NMC. Emerging NVM continue to increase memory density, endurance and non-volatility, for bringing larger memory capacity on-chip without costly off-chip memory transfers. 3D-staked architectures offer many research opportunities, such as programming portability of data-intensive applications, virtual memory management and thermal dissipation removal. Hence, the Chapter 3 summarizes and combines the architectural and technological opportunities to propose a long-term vision in order to reduce the memory wall impact in conventional architectures and identify the state of current and future challenges of the associated research topics.

Chapter 3

A Dream: a 3D Stacked Distributed Computing Architecture

Contents

Due to the growing memory needs of modern applications and the limitations of conventional computer architectures, data-centric architectures are becoming more and more necessary. In this chapter, I propose a target architectural and technological vision to reach and a review of the opportunities and research topics that remain. Therefore, I focus the problems exposed in this thesis and the different approaches studied in the state-of-the-art in order to reduce the memory wall problem. The contributions of my thesis are addressing part of the architectural and technological challenges of this dream architecture.

3.1 Architecture Vision Overview

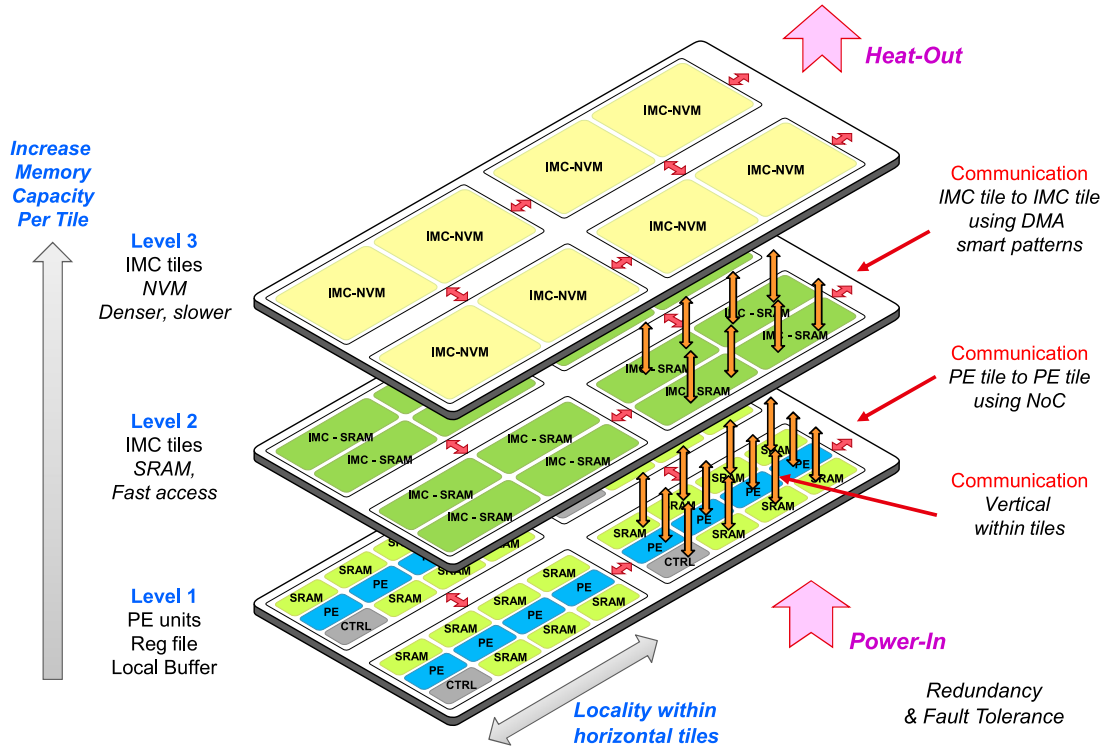


Figure 3.1: A 3D dream architecture to break the "memory wall".

In the perspective of collaborative research, I propose a long-term vision of architecture towards addressing a number of architectural and technological challenges to solve the problem of the memory wall in conventional computer architectures. Considering the state-of-the-art studied opportunities discussed in the previous chapter, this reconfigurable 3D-staked architecture enabling distributed computing is presented in the Figure 3.1.

This architecture combines on three layers the features needed to support data-intensive applications. Layer 1, contains multiple computing clusters tightly connected with a NoC interconnect to move data from neighbor to neighbor. These clusters integrate a local reconfigurable interconnect between Processing Elements (PE) associated with their local SRAM instruction memory and its dedicated hardware accelerators, such as FPU and SIMD vector computing units. Layer 2, contains SRAM memory clusters integrating the IMC (or NMC) approach to perform wide vector logical and arithmetic operations. These clusters use large buses within the layer to transfer data from one memory cluster to another through an on-chip Direct Memory Access (DMA) controller to move specific patterns at high bandwidth. IMC operations and vertical data movements between layer 1 and 2 are driven by each PE of layer 1. Finally, layer 3 contains the dense storage memory provided by emerging NVM technologies that support logical operations with ultra wide vector of data. Arithmetic operations are available in the lower layers involving additional vertical transfers.

At the software level, memory transfers and distributed computations executed within the memories are controlled at the PE level, with specific instructions according to the computation complexity and memory hierarchy. The programming model is based on Domain-Specific Language (DSL) allowing the developer to describe the vector operations interleaved with the

common processing instruction according to proposed architecture functionalities. The DSL needs also to support the high-level of parallelism of the proposed architecture, with both data handling and synchronization. Instead of GPU architectures or standalone hardware accelerators, operations on memory are executed in parallel to common instructions of the PE clusters. Data flow control and scheduling are supported through a compilation tool chain and dedicated libraries to exploit this energy-efficient architecture. At another software level, architectural simulators allows to evaluate the activity of computing flows in order to balance energy resources and heat dissipation in the architecture. The final perspective of this architecture is to be scalable at software and hardware levels in order to adapt its structure according to application memory requirements, to overcome the memory wall problems.

3.2 Architecture Challenges and Associated Research Topics

Today, the targeted 3D architecture is a proposal in which many technological barriers and difficulties remain to be solved. Indeed, the proposed technologies are not mature enough to be manufactured and as a first step, it is required to study and evaluate the architecture by using software and hardware simulators. Following the state-of-the-art and current research opportunities, different research challenges need to be addressed:

1. **3D-stacked architecture** – Thanks to 3D-stacked layers, different technologies could be implemented to provide a range of device characteristics with different trade-offs in terms of performance, leakage power, and reliability. Beyond the simple combination of memory and processor cores, various technology generations could be integrated, such as non-volatile memories, power regulators, and other technologies into the 3D stack. The stacking of two chips doubles the effective density of the device, but for heterogeneous architectures, the manufacturer still has to pay the cumulated total silicon area, so 3D does not allow strict economic scaling. 3D partitioning allows to choose the right technology for each computing layer, according to the target compute and power density. It would be most probably advanced Complementary Metal Oxide Semiconductors (CMOS) for the two first layers, while the third layer would be fabricated with a more mature technology and integrating the NVM. However, facing the scaling wall problem and following the technology advances, once 3D integration reaches a significant volume, tools and other design supports will develop rapidly, making more aggressive 3D architectures much less risky and more feasible.

2. **3D fine-grained integration** – Stacking multiple layers of chips together increases the number of devices per unit area and the connectivity between devices, but using 3D integration to maintain Moore's law for a few more generations is not without challenges. Early attempts at multi-layer stacking involved connecting several chips using Through-Silicon Via (TSV). However, the physical TSV size below 1 µm limited the possibility to have a smooth interaction between the different layers. The recent Monolithic 3D (M3D) technology enables integration of high-performance logic through dense fine-grained vertical interconnect between layers (or tiers) thanks to the Monolithic Inter-tier Via (MIV) (with via pitches as low as 100 nm). To benefit of this technology, it is required to solve fundamental challenges, such as the low-temperature process for the assembly of the layers (e.g. CEA's CoolCube technology proposal [78]) and the vertical chip alignment at transistor-level accuracy in order to improve the process yield of the complete chip, and new Electronic

Design Automation (EDA) tools to design 3D chip [79]. More mature 3D technology with less aggressive pitch (2-3 µm) using hybrid bonding are today available [80]. It is a research problem to define what is the required 3D interconnect density and corresponding technology pitch to achieve the fine-grain 3D connectivity of the proposed vertical interconnects.

3. **Efficient heat dissipation** – For the same technology node, using 3D integration doubles the number of transistors per unit area, and also doubles the amount of power consumed per unit area. At the physical level, the success of 3D architectures will depend on effective thermal management to reduce the impact of heat dissipation. Placing the power-hungry core layer closer to the heat sink could help alleviate the problem by benefiting from the thermal conductivity of the socket or the cold plate on the top of the chip. Architectural challenges and advanced technologies, such as micro-fluidic cooling solutions inserted between layers [81], integration of site-specific heat sinks within the stacked architecture design, cluster computing power balancing and scheduling, and frequency adaptive methods based on system resources can improve heat removal from the chip. Many opportunities exist to create new memory organizations, structures, and interfaces to better match the processor needs as well as power and thermal constraints.

4. **High-density eNVM** – As modern applications require more on-chip memory to avoid distant data transfers from external storage memories, eNVM (e.g. PCM, STT-RAM, RRAM) provide higher density and endurance than conventional NAND flash memory (e.g. SSD). The eNVM devices are pushing the limits of silicon-based Complementary Metal Oxide Semiconductors (CMOS) devices by promising highly scalable, cost-effective, low-voltage and low-power operation capabilities, high-speed switching, long retention, and the possibility of three-dimensional integration for high-density architectures. As of today, none of the existing eNVM technologies are able to address all of these challenges, some trade-offs must be achieved. Among these trade-offs, eNVM devices need to solve: reliability at cell-level and device-level, variability [82], increased memory density using multi-value memory bit cells [83], high-yield manufacturing process, and optimization of operation for different kind of applications. The eNVM provides opportunities to replace the standard CMOS memories to play the role of Storage Class Memory (SCM) in order to investigate novel architecture.

5. **On-chip in- and near-memory computing approaches** – The conventional computer architecture approach is to move data through the traditional memory hierarchy to perform the computations. To reduce memory traffic within this hierarchy, the IMC and NMC approaches are able to perform computations of medium complexity (logical and arithmetic) directly within the memory layers where the data is stored. However, three main challenges remain to be solved, such as (1) the vector programming support of IMC and NMC instructions and dynamic vector size configuration through the standard memory interface, (2) the scalability of the memory interconnect to distribute computations across the memory tiling, and (3) the reconfigurable inter-tile communication to enable computations between different tiles. Moreover, few comprehensive architecture proposals offer real solutions to the challenges mentioned above, as well as hardware integration that allows interleaving between memory accesses and IMC/NMC instructions.

6. **Reconfigurable architectures** – Compared to conventional computer architectures, so-called reconfigurable architectures are able to adapt their flexible hardware structure to optimize internal data movement and achieve high energy efficiency. Embedded or standalone FPGA, CGRA and many ASIC architectures comply with this definition. However, this domain-specific flexibility is also a key factor for the balance between energy efficiency and flexibility of these architectures. Such systems require further improvement and optimization to avoid bottlenecks in data movement, by addressing the following challenges: (1) improving the parallelism of distributed memory accesses through an efficient memory interface, (2) improving the flexibility of the memory interface with a programmable memory management unit, and (3) improving the bandwidth and latency between processing elements and memory using 3D-stacked chip technology. Even so, to address data-intensive applications, efficient hardware must be proposed as accelerator engines, but these PE engines must be able to cope with the wide variety of target applications, with still unknown kernels. A solution would be to propose reconfigurable architecture, such as the METEOR architecture, as presented in this PhD thesis. For the proposed architecture, we would like to benefit of efficient IMC/NMC with some reconfiguration features in terms of memory assignment, computation locality, with a partially fixed data paths.

7. **Distributed computing** – The purpose of distributed systems is to integrate stand-alone hardware and software systems into a larger, more comprehensive system. Since standalone systems have often been developed independently of each other, they are based on different interface, programming and operating system technologies. This raises new challenges for the integration of heterogeneous technologies. Thus memory and computing resources can be shared in a distributed system and there is a possibility that several processing elements will attempt to access to the same memory at the same time, also known as concurrency. Communication operations must be synchronized and scheduled in such a way that its data remains consistent by using standard techniques such as semaphores, provided by a high-level programming model commonly used in most operating systems. Moreover, the resource scalability of the architecture must be supported according to application requirements while maintaining performance and energy-efficiency despite of the number of resources available. Thus, the architecture must provide fine-grained 3D communication scheme to relax the synchronization constraints between the different layers, to preserve parallelism and distribution of the computation. Specific memory management units, such as DMA units, are required to perform data transfer, between tiles of the same layer, or between layers.

8. **General-purpose programming model** – Among all the challenges listed above, the development of a generic programming model is a key challenge to be solved, in particular to allow the portability of general-purpose applications to such distributed and dedicated architectures. The programming model define, at a high software level, the monitoring of system calls as well as the distribution of processes on hardware resources and, at a low level, the dedicated accelerator operations into the conventional instruction set of the processor. Over time, the execution model of stand-alone accelerators have diverged from the programming model, leading to a mismatch between software development expectations and system abilities. Decades of compiler research have shown this problem is extremely hard to solve. New opportunities such as data flow programming have allowed GPU architectures to emerge to satisfy the data-intensive application resources by offering better

throughput at the cost of increased latency. Finally, there are still many research prospects for the development of emerging programming models to interleaved IMC and NMC approaches into the processor instruction flow at fine-grained level in order to exploit data-level parallelism and reduce the overall execution latency for all types of applications. As a first step for IMC/NMC architectures, the implementation of a dedicated Instruction Set Architecture (ISA) and an interleaved programming model must be addressed.

3.3 Summary of the PhD Contributions

In order to tackle part of the challenges of the dream architecture proposed previously, I focused on the layer 2 of the overall system in my PhD: near-memory computing on a chip (challenge 5) within a reconfigurable and scalable memory architecture (challenge 6), allowing data transfers between memory tiles, and software integration (challenge 8) for the development of data-intensive applications.

I proposed an on-chip memory architecture composed of multiple smaller memory tiles, to contain the maximum amount of data for data-intensive applications, as presented in Chapter 4. This computing architecture is reconfigurable and scalable. Reconfigurability provides the opportunity to configure data paths in order to move data between memory tiles without overloading the processor system bus, and scalability allows the hardware interconnect's functionality and performance to match the memory requirements of the application. Similar to a cache memory, or a standard memory, this architecture complies the conventional memory interface for read and write accesses through the system bus or for tightly coupled interfaces with the processor. The specific design of the architecture provides large data movement between high-bandwidth memory tiles to reduce traffic between processor and memory. This architectural study of memory reconfigurability and scalability has been published in an international conference [84], several workshops [85, 86] and a patent submitted by the CEA [87]. Moreover, each memory tiles integrate the IMC and NMC approaches to perform large vector computations between tiles, supporting local data movements. An extended set of instructions is integrated into the processor in order to send instructions to the memory and also to configure the architecture topology to perform vector operations with configurable vector size given by the software developer. During my work, I propose a smooth software integration of this architecture with the objective of developing a future programming model that interleaved processor and memory instructions. This architecture can fit in the layer 2 of the overall system, it is considered as a slave memory but also as a co-processor unit from the processor point-of-view, as detailed in Chapter 5.

The study of this architecture involves other aspects, including hardware and software evaluations. At the hardware level, I explored the Power, Performance and Area (PPA) trade-offs of the wiring cost and the interconnect of this currently single layer architecture composed of small multiple memory tiles (multi-tiles), as presented in Chapter 6. Through a standard design flow, I established a methodology to choose the best memory topology according to application parameters (total size of the working memory and data bus width) and physical memory parameters (access time, energy, surface area). These observations led me to develop a wiring model to find the best performing multi-tile architecture using the founders' memory and considering all timing impacts of the interconnect. Indeed, there is always a topology optimized in performance and energy between a single large memory tile and multiple tiles

for a given memory size. The conclusions of this hardware study have been published in an international conference [88] and helped me to size the reconfigurable architecture according to physical constraints.

At the software level, I proposed an architectural exploration platform to evaluate the different memory topologies, as well as the software integration in a conventional compilation tool chain, as detailed in Chapter 7. Thanks to SystemC/TLM (Transaction-Level Modeling) abstraction, I could trace memory accesses, processor instructions and IMC/NMC instructions. In order to align performance between data accesses and computations inside the memory tiles, the architecture integrates a multi-cycle instruction pipeline. The modeling abstractions provided by SystemC/TLM enable fine-grained analysis of conflicts between data flows and memory instruction flows. Finally, the memory, interconnect and processor models are calibrated on some hardware extractions and estimations, the wiring cost model and a physical implementation of the RISC core, respectively. The impact of this architecture to accelerate kernels of data-intensive applications (presented in Chapter 1) is studied in Chapter 8.

In addition to my PhD contributions about this reconfigurable architecture and its feasibility study, I participated in collaborative work within the CEA team. I actively participated in the development of SystemC/TLM models for the behavior validation of the first single tile IMC integrated in a modified SRAM [89]. This design is implemented on a silicon chip to achieve gains up to 2× in energy reduction and 1.8× in speed-up compared to a 128-bit SIMD architecture. Moreover, the design space exploration of NMC tightly coupled with SRAM [90] contributed to refine the instruction set in order to identify trade-offs between surface area and energy. Based on these experiences around the NMC, towards the end of my PhD, I contributed to the discussion and elaboration of innovative stacked architectures coupling the NMC approach to NVM at SCM level [91]. This architectural exploration highlights the benefits of the NMC tightly coupled to the NVM, thus reducing memory write accesses and improving the endurance of the NVM. Finally, to move towards our architectural vision, I participated in the writing of a second patent involving a specific DMA unit to accelerate the data movement between intra-layers and inter-layers of a memory hierarchy without overloading up the processor system bus [92].

Chapter 4

A Reconfigurable Memory-based Computing Architecture Proposal

Contents

In this chapter, I propose a reconfigurable architecture to evaluate emerging technology gains for IMC and NMC. METEOR (Matrix of Elementary Tiles Enabling Optimal Reconfigurability) is a cluster architecture of computing tiles, reconfigurable in two manners: either a horizontal memory extension allowing larger vectors, or a vertical memory extension allowing more vectors. These computing tiles can use the IMC or/and NMC existing principle which consists in performing computation within the memory tile. I propose to extend these concepts in a scalable vectorization scheme allowing flexible parallelism directly in a vector format, detailed in Chapter 5.

4.1 METEOR: a Reconfigurable Memory-based Computing Cluster

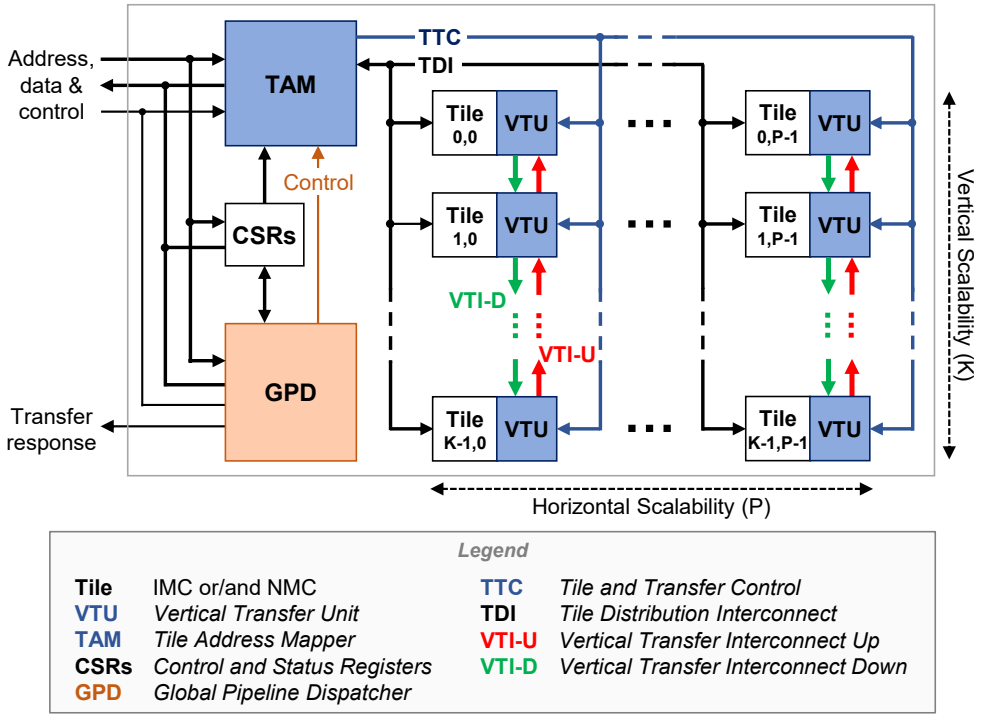


Figure 4.1: METEOR cluster architecture.

The METEOR cluster architecture consists of multiple tiles arranged in a physical grid of dimension K by P, as shown in Figure 4.1. Each tile is a memory circuit able to load data, store data and perform a number of logical and/or arithmetic operations using data stored in the memory as operands, thus referring to IMC and NMC approaches as defined in Chapter 2. In order to address kernels with larger dataset and real applications, it is necessary to enlarge the architecture with more memory tiles, while proposing the adequate programming model allowing vector acceleration and a communication scheme to perform computation between different tiles. Like any array of multiple memory architectures, the global *Tile Distribution Interconnect* (TDI) provides a 32-bit (or 64-bit) address and data interconnect to access to a 32-bit (or 64-bit) data requested by an external processing element (e.g. CPU), as presented in Section 4.1.2. To move large data vectors between tiles, these tiles are connected to a *Vertical Transfer Unit* (VTU) that is interfaced an additional vertical interconnect (VTIU and VTID) data paths. These transfers are globally managed by the *Tile Address Mapper* (TAM) with the *Tile and Transfer Control* (TTC) signals.

For application mapping facility, the physical grid can be logically reconfigured: (1) for horizontal vector extension to create larger vectors using the TAM and (2) for vertical memory extension to create more vectors using the vertical connections and the VTU. These reconfiguration mechanisms are detailed in Section 4.1.1. For overall control at cluster level, the *Global Pipeline Dispatcher* (GPD) resolves data hazards of incoming instructions to avoid future address conflicts and interleaved SRAM accesses coming from an external processing element (e.g. CPU). Thanks to this control, vertical data transfers can be coupled to the pipeline flow, all these mechanisms are explained in Section 4.1.3. Moreover, the GPD decodes dedicated METEOR instructions to modify *Control and Status Registers* (CSRs) in order to configure the tiling layout configuration deployed by the TAM.

To adapt the scalability between tiles, a single internal register is integrated in each tile to store intermediate results and avoid frequent read/write of the memory. In addition, coupled with the GPD, these registers decouple the timings to relax communication constraints onto the VTI, as presented in the Section 4.2.1.

4.1.1 Inter-tiles Reconfiguration and Communication

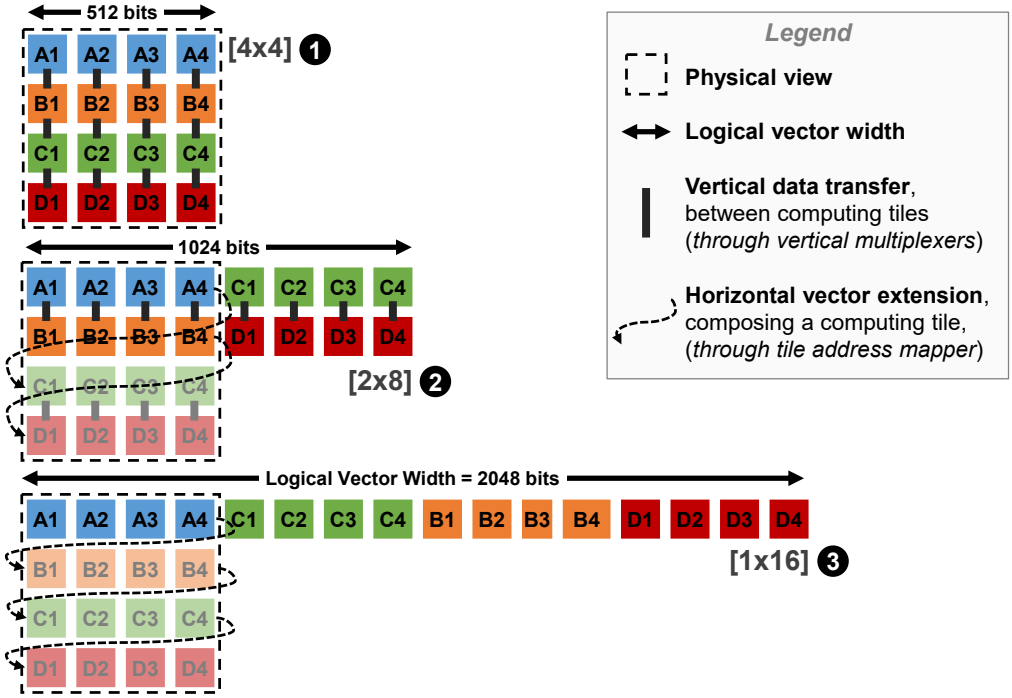


Figure 4.2: Physical and Logical views for 3 layout configurations of METEOR, with a logical vector size of: ❶ 512-bit, ❷ 1024-bit and ❸ 2048-bit wide.

Assembling these tiles horizontally permits to realize vectorized computation by distributing the same instruction and computation to a set of row. For data-centric kernel, such as matrix multiplication detailed in Chapter 5, data movement can be costly when computing on large vectors. The proposed architecture can dynamically resize the vector width during execution for each instruction to minimize this impact. For example, in Figure 4.2, the vector width is scalable from 512 bits up to 2048 bits with a 128-bit step, thus offering an address range that extends from 1024 vectors of 512 bits down to 256 vectors of 2048 bits. Thanks to this vector approach, computations are performed in an aligned and vertical way.

Horizontal vector extension – is a feature of the architecture to create scalable vectors using the TAM. These virtual vectors are composed of data stored in each physical memory tile, whose physical accesses are dynamically remapped by the TAM, as shown in the Figure 4.2. Therefore, in the configuration ❶, read and write data accesses by the CPU are physically aligned on the vector width of the architecture, in this case 64 bytes (512 bits). To broadcast the instruction, the TAM sends the same instruction to every physical tile composing the vector via the TDI. To save energy, the horizontal vector scalability is fine-tuned in runtime by activating only a subset of the tiles to work with a vector shorter than the maximum width.

Vertical memory extension – allows to access more vectors using the VTI, to use all the physical memory in all available configurations. For instance, in configuration ❷, the memory

is extended from A tiles to B tiles thanks to the VTU of each tile, vector operations between data A and B remain possible. The tiling configurations are always a power of 2 (1, 2, 4, 8, ...) which is proportional to the unit tile vector width. The tiling configuration parameters to set up the layout are detailed in the programmer's view in Section 5.1.2.

4.1.2 Overview of Vertical Transfers in the Pipeline Flow

To maintain the instruction throughput, the *Vertical Control Unit* of the GPD determines vertical data movement between tiles according to the instruction flow, then send global controls to the TAM in order to monitor the VTUs. For example, the implementation of a 5-stage pipeline is proposed, through the following stages of (1) DEC: decode, (2) RD1: read left operand, (3) RD2: read right operand, (4) EX: execute the operation and (5) WB: write back into the memory.

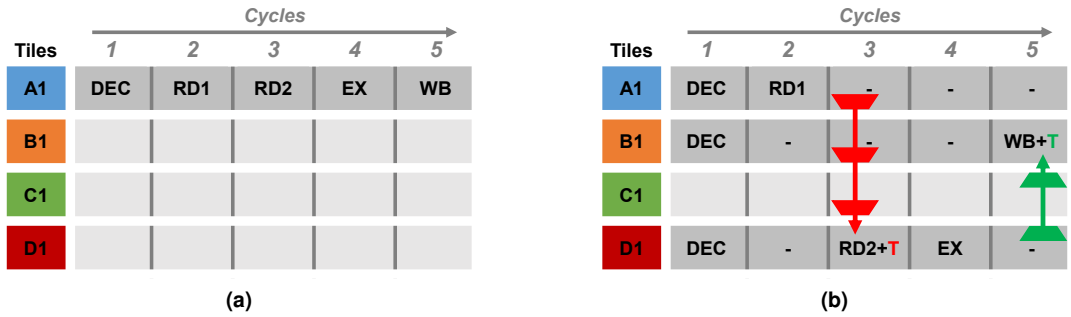


Figure 4.3: 5-stage pipeline flow views for 1 instruction in 4 stacked tiles (A1, B1, C1, D1). **(a)** data operands are located in 1 tile, **(b)** data operands are located in different tiles such as B1 = A1 <op> D1.

In Figure 4.3, we consider a tiling configuration of 4 vertically stacked tiles of vectors (A1, B1, C1 and D1), which received instructions, as the configuration layout ❶ in Figure 4.2. When all the data of the operation are located in the same memory tile, as in example 4.3(a), the instruction will be send only to this tile (A1). However, when operands are scattered across multiple tiles, as in example 4.3(b), vertical data transfers are required and the instruction will be broadcast among the tiles storing the data (A1, B1 and D1). On the first cycle, the instruction decoding (DEC) is performed in A1, B1 and D1 - on the second cycle, the first operand is read (RD1) in the tile A1 memory - on the third cycle, the first operand is transferred (T) to D1 and the second operand is read (RD2) in D1 memory - on the fourth cycle, the operation is executed (EX) between the operands - and on the fifth cycle the result is transferred (T) to B1 memory to be written (WB). The direction of the read data transfer is always from the tile that reads the left operand (RD1) to the tile that performs the operation (EX). Data transfer direction: data can be transferred to upper or lower tiles via the two vertical interconnects called *Vertical Transfer Interconnect Up* (VTIU) and *Vertical Transfer Interconnect Down* (VTID), as shown in Figure 4.1.

4.1.3 Interleaved Instructions and Memory Accesses

To ensure a constant instruction flow while maintaining timing performance, a data pipeline is embedded in each tile, as detailed in Section 4.2.1. A data pipeline is a sequence of stages where the output of one stage is the input of the next one. The pipeline controller must ensure the availability of data for each instruction passing through the pipeline. Data hazards occur when instructions modify the same data at different stages of the pipeline, listed in these three situations:

- Read-After-Write (RAW) conflict occurs when an instruction refers to a result that has not yet been calculated,
- Write-After-Read (WAR) conflict occurs when an instruction tries to write the destination before it is read,
- Write-After-Write (WAW) conflict occurs when an instruction tries to write an operand before it is written by a previous instruction.

To eliminate these hazards in METEOR, several solutions has been proposed:

- The **bubbling** (or **stalling**) is physically implemented in the GPD to slow down the instruction flow when conflict occurs between operands. This feature is a basic pipeline mechanism to avoid most of the data hazards at the cost of increased latency.
- The **operand forwarding** should be explicitly specify in the instruction by the compiler (or the programmer) to transfer operands through the pipeline stages using internal registers. This feature prevent from stalling mechanism and increase pipeline throughput thanks to software adjustments or compiler's optimizations.
- The **reordering** is not supported at hardware level because the GPD preserve the instruction order from the programmer's perspective. This feature could be done at software level, to achieve advanced optimizations of the instruction order in the execution flow.

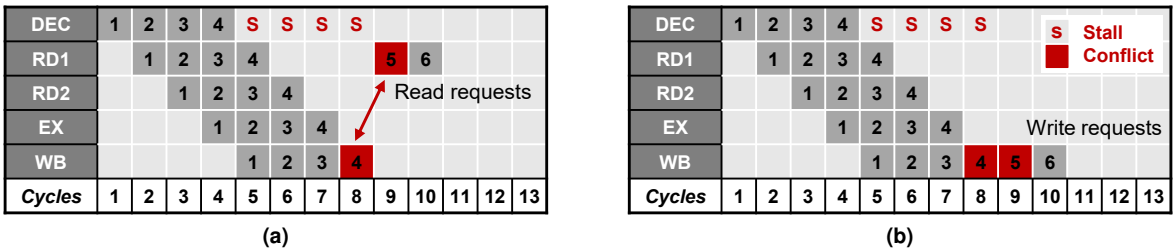


Figure 4.4: Global Pipeline Dispatcher view of a 5-stage pipeline with interleaved SRAM accesses. Legend: instructions are sorted by order of arrival in the pipeline, from 1 up to 6. **(a)** Read-After-Write (RAW) data hazards, **(b)** Write-After-Write (WAW) data hazards.

The stalling solution is supported in our architecture and the operand forwarding solution should be specified by the software developer. The *Data Hazard Unit* of the GPD enables the interleaving of standard memory requests without address or data conflicts. It avoids these conflicts by stalling requests when the vector address of the instruction is equal to the absolute address of the memory request, as presented in Figure 4.4. These figures show RAW and WAW data hazard conflicts occurring with previous instructions when read and write access to the memory is requested, respectively. The GPD stalls the Processing Element (PE) requests whenever there is a data conflict during the write-back instruction stage.

4.1.4 Vertical Transfers Detailed Implementation

The pipeline throughput can be enhanced in several ways. If a dual-port memory is physically integrated, the read and write pipeline stages can be performed in parallel, increasing the instruction throughput, as shown in [90]. With METEOR's multiple tiles, we can optimize throughput in two other ways, whatever the type of memory used: (1) by arranging the data placement, as in Figure 4.5(b) and (2) by using the internal tile register. The register optimizations are detailed in the Section 4.2.1.

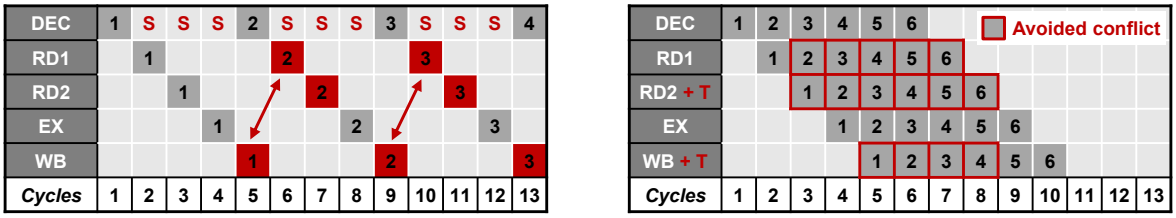


Figure 4.5: Global Pipeline Dispatcher view of a full 5-stage pipeline for 6 instructions. **(a)** data operands are located in one tile, **(b)** data operands are located in different tiles using vertical transfers.

As introduced in Section 4.1.2, vertical transfers move data across multiple tiles, so this principle is represented in a flow of multiple instructions in Figure 4.5. It represent a flow of six instructions passing through the GPD, for checking data dependencies, where data operands are located in the same tile in Figure 4.5(a) and where data operands are located in different tiles using vertical transfers as the previous figures 4.3(a) and 4.3(b), respectively. Thanks to the data operand scattering and the vertical communication scheme, a single memory tile avoids concurrent read and write accesses, thus eliminating data hazards and increasing the pipeline throughput.

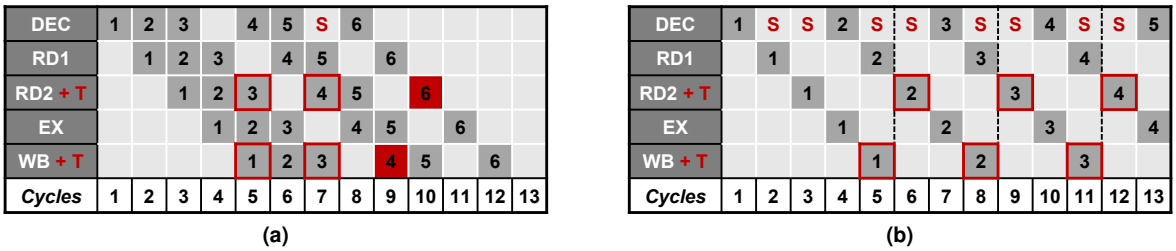


Figure 4.6: Vertical Transfer Interconnect hazards between instructions and reconfiguration. **(a)** Up-Up or Down-Down transfer hazards, **(b)** Tiling reconfiguration in the pipeline flow.

When executing a single instruction using vertical transfers, the read and write ports can be used simultaneously and transactions never conflict. However, when two instructions require vertical transfers on the same cycle, as shown in Figure 4.6(a), conflicts occur if the read and write requests must intersect in the same direction. For example, a *Down-Down* conflicts occurs when a read transfer is made from `A1` to `D1` and a writing transfer is made from `A1` to `B1`. In these cases, the *Vertical Control Unit* of the GPD ensures that vertical transfers do not conflict between instructions and can stall the PE requests if this happens. Similarly, this unit ensures that all vertical transfers are executed for each instruction before changing the global tiling configuration, as presented in Figure 4.6(b).

In conclusion, the TAM and the VTU provide an additional, scalable and reconfigurable communication scheme between tiles, and the GPD prevents IMC/NMC instruction operand hazards and standard memory access conflicts. These hardware mechanisms provide at the software level an interleaved programming scheme to increase the overall throughput of the architecture and alleviate the software development effort for vector processing. The impact of pipeline latency and forwarding mechanism are evaluated using a cycle-accurate simulation platform, as detailed the Section 8.1.2.

4.2 Design Specifications

In this section, each units of the overall METEOR architecture are detailed at the low-level to make possible its physical implementation at register-transfer level (RTL). The architecture is composed of four main units, detailed in the following sections, and local registers (CSRs) to store the current tile layout configurations in order to remap addresses in the TAM. This control interface is detailed in the Section 5.1.2.

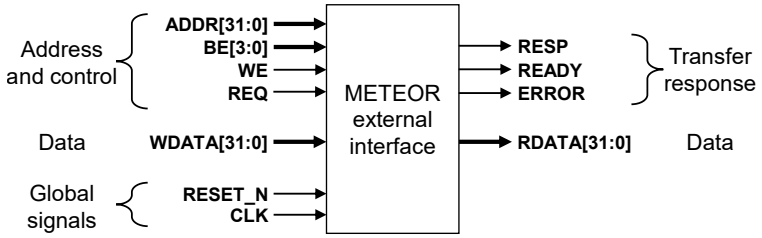


Figure 4.7: METEOR generic external interface (all signals are external to METEOR).

This architecture has a generic memory slave interface that can be connected on a processing element or a system bus, as detailed in Section 4.3. As shown in Figure 4.7, the master provides address (`ADDR`), data (`WDATA`) and control signals (`BE`, `WE`, `REQ`) for reading and writing operations. The slave provides the readout data (`RDATA`) and the status of the request (`RESP`), when the transfer is completed (`READY`) and when an error occurs (`ERROR`). Conventional data accesses and IMC/NMC instructions pass through this standard memory interface allowing smooth hardware and software integrations.

4.2.1 IMC/NMC Tile Unit

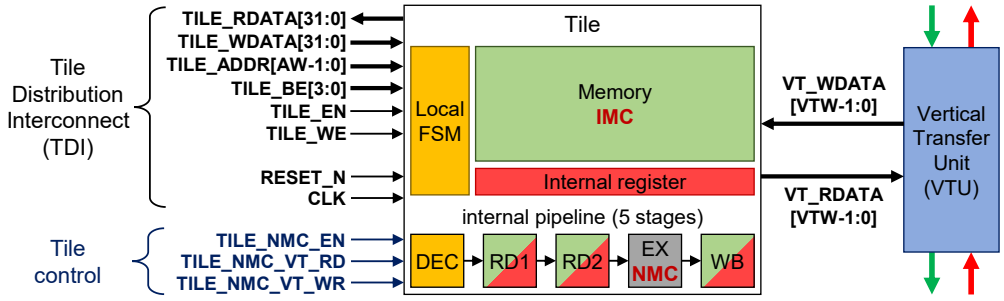


Figure 4.8: METEOR tile unit implementing IMC and NMC.

To evaluate the benefits of the IMC and NMC approaches described in Sections 2.2.2 and 2.2.3, the tile supports both logical (IMC) and arithmetic (NMC) operations to address data-intensive applications, such as database searching, image processing and neural networks. As shown in Figure 4.8, the proposed tile organization can support IMC and NMC approaches coupled with a 5-stage pipeline for timing decoupling and vertical transfer synchronization. The IMC part is able to perform operations between two operands, stored on two WLs, within the memory array in a single clock cycle. Since the IMC is limited to internal logical operations, the NMC provides an additional ALU integrated in the pipeline execute-stage, to perform arithmetic operations. Hence, the other element of the tile are: the local FSM, the internal register and the pipeline. The local FSM controls the pipeline flow and the data redirection of the read and write accesses to (1) the memory, (2) the internal register or (3) the VTU for inter-tile transfers.

The internal register is used to store intermediate results of operation (as an accumulation register), to implement the operand forwarding solutions by forwarding the partial result to the adequate pipeline read stage, and to reduce excessive memory accesses and associated power consumption. The internal register proposal is a contribution of my PhD thesis, to achieve efficient data transfers between tiles at the multi-tile level. Moreover, to integrate this tile in the METEOR grid and respect the global pipeline control, each tile has an 5-stage internal pipeline to sequence each instruction, as discussed in Section 4.1.2. For design optimization purposes, the pipeline data hazard control is implemented in the GPD at global level, saving circuit surface area. A early implementation of the NMC design has been proposed in [90] with a SRAM memory from *Global Foundries* 22 nm FD-SOI technology node, without including the internal register control.

To exploit the maximum vectorization scheme, the memory interface is split into: (1) the 32-bit standard memory interface (TDI), for regular memory accesses, (2) the internal communication interface passing through the VTU, for data movement between different tiles, and (3) the *Tile control* interface to synchronize pipeline flow at the global level. Thus, when an IMC/NMC instructions are transferred through the TDI, the local FSM redirects the instructions to the pipeline decode stage. The vector computations could be performed with internal memory vectors, register vectors or external vectors of a Vector Tile Width (`VTW`) larger than 32-bit words. Thus, the proposed vector programming scheme supports IMC and NMC instructions and vector movements through different memory tiles.

4.2.2 Vertical Transfer Unit

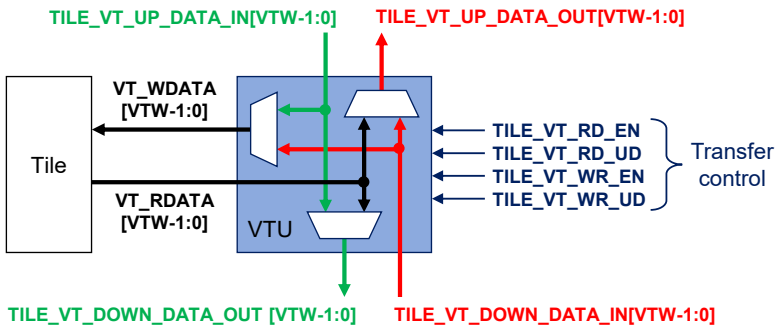


Figure 4.9: METEOR Vertical Transfer Unit and Tile interfacing.

The VTU provides three wide data interfaces (UP, DOWN, TILE) to perform vertical data transfers in one cycle between other tiles and control signals derived from the TTC interface as presented in Figure 4.9. All possible data movements driven by these control signals are listed in Table 4.1. The VTU interactions enable access to VTIU and VTID when the tile needs to process data from lower or upper tiles. By default, the VTU forwards the signals without interfacing with them, which involves passing the data through a cascade of multiplexers. In summary, the implementation of the METEOR architecture is physically constrained by: the standard read access time through the TDI and the vertical transfer time through the VTI using a series of VTU. This proposed communication scheme is partially fixed to achieve a convenient tiling scalability for large systems, but further RTL implementation and circuit measurement should be done.

Table 4.1: Vertical Transfer Unit control and interactions (Up: 1, Down: 0).

WR_EN	RD_EN	WR_UD	RD_UD	Description (related to the tile)
0	0	X	X	no tile interactions, vertical forwarding
0	1	X	0	Read data from the DOWN path
0	1	X	1	Read data from the UP path
1	0	0	X	Write data to the DOWN path
1	0	1	X	Write data to the UP path
1	1	0	0	Read data from the DOWN path, Write data to the DOWN path
1	1	0	1	Read data from the UP path, Write data to the DOWN path
1	1	1	0	Read data from the DOWN path, Write data to the UP path
1	1	1	1	Read data from the UP path, Write data to the UP path

4.2.3 Tile Address Mapper Unit

The TAM provides an external interface to perform the data transfers to or from a processing element and four internal interfaces as:

- TDI, a distribution network to access to the memory tile where the data is stored,
- GPD, control signals to monitor overall NMC features,
- TTC, a distribution network to dispatch NMC control and transfer signals to each tiles,
- CSR, a set of local registers used for the global tile layout configuration.

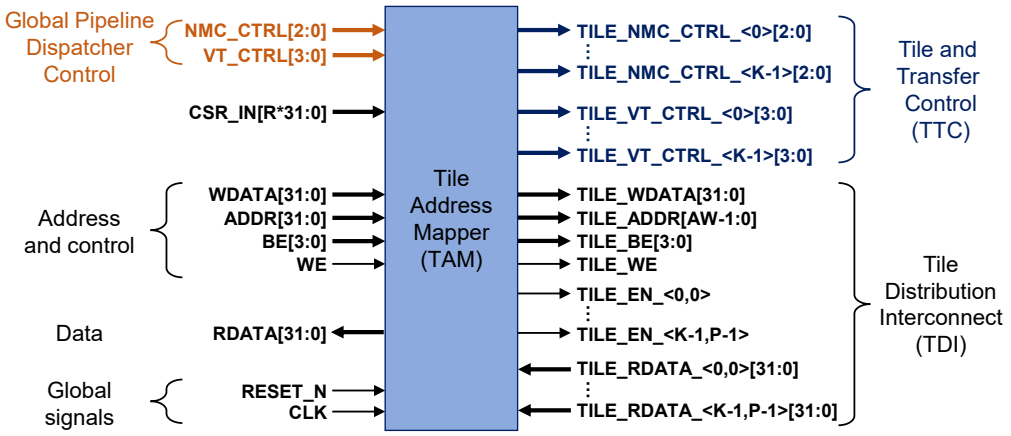


Figure 4.10: METEOR Tile Address Mapper unit.

The TAM operates in three modes: READ, WRITE and COMPUTE. The first READ mode involves standard data reads: a request is received by the *address and control* interface (`ADDR`, `BE`, `WE`), then the TAM creates an address relative to the tile memory size (`TILE_ADDR`) and it selects the tile containing the data (`TILE_EN`) via the TDI. In the next cycle, the 32-bit data is available and is returned by `TILE_RDATA` through a output read multiplexer leading to `RDATA`.

The second WRITE mode involves standard data writes: a request is received by the *address and control* interface (`WDATA`, `ADDR`, `BE`, `WE`), then the TAM selects the tile where to write the data, this operation takes also one clock cycle.

Finally, the third COMPUTE mode involves receiving an IMC/NMC instruction through the *address and control* interface. This interface is used to send IMC/NMC instructions to the memory tiles as explained in Section 5.2. The TDI distributes the instruction to the tiles and the GPD controls address conflicts. The TAM configures the TTC interface (`NMC_CTRL`, `VT_CTRL`)

to select the tiles where the operation is performed. With this mechanism, the tiles receive the operation and control signals in one cycle at the decode stage of the pipeline, then four more cycles are required to complete the computation. The instruction flow is synchronized by the TTC signals, avoiding future data conflicts, as presented in the previous sections.

By partitioning a large memory into smaller tiles, the TDI allows to scale the cluster with some limits: energy cost of individual accesses is inversely proportional to the read access time. In terms of physical design, the TDI provides an energy and performance trade-off as long as the number of tiles is limited, as presented in Chapter 6. In addition, this inter-tile scheme enables a large data movement and bandwidth between tiles to reduce the traffic on the system bus, its impact will be discussed in Chapter 8.

4.2.4 Global Pipeline Dispatcher Unit

The GPD provides an external interface to manage instruction transactions and to respond to the processing element (PE) in case of conflicts, and two internal interfaces to control the TAM and to modify CSRs, as presented in Figure 4.11. Its composed of three internal units:

- **Data Hazard Unit** checks data conflicts between instructions as explained in Section 4.1.3,
- **Vertical Control Unit** checks vertical transfers conflicts as explained in Section 4.1.4,
- **CSR Control Unit** manages the memory logical configuration explained in Section 5.1.2.

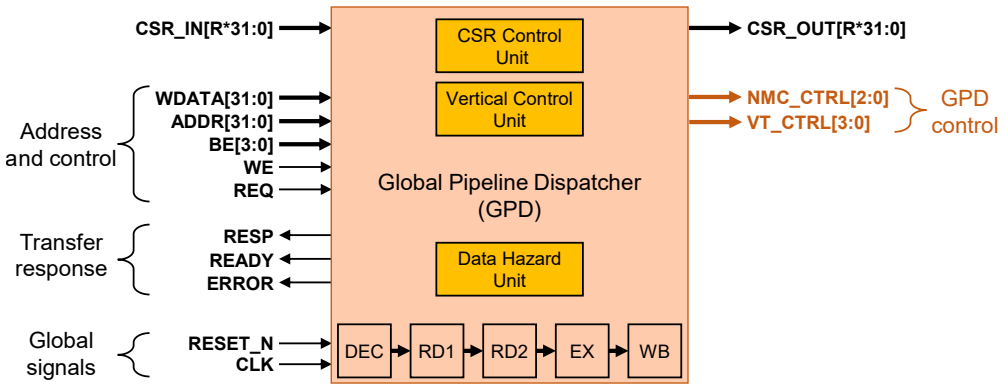


Figure 4.11: METEOR Global Pipeline Dispatcher unit.

The GPD stores the 5-stage instruction pipeline flow of the computing tiles in order to determine data movements (vertical and scalar) and instruction conflicts. Through the *transfer response* interface, the GPD is able to stall the instruction flow coming from the PE. Thanks to this global approach, the tiles execute all instructions synchronously with minimum local control, allowing the optimization of the multiple tiles at the global level.

In conclusion, these main elements of the METEOR architecture address the main challenges of the on-chip IMC and NMC approaches at the hardware level. Thus, the vector programming control of IMC and NMC instructions through a standard memory interface is supported by the GPD to avoid instruction and data conflicts. The internal communication scheme, using the VTUs, is scalable in order to build larger memories, but must be physically implemented for accurate scalability estimates. Finally, the TAM provide control signals to dynamically reconfigure the inter-tile communication scheme and the vector size for large vector processing, as presented in the Chapter 5.

4.3 System Integration Overview

The METEOR's generic interface is composed of: an address, data and control buses for sending an instruction, reading or writing data from or to the memory, as well as a transfer response in case of address or instruction conflicts. These signals are necessary and sufficient to integrate meteor into a standard scalar system processor as a slave memory. For example, it can be integrated as:

- A low-latency Tightly-Coupled Memory (TCM) working at the processor frequency, directly connected to the processor and the main system bus, as presented in Section 4.3.1,
- A loosely-coupled co-processing unit connected to an hardware accelerator and operating in parallel to the processor with its own frequency, as presented Section 4.3.2,
- A high-speed SPM connected to external sensors and an I/O system bus in order to preprocess raw sensor data before the CPU's post-processing, as presented in Section 4.3.2,
- A cache memory connected to a cache controller using a wide I/O interconnection, as presented in several state-of-the-art contributions [47, 55, 57].

4.3.1 Tightly Coupled Memory of a Processor

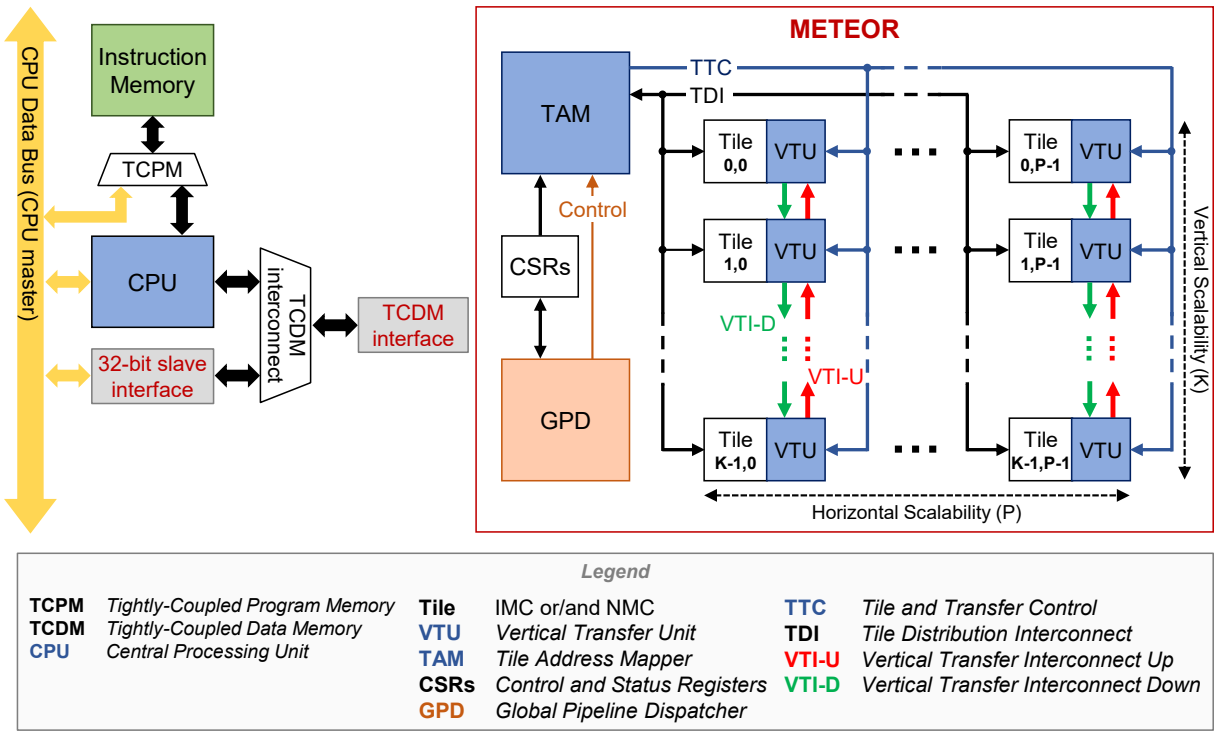


Figure 4.12: Standard processor architecture with METEOR on a TCM interface.

METEOR's external interface can be connected to a bus as a slave memory controlled by a master processing element (e.g. CPU). Through this generic slave interface, data access remains conventional for the master and specific NMC instructions are wrapped in address and data signals, as detailed in 5.2.1. The master provides address, data and control information to initiate read and write operations. The slave provides a response status of the transfer to the master, a signal when the transfer is finished and an error signal, as detailed in the standard memory interface protocol of *ARM* [93].

In the Figure 4.12, the CPU executes its instruction flow coming from the *Instruction Memory* tightly connected via the Tightly-Coupled Program Memory (TCPM) interconnect. METEOR has a Tightly-Coupled Data Memory (TCDM) interface with the CPU and a slave interface with the *CPU Data Bus*. Through the TCDM interface, the CPU can (1) send NMC instructions to METEOR, (2) read sequential data from METEOR and (3) write sequential data to METEOR. Through the slave interface, METEOR operates as a memory addressable by any other on-chip unit (CPU included) requesting data accesses.

4.3.2 Loosely Coupled Co-processing Unit

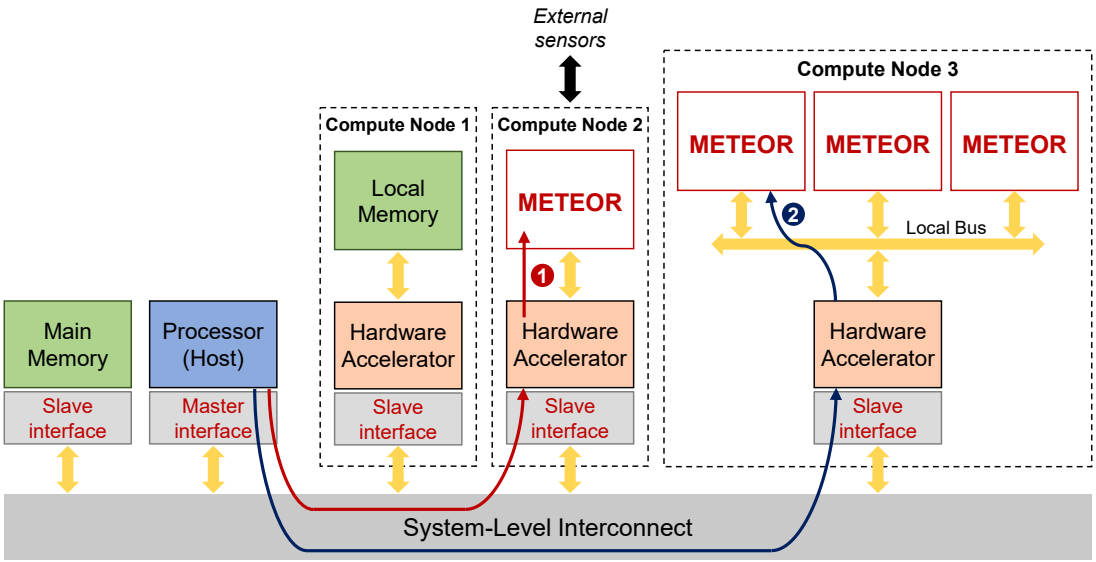


Figure 4.13: Standard processor architecture with METEOR as a co-processing unit.

In the Figure 4.13, the METEOR architecture is integrated as a co-processing unit in a standard system composed of a host processor, its main memory for storing program and data and several accelerators represented as compute nodes. These hardware accelerators are:

- **Compute Node 1**: a conventional hardware accelerator that can process specific tasks in parallel of the host processor (e.g. FPU co-processors, graphic co-processors, ...),
- **Compute Node 2**: a single METEOR unit for low-power near-sensor applications,
- **Compute Node 3**: multiple METEOR units for distributed vector computing.

The compute node 2, is similar to the tightly-coupled memory integration proposal, where the CPU is replaced by a hardware accelerator dedicated to stream the METEOR control flow. Through the system-level interconnect, the host processor send specific macro instructions to the hardware accelerator in order to launch data processing workloads inside METEOR. Connected to external sensors, METEOR allows pre-processing of the raw sensor data before the accelerator or the CPU post-processing. In the compute node 3, multiple unit of METEOR are connected on a local data bus driven by a hardware accelerator that increases the memory capacity (and the computing capacity). Thanks to this approach, specific kernels can be requested to the accelerator to execute repetitive tasks in order to reduce the instruction traffic on the system-level interconnect. In addition, these co-processing units can also be represented as a scratch-pad memory (SPM) from a system point of view (and host processor point of view). In this case, the host processor can directly access the METEOR data as a conventional data memory via path ❶ and ❷.

To conclude, the METEOR architecture can be integrated as a memory as well as a vector accelerator. In the following part of this thesis, the exploration of the tightly-coupled system (compute node 2) will allow to evaluate the gains of the IMC/NMC concepts integrated in conventional architectures. Thus, these systems integration proposals show the potential of the architecture to be integrated into a more comprehensive and distributed computing system, as proposed in Chapter 3. Integrating this architecture into a 3D-stacked architecture, whether it is a memory or a vector accelerator, would increase the connectivity of communication schemes between inter-tiles or intra-tiles, to increase performance and energy efficiency of data-intensive applications.

4.4 Conclusion

The proposed METEOR cluster is a scalable and reconfigurable tiles of IMC/NMC architecture, each tile enables arithmetic and logic operations within the memory. The architecture design provides the capacity to interleave standard memory accesses and compute between the different tiles with the IMC/NMC instruction flow, avoiding possible conflicts. Thanks to its generic memory interface, METEOR can be integrated as a tightly-coupled memory or as a loosely-coupled co-processing unit into any conventional architecture. The combination of a horizontal scalability scheme and a vertical data communication offers an adaptive vector size for maximum performance onto the IMC/NMC tiles. Stalling mechanism and operand forwarding solution are recommended to increase the pipeline throughput, thus reducing the overall memory power consumption. Finally, this architecture is evaluated at the software level in 5, and several performance and energy trade-offs between the tile partitioning and the standard memory interconnect are studied in Chapter 6.

Chapter 5

Software Integration for Scalable Vector Computing

Contents

Thanks to its generic communication interface, presented in the previous chapter, the METEOR architecture can be integrated into a standard processor system. In this chapter, I propose a programming model for its software integration and some examples of vector-based kernels. The vector data types allow the same programming model to be used for both METEOR and SIMD architectures, its integration in the compilation tool chain is detailed in Chapter 7.

5.1 Software Integration Overview

Since each tile in the METEOR architecture performs computations in parallel, a parallelism control over tiles should be used. The vector data type can manipulate large amounts of contiguous data, provides a strong Data-Level Parallelism (DLP), and many compilers support its format. The programmer's view of these vectors allows to execute computations between two vectors distributed over several memory tiles while still accessing the data sequentially.

According to a set of data-intensive applications, I propose an IMC/NMC instruction set operating in the IMC/NMC tiles of the proposed METEOR architecture. Thus, several instruction format are detailed to specify the type of operation, the operands and the result or constants to use in each vector instruction, as presented in the following sections.

5.1.1 Programmer's View

The dynamic horizontal reconfiguration and vertical transfers of METEOR allow to maintain a vertical memory continuity between each tiles. Indeed, the data does not move physically but logically with pointer remapping provided by the TAM, as described in Chapter 4. This specific memory remapping can be represented in a programmer's view, as a logical memory layout which is addressed on a physical memory map, as shown in Figure 5.1. Since the vectors can be dynamically reassigned per address vector, the programmer is able to choose the vector distribution parameters in the logical memory layout, such as the vector size and the stride pattern (for irregular memory accesses) of the vectors, as described in Section 5.1.2.

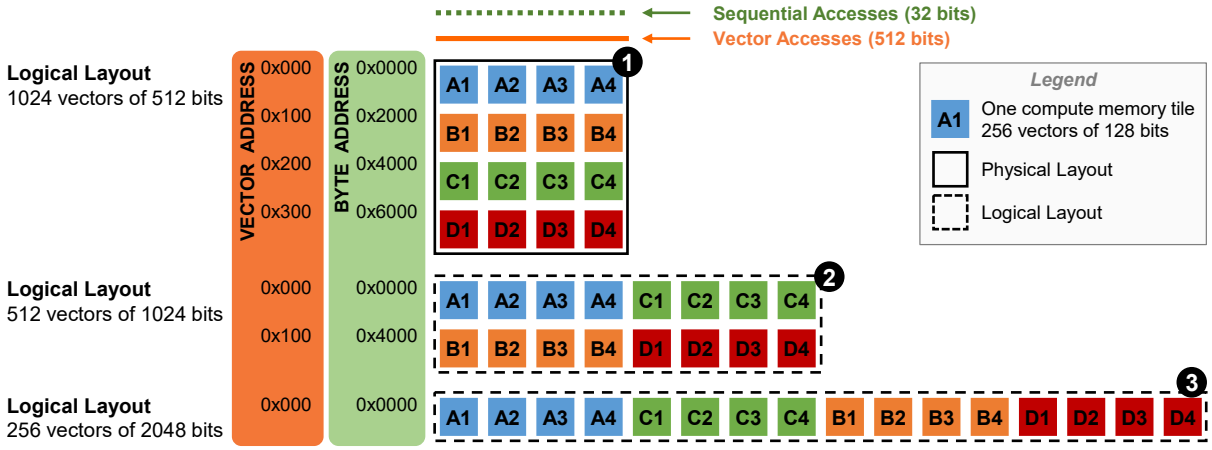


Figure 5.1: METEOR programmer's view of data accesses and vector handling. Byte addressing is used for sequential accesses (green), and the IMC/NMC instructions uses vector addresses (orange).

Figure 5.1 illustrates an implementation example of the METEOR architecture designed with 4×4 tiles with 256 vectors of 128 bits each. The data composing the vectors are stored and arranged horizontally in each tile according to the vector size required by the program, corresponding to a logical layout configuration (512, 1024, 2048 bits). For example, the program can assign 4 vector groups of 512 bits in A, B, C, D tiles, as in the logical layout ❶, and distribute the same instruction in parallel by configuring it to the layout ❷ to operate on 2 vector groups or to the layout ❸ to operate on all 4 vector groups. Indeed, each tile executes a part of the vector computation, and assembled horizontally, executes the whole computation on larger vectors. Moreover, it is also possible to arrange the data in 2048-bit vectors from configura-

tion ❸ to operate on smaller vectors of 512 bits. Processor scalar computations and 32-bit data accesses are still possible using byte addressing. Besides its computing capabilities, the METEOR architecture is still a data memory supporting standard load and store.

5.1.2 Layout Configuration Parameters

To configure the logical shape of the METEOR layout, the developer can use configuration parameters addressable through the control memory section and implemented in the Control and Status Register (CSR) unit of the METEOR architecture, as presented in the Figure 4.1. These parameters define a virtual grid pattern represented in Figure 5.2 to ensure the memory vector continuity. All parameters are listed in Table 5.1.

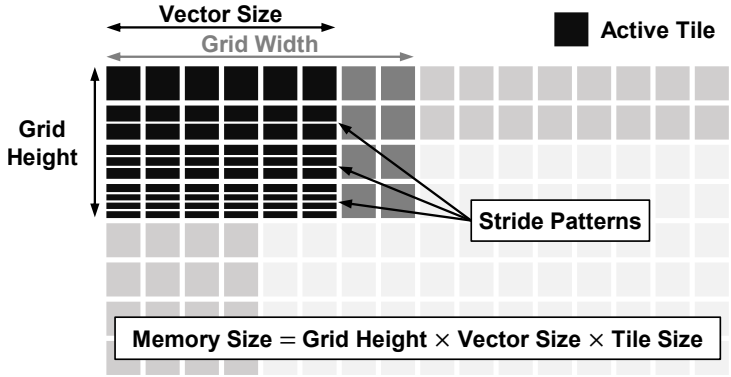


Figure 5.2: Layout configuration parameters.

To achieve an expandable architecture and still have vectors aligned with each other, it is necessary to have a number of tiles equal to a power of two. In Figure 5.2, this statement refers to the **Grid Width** and **Grid Height** parameters, that shape a rectangle of tiles. The **Grid Width** parameter defines the number of memory tiles arranged horizontally and the **Grid Height** parameter defines the number of memory stacked vertically. With the **Vector Size** parameter, the cluster can operate with vectors smaller or equal to the grid width, which saves energy by disabling unnecessary tiles but reduces the size of the virtual vector memory available (**Memory Size** parameter). The **Stride Pattern** parameter allows to write or read data vectors with an interleaving vector step. With this parameter, continuous memory blocks are equally distributed between several tiles, where the location of each vector is regularly alternated across the entire memory.

Table 5.1: Layout configuration parameter summary of METEOR (CSRs).

Parameter	Description	Reg #	Type	RD flag
Setter				
Grid Width	set the physical vector size (multiple of tile vector size)	1	U	0
Vector Size	set the logical active vector size (inferior to grid size)	2	U	0
Stride Pattern	set the read/write address stride pattern	3	U	0
Getter				
Grid Width	get the physical vector size (multiple of tile vector size)	1	U	1
Vector Size	get the logical active vector size (inferior to grid size)	2	U	1
Stride Pattern	get the read/write address stride pattern	3	U	1
Memory Size	get the total cluster active memory size	4	U	1
Grid Height	get the physical number of tile available	5	U	1

To configure the layout, a library of functions associated to the parameters is provided to the developer, to update the architecture (Setter) or to read the values of the layout parameters (Getter), as shown in Table 5.1. Moreover, the **Memory Size** and **Grid Height** are determined by hardware mechanisms of the GPD unit, as presented in Section 4.2.4. Thanks to these features, the developer has the possibility to dynamically reconfigure the architecture during the program execution with the `vreg` instruction, which is a part of the IMC/NMC instruction set as presented in the next section.

5.1.3 Instruction Set Formats

The instruction set defines which registers and instructions are supported by the processor and how these instructions and operands are represented in memory. The IMC/NMC instruction formats, detailed in Figure 5.3, have a total length of 56 bits and groups all instructions in three base formats (`R`/`I`/`U`), defined as follows:

- **R-Type** (Register Type): all instructions operating on two source vectors (`S2`, `S1`),
- **I-Type** (Immediate Type): all instructions operating on an immediate 16-bit (`imm[15:0]`) value and one source vector (`S1`),
- **U-Type** (Upper immediate Type): all instructions which load/store an immediate 32-bit (`imm[31:0]`) value in one destination vector (`D`).

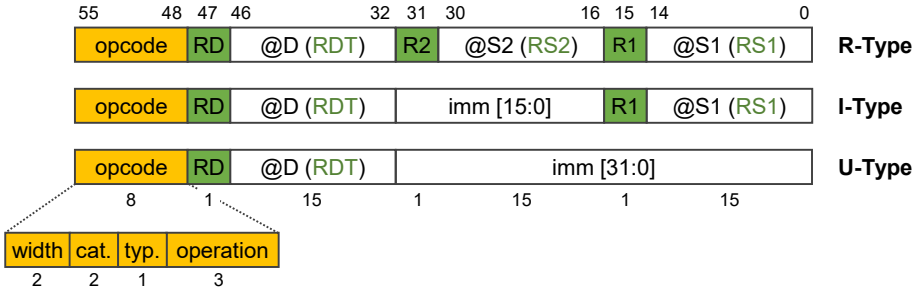


Figure 5.3: Instruction formats of IMC/NMC instructions with internal register support.

The 8-bit opcode field specifies the instruction operation, depending on the operand width, the operation category, the type format and the operation identifier, as detailed in Table 5.2. The 15-bit operands and destination fields refers as memory addresses (`@S1`, `@S2`, `@D`) or register numbers (`RS1`, `RS2`, `RDT`) according to their enable bits (`R1`, `R2`, `RD`). For example, when the `RD` bit is enabled, the destination operand field will be a value corresponding to a register number (`RDT`), else the destination operand field will a value corresponding to a memory address (`@D`).

These instruction formats enable to manage the internal tile register, as presented in the Section 4.2.1, in order to store intermediate computation results and to optimize the pipeline throughput. According to the logical layout configurations of the vertical and horizontal tiled assembly presented previously, the METEOR architecture can access a register stack addressable by its number according to the number of stacked tiles (Grid Height parameter). As the internal registers have the same size as the memory vectors, the consistency between memory and register is preserved, and the number of available vector registers depends on the layout configuration, as presented in Figure 5.1. In the METEOR layout configuration ❶, the software developer can address from `R0` to `R3` registers, in the configuration ❷, from `R0` and `R1`, and in the configuration ❸ only the `R0` register. Moreover, this internal register enhances

complex instruction execution, such as multiply-accumulate (MAC). So, the program executes the operations `fxmul` and `fxadd` sequentially to compute partial sums, which are stored in the internal register (accumulation register), thus reducing the number of reads and writes in the memory. This reduces the overall memory power consumption and increasing the pipeline throughput, as explained in Section 4.1.3.

Table 5.2: Instruction summary of METEOR (54 instructions in total).

Category	Type	Width (bits)	Operations (28)
memory	I	Line	copy, hswap64, hswap128
	R	8, 16, 32	copyeq, copygeq, copygt, copyleq, copylt, copyneq
	U	8, 16, 32	bcast
logical	I	8, 16, 32	slli, srli
	I	Line	not, redor
	R	Line	and, or, xor, nand, nor, xnor
arithmetical	I	8, 16, 32	abs
	R	8, 16, 32	add, sub, cmp
	R	8	fxadd, fxmul, mul
CSR	U	32	vreg

After studying the requirements of data-intensive applications, the instruction set can be classified into four categories, as presented in Table 5.2. The categories are as follows: (1) memory: for data manipulations, (2) logical: for operations using binary data, (3) arithmetical: for complex operations using integer and fixed-point data (`fxmul`), and (4) CSR: to configure the METEOR layout through the CSR unit.

5.2 Instruction Set Architecture

The Instruction Set Architecture (ISA) serves as the boundary between software and hardware. It describes the design of the basic operations that the system must support from the programmer's perspective. Previous work [94] demonstrated the compatibility between a conventional architecture and the IMC and NMC concepts, integrating the ISA on the system bus without modifying the design of the processor. Indeed, the previous instruction formats proposed can be integrated on the system bus to send instructions to a dedicated control interface memory section. In this thesis, the improved IMC/NMC ISA allows to extend this compatibility to several computation tiles and to manage intermediate registers. Furthermore, this ISA will be integrated in the software and hardware layer of a simulation platform to evaluate the impact of the IMC/NMC at the system level, as presented in the Chapter 7.

5.2.1 System Bus Integration

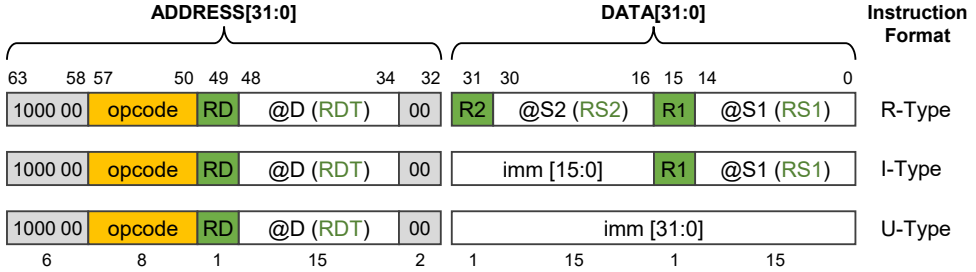


Figure 5.4: IMC/NMC ISA integrated on a standard 32-bit system bus.

To integrate the proposed instruction set into an architecture, there are different solutions, such as adding a control line between the CPU and the memory, or adding new instructions to the CPU. To ensure a smooth integration and minimize system modifications, the instruction formats are encoded using the actual address bus and data bus, as shown in 5.4. A 32-bit system bus offers a total space of 64 bits. For bigger architectures, it is possible to integrate the ISA on a 64-bit system bus, allowing larger system addressing. The 6 most significant bits (MSB) of the address define the start address of the control interface memory section, detailed in the following section. However, the load-store unit of the processor, responsible for the core's memory accesses, ensures that 8-bit and 16-bit words are aligned with 32-bit words. In order to avoid automatic word alignment that modify the instruction sent, the 2 least significant bits (LSB) of the address are left at zero.

5.2.2 Control Interface Memory Mapping

From the execution perspective, sending a IMC/NMC instruction to the computing memories is equivalent to writing a specific data to a specific address in a specific memory section. According to the proposed ISA, the 6 MSB of the address bus defines the address range size of 64 MB and starting at the address `0x8000 0000` for the control interface memory section. To simplify the system integration, the vector operand bit fields are encoded on 15 bits allowing the manipulation of 32 k vectors. If the minimum vector size in the METEOR architecture is 512 bits, this corresponds to a maximum addressable physical data memory size of 2 MB. Thus, the physical addressable memory size depends on the control interface memory size. This limitation can be overcome by using a 64-bit architecture.

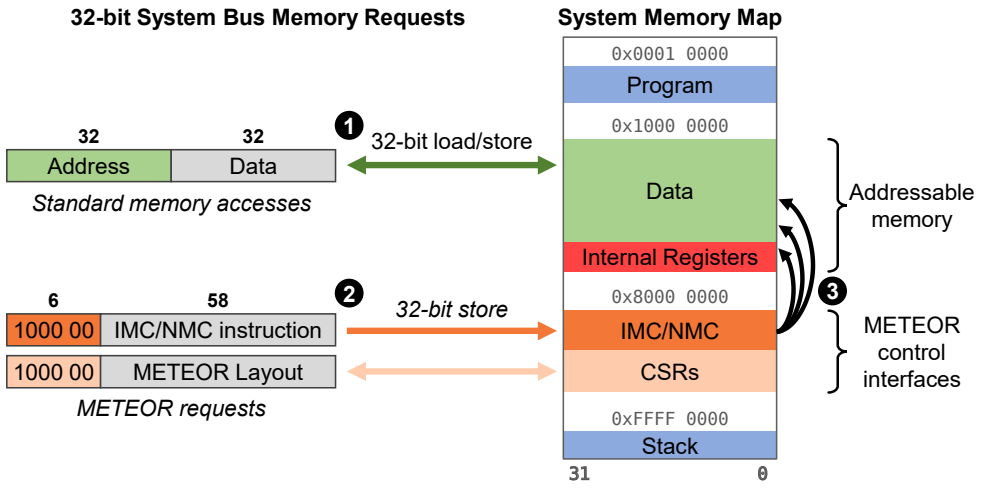


Figure 5.5: METEOR control interface integrated in a standard system memory map. Request ❶ is used for standard memory accesses and request ❷ is an IMC/NMC instruction or a layout parameter send to the METEOR architecture then ❸ to distribute to each tile or to configure a CSR.

Figure 5.5 illustrates the memory requests that the processor can perform with the METEOR architecture. The processor can ❶ access a data or internal register of the memory through the standard access, from the address `0x1000 0000`, where data is aligned according to the METEOR vector size. When sending an IMC/NMC instruction, the request ❷ passes through the METEOR control interface to ❸ dispatch the instruction to the IMC/NMC tiles, thus editing the content of the physical memory. In the same way, the processor address a new METEOR layout configuration via the same control interface to write or read the parameter in the CSRs.

Listing 5.1: Operation `vmul8`: C macro of 8-bit chunk multiply operation on a C-SRAM vector.

```
1 #define vmul8(_result, _operA, _operB) do {                                    \
2     uint64_t            cm_word = _MAKE_ISA_RTYPE(OPC_MUL8, _result, _operA, _operB); \
3     volatile uintptr_t* cm_addr = (uintptr_t*)((uint32_t)(cm_word >> 32));     \
4     uintptr_t           cm_data = (uint32_t)(cm_word);                         \
5     *cm_addr = cm_data;     /* equivalent to a store instruction */            \
6     asm("" ::: "memory");   /* memory fence */                                 \
7 } while(0)
```

From the software-level perspective, the ISA defines IMC/NMC instructions as a set of C macros to perform operations on data vectors, as proposed in Listing 5.1. On line 2, the bit fields of the IMC/NMC instruction is assembled onto a 64-bit word, as proposed in Figure 5.4, to be assigned to a 32-bit address and 32-bit data on lines 3 and 4, respectively. Then, the instruction is sent to the control interface memory by writing the data to the address, on line 5. In order to enforce an ordering constraint on memory operations, which would alter the behavior of the program, on line 6, the memory fence instruction prevents the compiler to reorder memory operations. To conclude, this IMC/NMC ISA proposal avoids any modification of the instruction set of the processor, hence the IMC/NMC instruction assembling and forwarding to the control interface will be optimized by the compiler.

5.3 Programming Model for Scalable Vector Processing

The programming model acts as a bridge between algorithms and actual implementations in software. Here, the parallel execution model of the IMC/NMC approaches must be interleaved with the execution model of the processor, in order to distribute the computation to the memory tiles of the METEOR architecture. A first version is available using static C-macro for an early evaluation of the IMC/NMC instructions at the software level. Further optimizations will extend this model with dedicated instructions to support dynamic vector reconfiguration and micro-codes to assemble several IMC/NMC instructions in an optimized way.

In the following code listings, the tiles composing the METEOR architecture are defined as CSRAM, for Compute SRAM, due to the IMC/NMC design implementations based on SRAM.

Vector/Matrix representation – To represent the matrices in the memory, each row of the matrix is considered as a vector (or multiple vectors) containing the elements ordered by column. When the vector size is smaller than the number of columns in the matrix, then the rows of the matrix are represented on multiple vectors. Depending on the METEOR layout configuration, it is possible to adjust the vector size (line 5: `VECTOR_SIZE`) to the size of the matrix, as shown in 5.2. As described in the programmer's view, the data can be accessed with the appropriate scalar type (`i8`, `i16`, `i32`) and the vector computation can be performed with the vector type (`v`).

Common vector format – Since the matrix dimensions are known, the maximum vector size is fixed at compile time for the METEOR and SIMD architectures (128, 256 and 512 bits). To compare these vector architectures, the proposed `VectorLine` type (line 17) is used for the definition, the declaration and the allocation of the vector variables. The `union` (line 12) feature allows to share the same memory location for different data types (`v`, `i8`, `i16`, `i32`), so the vector can be addressed with 8, 16 or 32-bit scalar accesses. Moreover, the data is

Listing 5.2: SIMD and CSRAM vector types.

```
1  /* Defined when 512-bit SIMD vectors are used */
2  #define VECTOR_SIZE     64
3  #define VECTOR_TYPE      __m512i /* data type define in <avx512fintrin.h> */
4  /* Defined when 8192-bit CSRAM vectors are used */
5  #define VECTOR_SIZE     1024
6  #define VECTOR_TYPE      __CSRAM_Line /* data type define below */
7
8  /* CSRAM data type as large as the vector (up to 8192 bits) */
9  typedef int __CSRAM_Line __attribute__((vector_size(VECTOR_SIZE)));
10
11 /* Common vector type for every benchmark evaluation, data are always aligned */
12 typedef union {
13     VECTOR_TYPE     v;                                  /* used for vector handling      */
14     int8_t          i8 [VECTOR_SIZE/sizeof(int8_t) ]; /* used for  8-bit word accesses */
15     int16_t         i16[VECTOR_SIZE/sizeof(int16_t)]; /* used for 16-bit word accesses */
16     int32_t         i32[VECTOR_SIZE/sizeof(int32_t)]; /* used for 32-bit word accesses */
17 } VectorLine __attribute__((aligned(VECTOR_SIZE)));
```

automatically aligned to the vector size at allocation using the `aligned` attribute (line 17) and `vector_size` attribute (line 9), which are supported by many C/C++ compilers (GCC).

Since the same data-level parallelism is targeted, this vector format allows to reuse directly low level vectorized kernels already written for SIMD architecture for our METEOR architecture. This avoids the tedious work of application mapping and parallelism extraction.

5.3.1 Vector Processing Capability

Beyond this vector format compatibility, the METEOR architecture can adjust the vector size in run-time, referred as dynamic vector reconfiguration. The data pointers are physically re-addressed through the TAM to adapt each scalar or vector access and the vertical communication paths are re-mapped to perform operations between vectors at different size, as presented in Section 4.2.3. Thus, the proposed `vreg` instruction is wrapped in a C library providing several functions, such as `set_mvl` for "*set METEOR vector length*", to properly configure the METEOR layout according to the user program. Also, the METEOR internal registers are addressable during the reconfiguration from the programmer's perspective to compute between intermediate vectors or to retrieve the final results (`CRegs` memory section), as presented in Listing 5.4. These reconfiguration mechanisms allow better manipulation of intermediate data in memory, which is a challenge to implement in conventional vector accelerators. Indeed, the flexibility of the interconnect in the memory allows complementary data management to the vector accelerators.

Furthermore, to abstract reconfiguration mechanisms for the developer, dedicated micro-codes are implemented as complex instructions, assembling pure vector and scalar instructions. Thus, in the Listing 5.3, the reduction operation (`vreduce_add8`) is a micro-code involving METEOR's reconfiguration to avoid additional memory transfers to the processor, as detailed in Section 5.4.2. The data of A, B and C matrices are stored in the vector memory (line 4), aligned according to the given vector size (line 2). The rows of A matrix are multiplied by the column of B matrix (line 11), then the elements of the `tmp` intermediate vector line are added together (line 12) and the result is stored in the C matrix (line 14), element by element.

Listing 5.3: Example of N×N matrix product written in C language

```
/* Matrix data type, reshape according to the vector size */
typedef VectorLine Mat[N][N / VECTOR_SIZE];
/* All matrices will be stored aligned in the vector section (.csram) */
__attribute__((section(".csram"))) Mat A, B, C;

/* Example of 8-bit word NxN matrix multiplication for SIMD and C-SRAM architectures */
for(int i = 0; i < N; i++)
    for(int j = 0; j < N; j++) {
        int sum = 0; VectorLine tmp;
        for(int k = 0; k < N / VECTOR_SIZE; k++) {
            vmul8(tmp[k].v, A[i][k].v, B[j][k].v);  /* vector instruction */
            vreduce_add8(sum, tmp[k].v);            /* dedicated micro-code */
        }
        C[i][j / VECTOR_SIZE].i8[j % VECTOR_SIZE] = sum;
    }
```

This kernel can be compiled for SIMD architectures from 128 bits to 512 bits as well as the METEOR architectures using larger vector sizes. These benefits are discussed in Chapter 8.

To conclude, a first version of the programming model for scalable vector processing is proposed for ease of programming for any vector size, vector alignment and dynamic vector reconfiguration. Although this is still a low-level ISA, which is implemented as a set of C macros at the developer level, all translation work is supported by the compiler to generate optimized IMC/NMC instructions. Moreover, internal registers and layout configuration parameters are tightly integrated to this ISA to provide user-friendly micro-code for flexible architecture reconfiguration. Many software research opportunities are still needed to integrate IMC/NMC ISA into the processor instruction set, as well as a seamless interlacing of the architecture reconfiguration mechanisms in a parallel programming model, for the development of data-intensive applications.

5.4 Vector Data-centric Kernels

By analyzing data-intensive applications, some kernel (sub-parts of the application) can be implemented using the proposed IMC/NMC programming model. In this section, I present a few kernel examples integrating IMC/NMC instructions, operation reduction micro-code and refined data placement to fully exploit METEOR reconfiguration. Today, they serve as a micro-benchmark for the development of complete data-intensive applications.

5.4.1 Shared Data Memory

Convolution and Fully-Connected (FC) kernels are mainly used in Convolutional Neural Network (CNN) applications. These kernels use three groups of vectors to store input, output and coefficient data and arithmetic operations such as multiplication and addition. Thanks to the METEOR architecture, these operations are performed with large vectors, thus increasing the data parallelism. The vertical communication scheme of METEOR allows to transfer the intermediate vector lines through the entire memory, where the operands are located, to maintain the processing throughput, as presented in the Section 4.1.2. In addition, the horizontal vector reconfiguration allows to adapt the vector size according to the neuronal layer requirements.

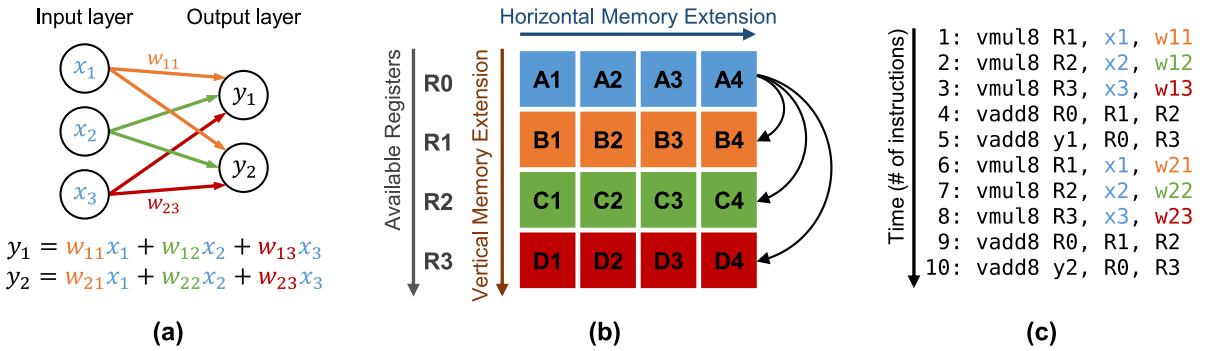


Figure 5.6: Fully Connected (FC) kernel. **(a)** Neural network layer representation, **(b)** Vector representation, **(c)** Pseudo instruction flow representation.

The FC kernel is used in the final layers of CNN, using a low memory area for input data and a high memory area for weight data. In the Figure 5.6, the data of the input layer (*x1*, *x2*, *x3*) stored in the A tiles are multiplied by a large amount of weights (*w11*, *w12*,... *w23*) distributed in tiles B, C, D tiles. The vertical transfers move the input data to the tiles containing the weights to perform multiplications and additions, thus avoiding the input data duplication in tiles B, C, D and reducing the memory footprint.

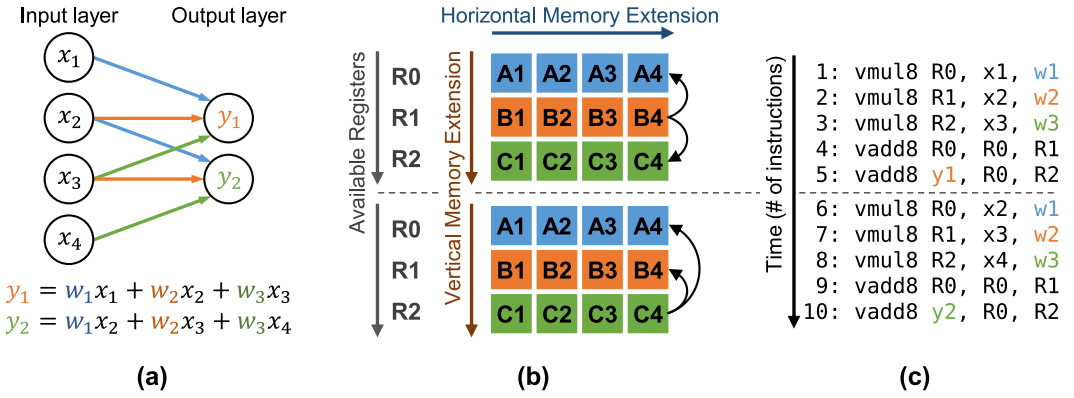


Figure 5.7: Convolution kernel. **(a)** Neural network layer representation, **(b)** Vector representation, **(c)** Pseudo instruction flow representation.

The convolution kernel is used in the main layers of CNN, using a high memory area for input data (image) and a low memory area for weight data (filters). In the Figure 5.7, the weight data (*w1*, *w2*, *w3*) are stored in the A, B, C tiles and input data are distributed in all tiles, thanks to the stride pattern distribution. The vertical transfers move (up/down) the weight data to the tiles containing the input data to perform multiplications and additions, thus allowing weight data sharing between the B, C and D tiles and reducing the memory footprint.

5.4.2 Reduction Operations

The reduction operation consists in performing the same operation (e.g. an addition) between all the elements composing a vector, as used in the previous code listings or in different matrix product kernels of the *PolyBench* suite [95]. The skeleton behavior of the 8-bit reduction addition is shown in Figure 5.8 and the specific micro-code used by the METEOR architecture is detailed in Listing 5.4.

Thus, after performing an 8-bit multiply (`vmul8`) between two vector of 2048 bits, the METEOR architecture is able to perform the reduction addition (`vreduce_add8`) using the proposed IMC/NMC instructions, the dynamic vector configuration functions and the internal registers. The first part of the micro-code, involves successive vector downsizing using the `set_mvl` functions (including `vreg` instruction), and compute 8-bit additions between sub-vectors using the internal registers, in order to increase the pipeline throughput (line 3 to 7). The second part of the micro-code, involves successive IMC/NMC instructions to horizontally swap the sub-element of the vector, using internal registers, and compute 8-bit additions between the swapped and the original vector (line 9 to 16). Finally, the last part of the micro-code is performed by the CPU thanks to the internal register accessibility to perform the last 8-bit scalar reduction (line 19). This micro-code is quite consequent, but it spares 256 scalar instructions, when executed only on a single CPU, replaced by 18 meta-instructions.

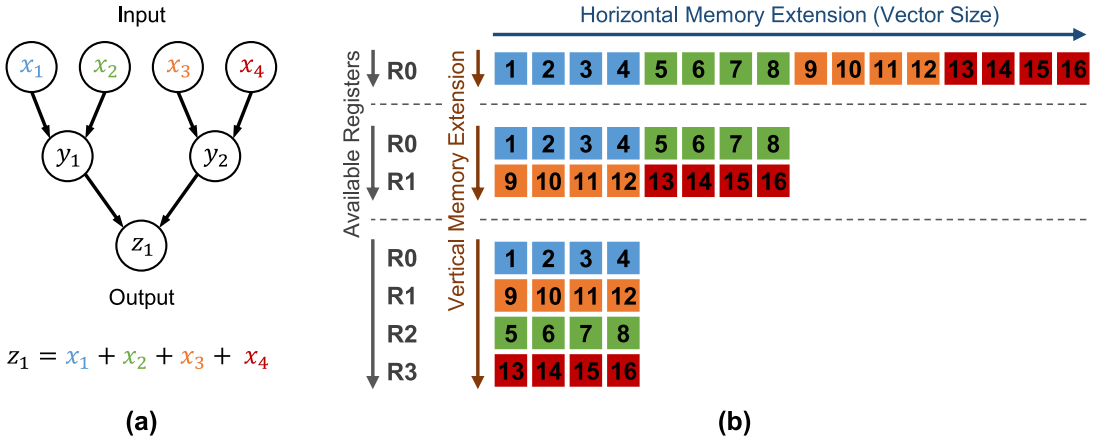


Figure 5.8: 8-bit integer addition reduction of 2048-bit vectors (A and B) with results in C. **(a)** Adder tree representation, **(b)** Vector representation (from 2048 bits down to 512 bits).

Listing 5.4: Operation `vreduce_add8`: sum of 256 8-bit words of a 2048-bit vector width. This version uses METEOR dynamic vector reconfiguration and C-SRAM internal registers.

```
1  #define vreduce_add8(_8b_dest, _v_src) do {                                    \
2      /* METEOR reconfiguration: dynamic vector and addition (2048 bits -> 512 bits)   */ \
3      _cm_copy_rm(    R0, _v_src );       /* sum = vector_source[2047:0]          */ \
4      set_mvl(              1024 );       /* set METEOR vector length at 1024 bits */ \
5      _cm_add8_rrr(   R0, R0, R1 );       /* sum = sum[2047:1024] + tmp[1023:0]   */ \
6      set_mvl(               512 );       /* set METEOR vector length at 512 bits  */ \
7      _cm_add8_rrr(   R0, R0, R2 );       /* sum = sum[1023:512]  + tmp[511:0]    */ \
8      /* CSRAM reduction: each tile operates a software adder tree (128 bits -> 8 bits)*/ \
9      _cm_hswap64_rr( R1,     R0 );       /* tmp[63:0] = sum[127:64]              */ \
10     _cm_add8_rrr(   R0, R0, R1 );       /* sum[63:0] = sum[63:0] + tmp[63:0]    */ \
11     _cm_hswap32_rr( R1,     R0 );       /* tmp[31:0] = sum[63:32]               */ \
12     _cm_add8_rrr(   R0, R0, R1 );       /* sum[31:0] = sum[31:0] + tmp[31:0]    */ \
13     _cm_srli32_rr(  R1, R0, 16 );       /* tmp       = sum >> 16                */ \
14     _cm_add8_rrr(   R0, R0, R1 );       /* sum[15:0] = sum[15:0] + tmp[15:0]    */ \
15     _cm_srli16_rr(  R1, R0,  8 );       /* tmp       = sum >>  8                */ \
16     _cm_add8_rrr(   R0, R0, R1 );       /* sum[ 7:0] = sum[ 7:0] + tmp[ 7:0]    */ \
17     set_mvl(              2048 );       /* set METEOR vector length at 2048 bits */ \
18     /* CPU reduction: 5 memory accesses for the last reduction (4 reads, 1 write)  */ \
19     _8b_dest = CRegs[0]->i8[0] + CRegs[0]->i8[16] + CRegs[0]->i8[32] + CRegs[0]->i8[48];\
20 } while(0)
```

In conclusion, the proposed micro-code benefits from the different features proposed in the METEOR architecture. At the hardware level, the dynamic reconfiguration and the vertical communication scheme enable computation between the different IMC/NMC tiles, supporting vector operand transfer, and the internal registers provide an intermediate energy-efficient buffer and high pipeline throughput, as presented in the Chapter 4. At the software level, the enhanced IMC/NMC instruction set allow to support scalable vector processing for any vector size, vector alignment and vector allocation, and the early version of the programming model allows seamless integration of the micro-code according to the METEOR architecture execution model.

5.5 Conclusion

In order to execute vector-based kernels in our METEOR architecture, the integration of a dedicated IMC/NMC instruction set and the METEOR layout parameters are proposed. Integrating the internal register in the instruction format allows to optimize the instruction flow in the multi-tile architecture as well as the execution of dedicated micro-codes. This will be evaluated by simulation at the system level in the Chapter 8. Moreover, the METEOR architecture is compatible with standard architectures, without modifying the design of the processor, thanks to the proposed programming model while ensuring support for other vector architectures.

However, this is still a low level ISA, which is implemented as a set of C macros, and resolved at compile time. The current ISA and micro-code library allows to evaluate kernels of data-intensive applications by exploiting a large data parallelism. In a long-term perspective, this high-level library will be extended to explore complete data-intensive applications in order to measure the benefits of IMC and NMC approaches on a larger scale.

Chapter 6

Design Space Exploration of the Memory Interconnect

Contents

Since a unique large memory instance cannot satisfy the capacity and performance requirements of data-intensive applications, an architecture composed of multiple memory tiles (multi-tile) is necessary to design larger memories. In this chapter, I have evaluated the low-level constraints of multi-tile systems and developed a model of the interconnect between memory tiles. Such a model facilitates the scaling of the METEOR architecture with the design constraints estimated for multi-tile architectures. These estimations serve as input to the simulation platform, presented in Chapter 7, in order to improve the accuracy of architectural explorations.

6.1 Interconnect Overview

The design of Integrated Circuits (ICs) or chips consists of many development steps, known as the physical design flow, necessary to ensure the required functioning of the components and to build an error-free chip. This physical design is divided into two categories: full-custom designs, in which the designer has full flexibility in layout design, and semi-custom designs, in which the designer uses pre-designed library cells (standard cells) and has the flexibility in cell placement and routing. Usually, the inputs to physical design steps are (1) a netlist, composed of the design components and their connections, (2) standard cell libraries of each device composing the design, and (3) a technology file containing the manufacturing constraints. Finally, this design flow generates a layout file containing planar geometrical shapes and patterns of the metal, oxide and semiconductor layers of the chip to be manufactured on a silicon wafer.

Synchronous digital circuits are systems where data processing is coordinated by a clock signal. This clock signal is distributed globally to all the sequential elements of the circuit in order to simultaneously synchronize the data processing through these elements. Large memory systems are composed of memory tiles (or memory cuts or memory instances) connected together with a global distribution interconnect, mainly made up of the clock signal, data bus (input and output), address and control signals specifying the reading or writing operations. The larger the system, the more difficult it becomes to synchronize the arrival times of the same clock edge to different sequential elements.

To synchronize data arrival times between elements, H-tree distribution interconnects are commonly used in low power designs due to their cost efficiency, as presented in Figure 6.1(a). In this interconnect, the internal clock tree network achieves equal propagation delays to all memory from the clock source signal to every memory thanks to clock buffers, as presented in Figure 6.1(b). The clock buffers are placed between the clock source and along the clock paths according to the arrival time requirements as detailed in Section 6.2.2.

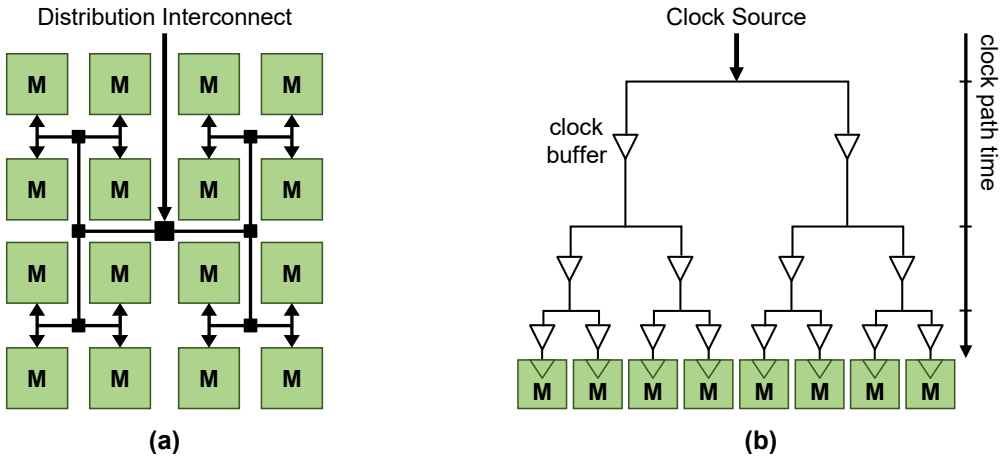


Figure 6.1: Memory interconnect: **(a)** H-tree distribution interconnect and **(b)** clock tree network

.

The multi-tile METEOR architecture, presented in the previous chapters, offers a strong parallelism achieved by performing the computation in several tiles in parallel. To dispatch IMC/NMC instructions to each tile, the same distribution interconnect (TDI) as the memory interconnect is used, as described in the Chapter 5. Moreover, this efficiency of this parallelism on the vectorization width (i.e. vector size) which is proportional to the number of tiles used.

Unfortunately, by considering a realistic memory interconnect, the larger the vector, the lower the timing performance will be, as presented in this chapter. This study focuses mainly on the TDI of the METEOR architecture without considering the additional vertical communication scheme (VTI). Nevertheless, the design constraints related to the memory interconnect of this study, provide an overview of the performance limitations of this architecture.

The Design-Kit (DK) of SRAM provided by foundry allows to explore the performance of the individual memory tiles (capacity, area, performance) to build a large system, but not their memory interconnect between tiles. Hence, I propose to provide a generic memory interconnect model of multiple SRAM using the standard physical design flow.

6.1.1 SRAM Organization

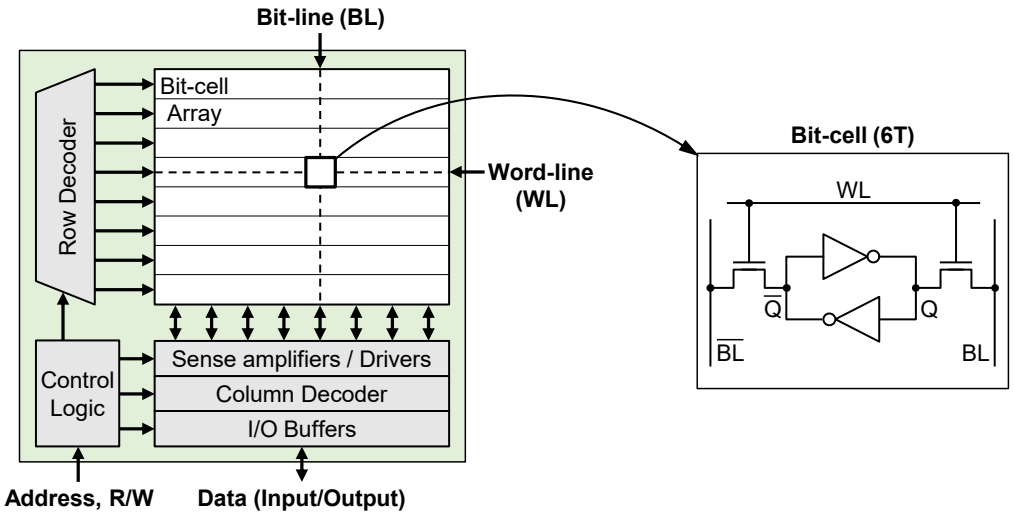


Figure 6.2: SRAM tile architecture (6-Transistor bit-cells).

The SRAM tile architecture, also called a memory cut or instance, as generated by foundry generators is usually organized as in Figure 6.2. It is composed of an array of bit-cells, each bit-cell retaining the value of a single bit (0 or 1), a row decoder and a column decoder responsible for addressing the memory word in vertical and horizontal lines, senses amplifiers and drivers to digitally read or write the data in the bit-cells, input/output (I/O) buffers to interface with external components and a control logic unit to drive the internal elements during memory requests. According to the data location in the bit-cell array, the row decoder selects the Word-Line (WL) and the column decoder selects the Bit-Lines (BLs) to access to the data. When reading data, low-power signals coming from the bit-lines are amplified to recognizable logic levels by the sense amplifiers before sending the data outside of the memory with the output buffer. When writing data, the input buffer transfers the data through the drivers, which adapt the current level to write each bit-cell value.

Other architecture parameters exist to optimize memory performance, such as the number of transistors per bit-cells (4T, 6T, 8T,...) for multi-port SRAM, multiplexed memory bank partitioning, clock gating to reduce energy consumption and read and write assist techniques. These optimizations satisfy hardware constraints in order to have: dense memory area (high-density), fast memory access time (high-performance) or energy efficient memory (low-power). In any implementation or technology, the size of the bit-cell array will defines the trade-offs between access time performance, power consumption and surface area.

6.1.2 Performance and Power Impacts

The performance of memory devices has a major impact on the overall performance of the system architecture. To reduce the technological impact of scaling and to achieve higher performance than the standard design flow, memories are designed and optimized by foundries and design companies. Indeed, during the design of integrated circuits, the Design Rule Checking (DRC) step verifies the spacing between the components (gates, transistors, ...) to ensure the function of the circuit. The reliability of SRAM bit-cell is characterized by specific design spacing rules defined by the founders, who sell memory cut libraries optimized for a given technology node. The proposed design space exploration involves SRAM cuts optimized by founders to target high-performance multi-tile designs.

6.2 Evaluation Methodology

Regarding the performance and power trade-offs between SRAM tiles and the distribution interconnect, a multi-tile interconnect exploration model is proposed to analyze the architectural constraints of the wiring cost. This model is calibrated on a design space exploration studying various SRAM designs implemented in a 28 nm FD-SOI technology node and uses high-performance SRAM cuts provided by *STMicroelectronics* (ST). Although this model does not include the IMC/NMC tiles proposed in previous chapters, the SRAM design mimics its behavior and allows to extrapolate the memory interconnect impact.

6.2.1 Physical Design Flow

The proposed evaluation methodology is based on a standard physical design flow used to design digital integrated circuits, as shown in Figure 6.3. In this study, the design space exploration parameters depend on the data-intensive application requirements (e.g. memory size) presented in Section 8.1. The architecture of the memory system is implemented through the standard design flow made up of: ❶ the description of the architecture behavior in RTL, ❷ the memory array netlist of each system elements (registers, gates, ...) through the logic synthesis step, ❸ the placing and routing (P&R) of components on multiple floorplan configu-

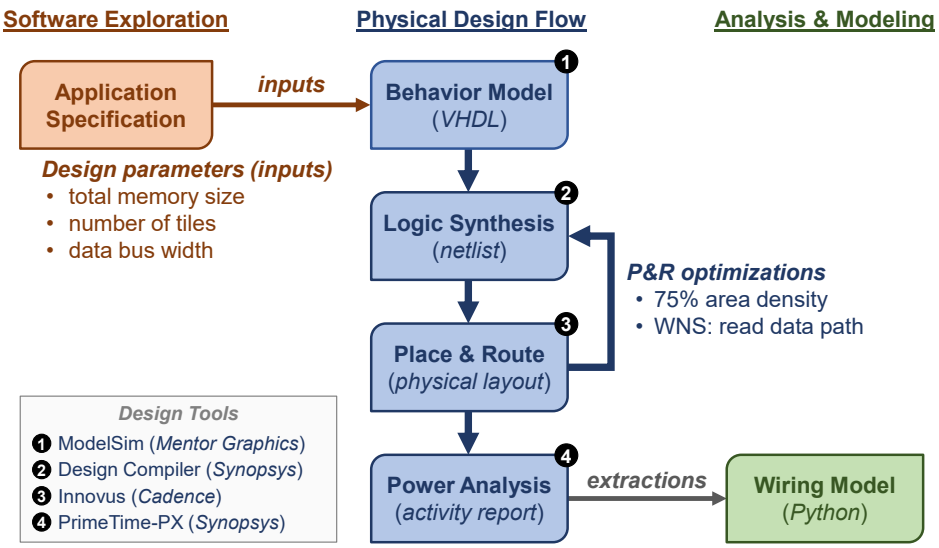


Figure 6.3: Evaluation methodology based on a standard physical design flow.

rations through many circuit optimizations (clock tree synthesis, static timing analysis (STA), ...), ❹ the timing and power extractions according to the system activity traces. These extractions combine key parameters to create the wiring interconnect model using polynomial regression algorithms.

To optimize system performance, the design is constrained by the memory tile placement on a given floorplan and the system frequency. The P&R optimization steps minimize the time latency between circuit components still perform system functionality. In every designs, a symmetrical memory tile placement - such as the H-tree interconnect, for a balanced latency time distribution - and a design density of 75% are targeted. In addition, the frequency factor constrains the latency of every path between the components until the critical path is minimized. Static Timing Analysis (STA) method analyses each path of the system to find the critical path, as detailed in Section 6.2.2. As both physical design flow and P&R optimization steps require development work and time, the wiring interconnect model quickly estimates power and performance with the same input parameters as the physical design flow, as detailed in Section 6.3.2. To enable fast analysis and iterations, the methodology has been automated on various configurations using various scripts.

6.2.2 Static Timing Analysis

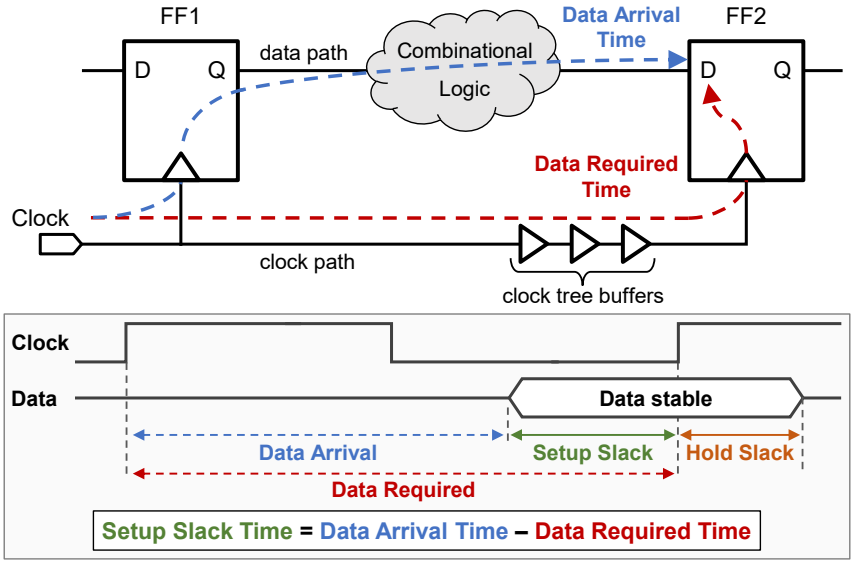


Figure 6.4: Static timing analysis (STA) of the Flip-Flop (FF) to Flip-Flop path.

The P&R step has solver algorithms (e.g. timing optimizers) to optimize circuit timings according to the clock frequency, arrival times and the data validity occurring at the clock edges. To modify the arrival times, the solver adjusts the placement of standard cell (not the SRAM) in the floorplan and adds inverters (buffer) along the data or clock path to slow down signal propagation. The STA provides time metrics such as the register-to-register access time, also represented by Flip-Flop (FF) in Figure 6.4. Such standard Flip-Flop (FF) model is used since the SRAM memory is clocked and behaves as a register. The setup slack time is defined as the amount of time when the data is stable before the active clock edge arrives. The hold slack time is defined as the amount of time when the data is stable after the active clock edge. In design exploration, the critical path is defined as the path with the worst setup slack time of all paths in the circuit, also called the Worst Negative Slack (WNS) time.

6.2.3 Multiple Memory Tile Exploration

The multi-tile design space exploration is based on a system composed of multiple memories and a FSM, in order to verify the functioning of the system after the Place and Route (P&R) step for power analysis. This system is developed into a generic RTL and applied for various multi-tile design configurations and various memory sizes. These designs have mirrored floorplans composed of tiles with the same memory size to achieve a balanced distribution interconnect, such as H-trees.

During the exploration, the STA indicates that the WNS of each design is located between the memory and the reading register (considering an external register to the multi-tile design). As shown in Figure 6.5, to read data from the memory, an additional logic (readout multiplexer) is implemented to select the tile storing the data. The critical path (or WNS) is composed of: the memory internal delay (T_{CD}), the net delay and the read logic delay (T_{DE}). In this exploration, the read access time serves to quantify the worst timing performance of each multi-tile design.

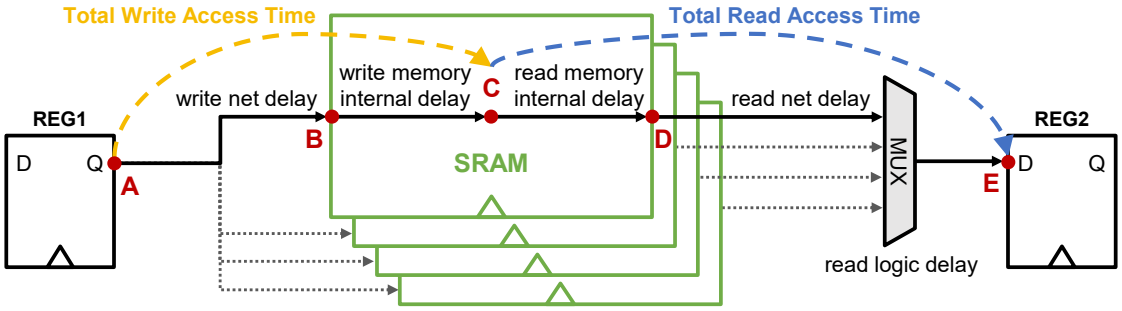
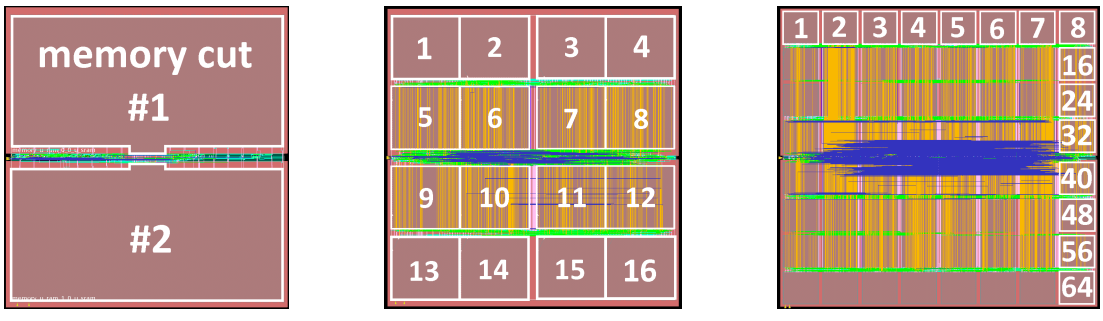


Figure 6.5: Multiple memory tile exploration: write (T_{AC}) and read (T_{CE}) timing paths.

6.3 Experimental Results

The design space exploration is a set of 22 different multi-tile circuits composed of 10 SRAMs tiles provided by the cut generator in ST 28 nm FD-SOI technology (for memory size of 64 Bytes up to 32 kB). The multi-tile design parameters are the number of tiles (or cuts) and the total memory size. The distribution interconnect width is fixed at 32 bits, as well as the memory interface. A custom floorplan is defined according to the memory cut size, as presented in



(a) (b) (c)

Figure 6.6: Floorplans of different multiple tile design circuits (not to scale). **(a)** 2 cuts of 8 kB, **(b)** 16 cuts of 1 kB, **(c)** 64 cuts of 1 kB. All pink objects surrounded by white is a memory cut, green nets are wires on metal layer 4 (M4), yellow nets are wire on M5 and blue nets are wires on M6.

Figure 6.6. These floorplans are placed and routed through the standard design flow presented previously for three architectures respectively composed of (a) 16 kB, (b) 16 kB and (c) 64 kB total memory size.

6.3.1 Performance, Power and Area Trade-offs

The performance (read accesses) of designs composed of 1, 4, 16 and 64 memory cuts is presented in the Figure 6.7. These curves result from the timing extractions of the founder's cut generator (for the "no wiring" curves) and from the timing analysis of the implemented and estimated designs (for the "wiring" curves). Each curve refers as a fixed number of memory cuts for a given total memory size. For example, to obtain the red curve for a total memory size of 64 kB, the architecture is composed of 16 memory cuts of 4 kB each.

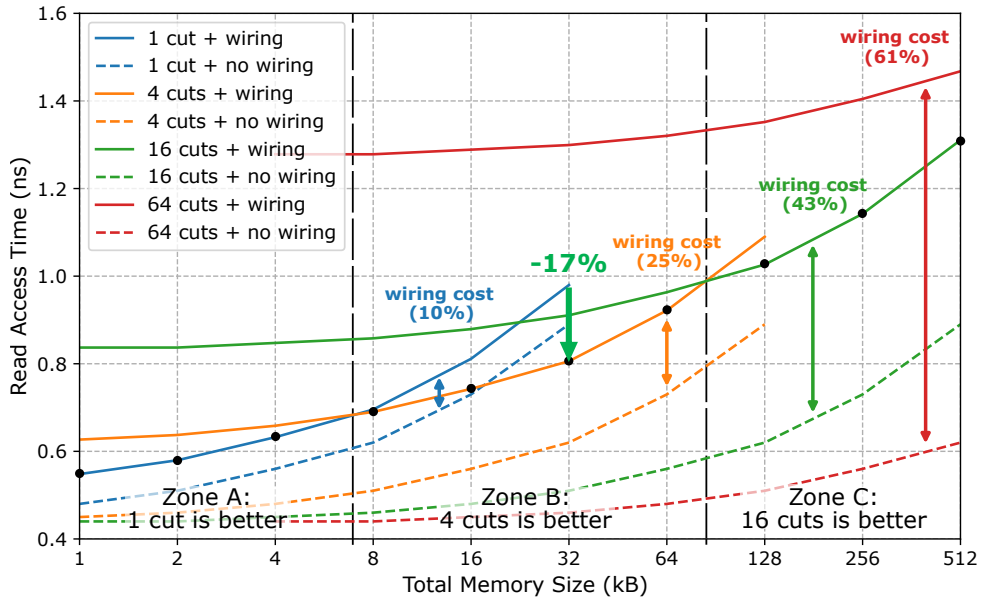


Figure 6.7: Multi-tile (read) timing performance versus memory size for various multi-tile designs.

For small memory sizes (Zone A), the single cut architecture has better performance than multi-tile architectures (Zone B and C) but for large memory sizes, the memory bit-cell access time becomes significant. Thus, three zones define the multi-tile architectures with the best performance according to memory size, such as: the 1-cut architecture for memories from 1 kB to 4 kB (Zone A), the 4-cut architecture for memories from 8 kB to 64 kB (Zone B) and the 16-cut architecture for memories from 128 kB to 512 kB. In addition, for a total memory size of 32 kB, switching from the single cut architecture to the 4-cut architecture provides a timing performance gain of 17 %. In summary, memory interconnect must be considered when designing large memory architectures because the founder's cut generator does not reflect the performance of large multi-tile systems. This wiring cost is proportional to the number of cuts (from 10 % up to 61% of the access time according to these architectures) and will be considered for architectural exploration and model calibration in Section 7.5.2.

In Figure 6.8, the impact of increased tile fragmentation (increase the number of tiles for a given memory size) is presented in terms of energy and area for a total memory size of 32 kB. Compared to the same point as the performance results, by switching from a 1-cut to a 16-cut architecture, the total energy access is reduced by 78%, but the surface area is increased

by 1.8×. Indeed, small memory cuts involve smaller BLs and WLs, which reduces bit-cell access energy compared to a single large cut. Also, the total surface area of the multi-tile architectures increases proportionally to the sum of each small memory periphery and the additional interconnect (small compared to the surface area of the peripheries). Moreover, in every architectures, only one memory cut is accessed at a time (*single active memory cut energy*) and the other memory cut are in idle energy mode (*memory idle energy*). Splitting a single cut architecture into a multiple cut architecture increases the SRAM's periphery in terms of surface area, but also the static leakage and idle energy related to the inactive memories. In terms of energy ratio, 98 % is consumed by the memories (active or inactive) and 2 % is consumed by the interconnect logic (*all other gates*).

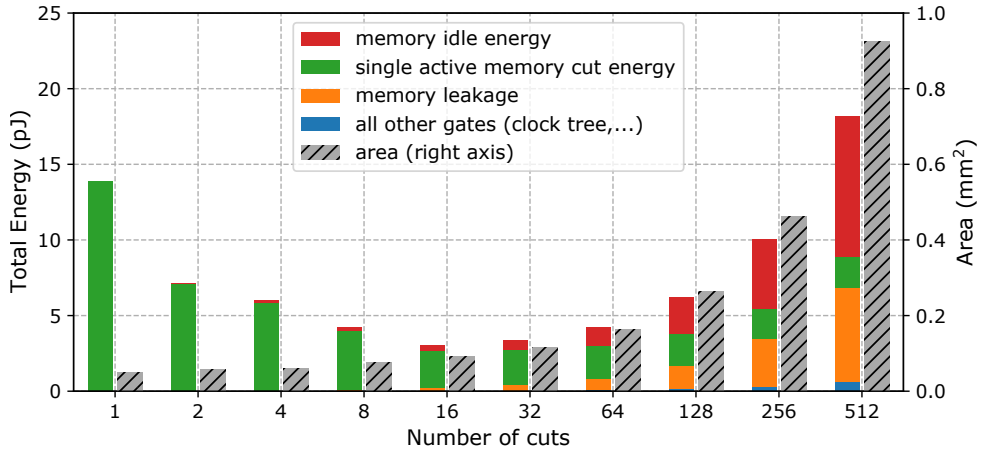


Figure 6.8: Total energy and area for a sweep of the number of cuts composing a 32-kB memory (lower is better). X-axis: number of cuts needed to obtain a memory size of a 32 kB.

In conclusion, for this 32-kB memory architecture, the best energy is provided for a tiling of 16 cuts while for performance it was 4 cuts. Hence, to highlight all trade-offs between performance and energy and design the suitable memory architecture, a wiring interconnect model is proposed in the next section.

6.3.2 Wiring Interconnect Model

To compare multi-tile design architecture trade-offs between them, I developed a wiring interconnect model according to the input design parameters. This model is based on a polynomial regression between the design parameters and the experimental results in order to estimate the timing performance, energy consumption and surface area (PPA) of a virtual multi-tile architecture. Figure 6.9 presents performance and energy estimates of the wiring model based on extracted timing and power analysis of 22 designs for various tiling memory configurations composed of various memory cut size.

The 32-kB memory is represented by the green curve implies designs ranging from the single tile to the multi-tile architectures. As discussed previously, the best energy point is achieved with a 16-cut architecture composed of 16 small memory cuts of 2 kB each. Thus, the performance is improved by 49% compare to a 128-cut architecture and the energy is reduced by 78% compared to a single tile. By analyzing the bottom left side of the figure, where the best performance and energy designs are achieved, it becomes clear that there is no optimal design that combines the best performance and energy in a unified architecture. Therefore, design and architectural choices must be done.

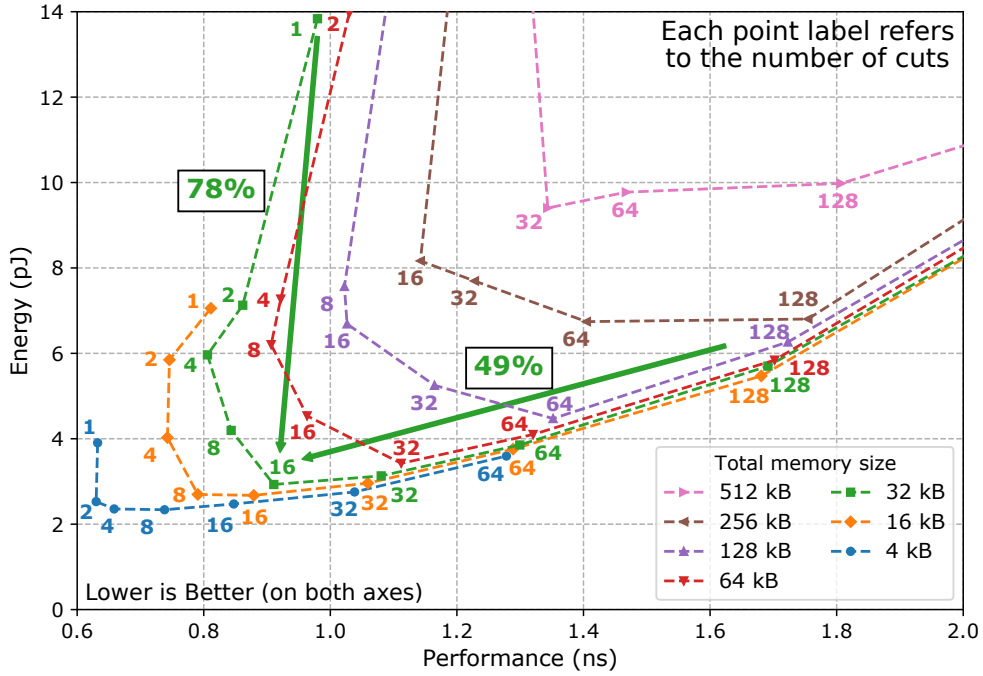


Figure 6.9: Multi-tile performance and energy trade-offs (lower is better on both axes). Each curve refers to the total memory size and each point label refers to the number of cuts composing the design.

Since the energy cost of the interconnect logic is low compared to the memories energy consumption (about 2%) and the timing cost depends of the number of tiles, this model is scalable for all types of memories (high-performance, high-density and low-power) according to a given tiling configuration. Furthermore, to explore other technological nodes, the wiring model must be calibrated with the corresponding founder's SRAM cut generator, as well as the standard cell library related to technological constraints.

6.4 Conclusion

In order to propose an optimized on-chip memory architecture for data-intensive applications, an accurate wiring interconnect model is required. The proposed model allows to size scalable memory architecture according to power, performances and area trade-offs, calibrated in a ST 28 nm FD-SOI technology node. By fragmenting a large memory into smaller memory tile, the memory architecture achieves a 49% performance improvement and reduces energy consumption by 78% at a cost of 1.8× the surface area. This model is a useful tool for circuit designers who want to have an accurate estimation of the wiring interconnect cost. However, there is no optimal multi-tile design, so the designer must choose the best trade-off between performance and energy consumption according to the architecture specifications and the application requirements.

Since the METEOR architecture brings performance gains through vectorization and parallelism proportional to the number of memory tiles, the interconnect cost becomes limiting when the number of tiles is excessive. Thus this model will be coupled with the system evaluation of the proposed architecture, as presented in Chapters 7 and 8. Furthermore, the vertical communication scheme proposed in Chapter 4, is not evaluated in this study, which would have an impact on the critical path during the physical design implementation flow.

Chapter 7

ArchSim: an IMC-NMC Software-Hardware Simulation Platform

Contents

Due to the trade-offs on the memory interconnect, detailed in the previous chapter, and the various implementations around the IMC and the NMC, it is relevant to provide a system view. In this chapter, I present a software and hardware simulation platform to evaluate these emerging concepts. The METEOR architecture and its programming model proposed in Chapters 4 and 5 are integrated into this proposed exploration platform and the simulation results produced are discussed in Chapter 8.

7.1 Introduction

In the area of emerging memories, a great diversity exists across the various types of technologies and innovative architectures provided. In order to compare all these architectures between them, it is convenient to use a high level and flexible simulator for architectural exploration, to find the sizing parameters and identify the trade-offs of each solution. Unfortunately, existing conventional simulators do not support these emerging technologies and complicates the integration and evaluation of new models into their ecosystem. Moreover, in the domain of near-memory computing, programming models and communication protocols between memory and computing components are often poorly detailed to describe a hardware implementation. To achieve these goals, our platform should evaluate the different ISA proposals, described in Section 5.2, at the hardware and software level. Lastly, it is necessary to define and size a scalable interconnect up to the micro-architectural level to respect the timing constraints between the small memory tiles according to the model developed in Chapter 6.

Table 7.1: Overview of virtual simulators and hardware platforms for architectural explorations.

	Gem5	Sniper	SystemC + TLM	LLVM	ZSim (Pin)	QEMU
ISA support, Intermediate Representation (IR)	x86, ARM, RISC-V	x86, RISC-V	any	virtual (IR)	x86	x86, ARM, RISC-V
Memory interfaces (interoperability)	point-to-point model	none	TLM sockets	none	none	callback functions
Signal (SA), Cycle (CA), Instruction (IA) Transaction (TA) Accuracy	CA	CA	IA/TA	IA	IA	IA
Development effort (integration complexity)	- -	- -	++	+++	++	-

Notes: memory interfaces referring to a conventional protocol in order to accurately simulate a data accesses (read or write).

In Table 7.1, I present a non-exhaustive overview of the different architectural simulators known and supported in our laboratory. The simulators must be able to model a high-level memory topology down to the micro-architectural level, in order to study interactions between memory and computing elements.

Previous works on IMC have used Low Level Virtual Machine (LLVM) as a high-level compiler tool chain for high-level performance profiling of IMC architectures [94]. Nevertheless, this software exploration platform lacks detailed modeling of instruction control flow and memory sizing related to application requirements. Multi-core simulators referenced as Gem5 [96] or Sniper [97] work well on standard architectures, using conventional protocols, when you have concise knowledge of the implementation and design parameters. However, cycle-accurate (CA) simulators can slow down simulation time and Gem5 is based on point-to-point memory modeling where memory accesses are simulated without modifying the data. ZSim [98] is a fast simulator based on Intel's Pin software [99] providing instruction instrumentation on x86 architecture to generate execution statistics. Three techniques are used to accelerate the simulation time, such as instruction-driven core models based on Dynamic Binary Translation (DBT), virtualization of the Operating System (OS) and ISA for complex workload support, and bound-weave algorithm, which is a custom protocol composed of two lock mechanisms to catch or release memory accesses. Unfortunately these accesses are also simulated without modifying the data. Similarly, QEMU [100] is a generic machine emulator and virtualizer that integrates an equivalent high-level abstract memory model and DBT-based core models to achieve fast simulation time.

These simulators are convenient for compilation purposes but lack of sharpness for architectural exploration. The SystemC [101] is a C++ library that provides high-level hardware representation as C++ objects and the Transaction-Level Modeling (TLM) [102] library provides communication mechanism abstractions to simulate any transactions between memory and computing elements. Thus, to provide a fast and flexible modeling platform, I developed a virtual simulation platform using SystemC/TLM abstraction, compliant with all limitations previously mentioned.

7.1.1 ArchSim Platform Overview

The ArchSim simulation platform is composed of ❶ a software layer, ❷ a hardware layer, ❸ an ISA layer and ❹ a simulation launchers, as shown in Figure 7.1. To evaluate the interoperability between hardware and software layers, the PyISAGen tool allows to generate a dedicated ISA communication interface used as an input parameter of the platform. To model hardware components, each distinct event (memory accesses, core instructions, ...) is calibrated according to hardware systems developed through a physical design flow (post P&R analysis for early explorations, circuit measurements for large-scale systems).

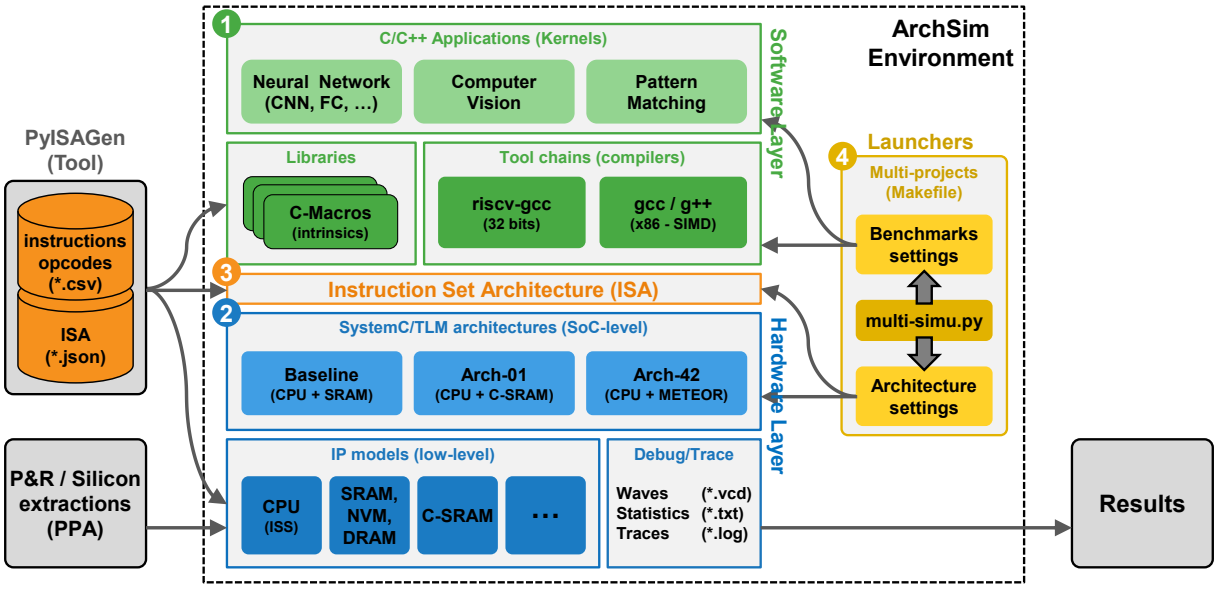


Figure 7.1: ArchSim: an hardware/software simulation platform.

The software layer is a collection of functional kernel applications that the user can compile and execute on different architectures, detailed in Section 7.2. The hardware layer simulates the architecture behaviors and memory transactions between hardware components, thanks to SystemC/TLM abstractions, and generate activity traces and synthesis reports regarding the power and the timing events, detailed in Section 7.3. The ISA layer is described respectively in the hardware and software layers of the platform using the input header files generated by the PyISAGen tool, detailed in Section 7.2.2. Finally, the simulation launchers provide a python cockpit for building and executing the platform with lists of compilation options, kernel input arguments and architectural parameters for the architectural space exploration. All results are formatted to capture as many details about the processor, memories and interconnects as possible.

7.2 Software Layer: a Macro Cross Compiler

To explore data-intensive applications on the METEOR architecture, the IMC/NMC ISA for scalable vector processing (see Section 5.1) is implemented in the ArchSim software and hardware layers, as shown in Figure 7.1. To avoid a misalignment of the communication between layers and components, the PyISAGen tool generates ISA translation files for a smooth integration on different targets (design, compiler, simulators, ...). At the software level, C macro files are generated to describe the instruction set and the communication protocol, such as the system bus integration and the control memory interface, as detailed in Section 5.2.2. When exploring on vector architectures (e.g. SIMD), these C macros are replaced by intrinsics (by *Intel* for x86) in order to evaluate the user program on different architectures without modifying the initial application user program.

This solution enables the native compilation, the execution of the program directly on the host machine (x86) and the functional verification of the kernel. The application kernels are written in C language and compiled with GCC 7.3.0 (c++17 standard). In addition, by using cross-compilers, the platform can generate a binary for other architectures (such as RISC-V), translated to Executable and Linkable Format (ELF), thus providing a bridge between the software and hardware layers.

7.2.1 Cross Compiler Tool Chain

A cross-compilation tool chain is the most often used for embedded development. It is typically compiled on the host architecture (x86) and generates code for the target architecture (ARM, MIPS, PowerPC or RISC-V). In Figure 7.2, the tool chain is composed of a compilation step to convert C files into object files and a linkage step to generate the ELF file. It partitions the binary into several memory segments (program, data, stack, C-SRAM control interface, ...) previously specified in the memory layout of the Linker Descriptor (LD) script. Various post-compile analysis tools provide information on the structure of the generated code, the size and occupied memory segment locations, up to the disassembled code for fine analysis.

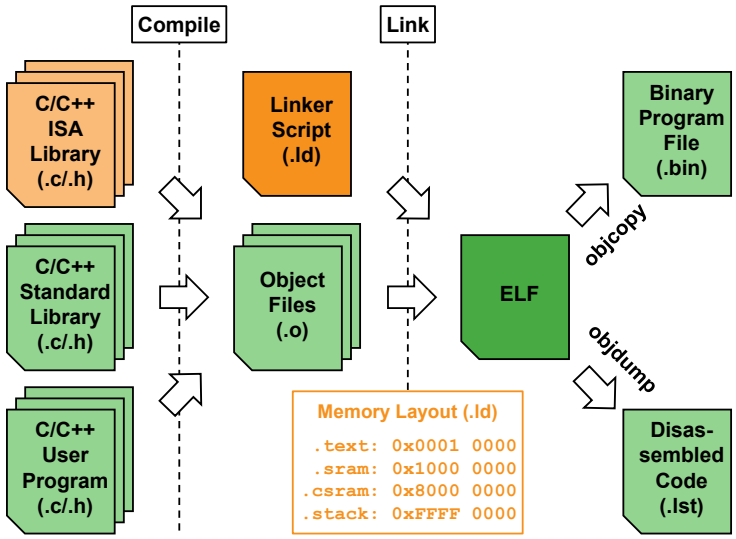


Figure 7.2: ArchSim software layer: the cross-compile tool chain.

As the software development on the RISC-V architecture and its RTL implementation is supported by the laboratory, we use its associated cross-compilation tool chain. Moreover, the RISC-V architecture environment is maintained by a strong community (industrial, academic, ...) oriented open-source and open-hardware. In order to evaluate the IMC and NMC approaches, specific libraries and memory mapping are embedded in this tool chain. In the Figure 7.2, the ISA library consists of C macro definitions and in-lined functions to exploit a vector format for scalable vector processing, involving IMC/NMC instructions for the METEOR architecture and vector intrinsics for SIMD architectures. Finally, the LD script file defines the memory map of: the program (`.text`), the data (`.sram`) and specific virtual (`.csram`) sections, as described in the Chapter 5.

7.2.2 ISA Integration Proposal, using PyISAGen

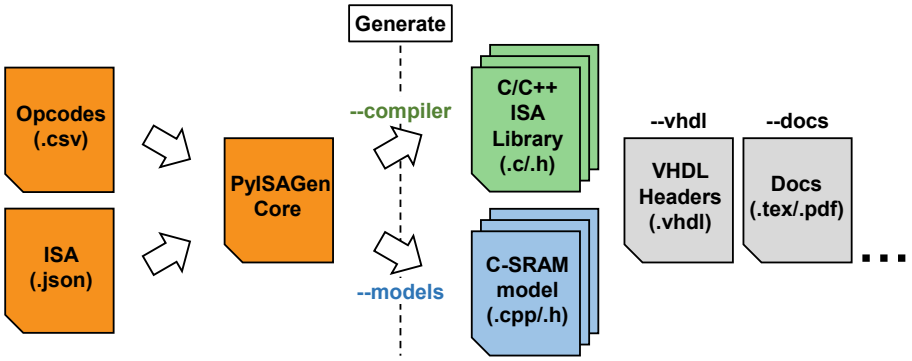


Figure 7.3: PyISAGen tool: a Python ISA generator tool.

The objective of the PyISAGen (for Python ISA Generator) tool is to explore and integrate into the ArchSim platform various ISA proposals (instruction set and communication protocol) while maintaining its integrity from the user program to the physical implementation. Thus, two input files are used, the ISA file (*.json) defines the bit field scheme of the communication protocol between the software and hardware layers and the Opcodes file (*.csv) defines the instruction set implemented in the IMC and NMC tiles, as shown in Figure 7.3. Depending on the options, the tool can generate: the C macro files for the software layer tool chain (`-compiler`), the physical model files written in SystemC/TLM for the hardware layer (`-models`), as well as the VHDL files for the physical implementation (`-vhdl`) and the specifications to be satisfied (`-docs`). This methodology helps to assess the benefit of the new instructions, the internal registers of the IMC/NMC tile or to minimize the footprint of the virtual memory section (`.csram`) for sending instructions. This tool, written in Python, accepts new options to keep consistency between new target files. Although this tool has been applied to the METEOR architecture, it is generic and can be adapted for further uses.

7.3 Hardware Layer: a Module-based Platform

The purpose of the hardware layer is to simulate the execution of the compiled program according to the physical behavior of the system. The system components (memory, core, interconnect, ...), also called modules, describe a behavioral model of the real system interacting with the input/output signals of the module. In order to reduce the simulation time, SystemC/TLM provides additional abstractions in the communication protocols between the modules. Contrary to SystemC, where the behavioral models are pin-level sensitive (signals, clocks,...),

elements interact by socket transactions between modules. To evaluate energy and power consumptions, the TLM Power library [103], developed in our laboratory, is included in the hardware layer allowing to annotate power values of all elements, including idle state and leakage values. Each module inherits a C++ class providing methods to update power states according to events, its configuration is detailed in Section 7.4.2.

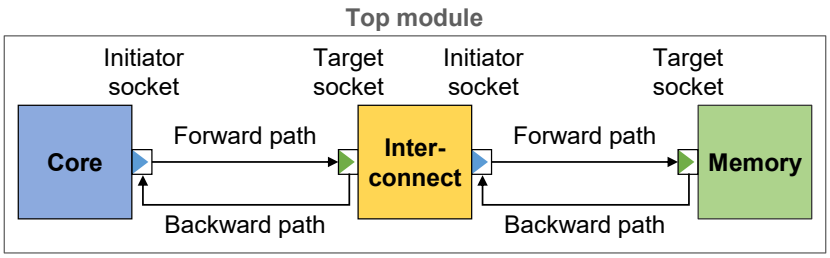


Figure 7.4: ArchSim hardware layer: basic example of a top module.

As shown in Figure 7.4, each system can be represented with a core module (initiator), a memory module (target) and an interconnect module (initiator and target) between them. The transaction is generated by the core module through the forward path whose content (payload) is modified by the interconnect module, if the transaction is valid it will be transferred to the memory module. With the Approximately-Timed (AT) protocol, the response of the memory module can be returned directly or delayed into the future with a new socket sent by the backward path, as detailed in 7.3.1. These transactions provide a performance metric for the system according to the accuracy of the chosen behavioral model (at the signal, cycle, instruction or event level).

7.3.1 Approximately-Timed Interconnect

In SystemC/TLM [102], "*the Approximately-Timed (AT) coding style is supported by the non-blocking transport interface, which is appropriate for architectural exploration and performance analysis*" (documentation). This coding style has a great versatility of communication protocol thanks to the forward and backward methods providing multiple phase and timing points during the lifetime of a transaction. Indeed, the target can respond with alternative requests via the backward path, suitable for the future implementation of a DMA controller. The non-blocking transport interfaces allow to manage multiple events coming from several initiators

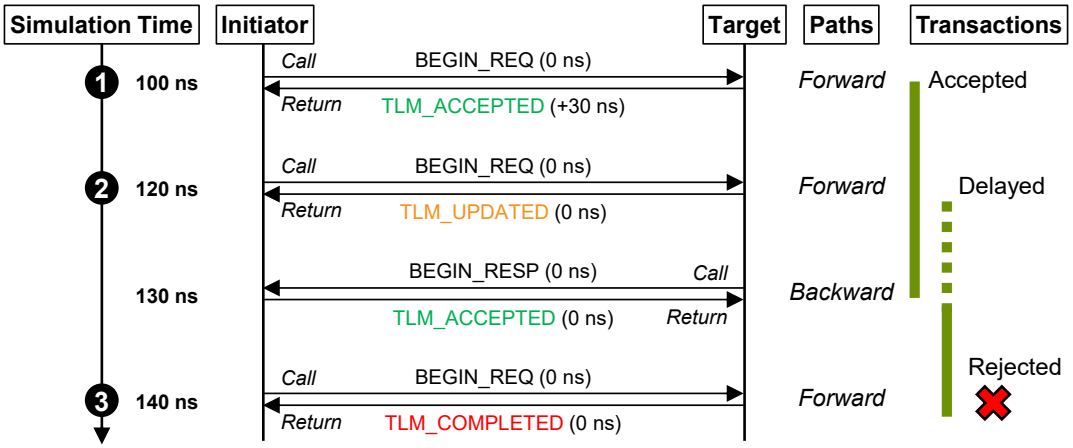


Figure 7.5: TLM message sequence chart using approximately-time coding style.

(cores, DMA, peripherals, ...). For the simulation of the METEOR architecture, presented in Chapter 5, the AT coding style is the most relevant for pipeline modeling in order to control instructions and memory conflicts.

The Figure 7.5 is a message sequence representation of four transactions between an initiator and a target. Request ❶ is sent by the initiator at the simulation time of 100 ns. During this lifetime the target accepts the transaction and starts its first process. Request ❷ is sent by the initiator at the simulation time of 120 ns and the transaction is updated by the target because a process is still running. When the first process is completed, the target can start the second process and notify its status by the backward path, accepted by the initiator. In request ❸, the transaction sent by the initiator at the simulation time of 140 ns is not valid (according to the model description) and the target returns a completed status corresponding to an error in our implementation. Moreover, transactions can support multiple phases (`BEGIN_REQ`, `END_REQ`, `BEGIN_RESP`, `END_RESP`) and the content of the transaction (payload), can be modified accordingly.

7.3.2 ISS-based Core Modules

Since the program is compiled with the cross-compiler, the binary ELF file can be interpreted by an Instruction Set Simulator (ISS) that "read" the instructions to be executed on the target processor without requiring the hardware core. It simulates the execution of the target processor pipeline and provides instruction by instruction the status of the internal registers, the stack pointer and the load/store accesses to the data memory. Any target architecture (ARM, MIPS, PowerPC, or RISCV) can be simulated on our host architecture (x86) according to their respective instruction set.

In the hardware layer, the core module consists of integrating a RISC-V ISS into a SystemC/TLM object in order to generate TLM socket for each memory access. These transactions will be dispatched to specific memory sections by an interconnect module according to the address contained in each payload of the transaction. Thus, I integrated the open-source RISC-V ISS written in C/C++ [104] and developed by *Western Digital* for the functional verification of the RTL design of their open-hardware RISC-V SweRV core. This methodology provides an instruction-accurate core module that is clocked at the actual frequency of the physical processor. Although the core is instruction-accurate, it is possible to stall its execution if a memory conflict occurs, thanks to the AT protocol which will cause a bubble in the processor pipeline. Therefore, the impact of memory conflicts can be assessed to evaluate the system in an cycle-accurate approach.

7.3.3 METEOR SystemC/TLM model

Thanks to the memory modules, ISS-based core module and the AT interconnect protocol provided by SystemC/TLM, the METEOR architecture model is assembled in the hardware layer of the ArchSim platform. Thus, each component is connected together with different TLM sockets, as shown in the Figure 7.6. According to the architecture specifications of Chapter 4, the complete system is composed of: (1) C-SRAMs: a matrix of IMC/NMC tiles considered as slave memory modules enabling computation inside each tile, (2) the TAM: a memory interconnect connecting the tiles for standard data access, distribution of IMC/NMC instructions among the tiles and transfer of operands between different tiles (such as the proposed VTU),

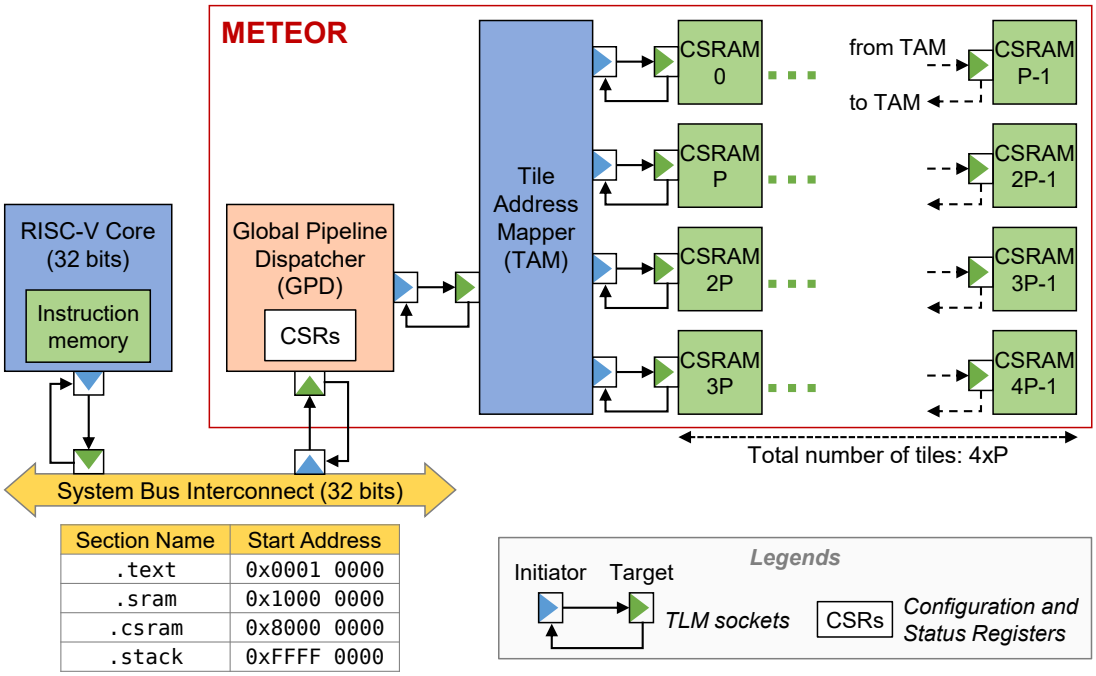


Figure 7.6: SytemC/TLM model of the METEOR architecture coupled to a RISC-V core.

(3) the GPD: a global pipeline controller to solve conflicts between IMC/NMC instructions and standard memory accesses, (4) the system bus: an interconnect system to dispatch TLM memory transactions between the processor and specific memory sections, (5) the RISC-V core: a core module to execute the user program and send IMC/NMC instructions to METEOR.

The reconfigurability and scalability of the METEOR architecture model is provided by the TAM. This TAM is an interconnect module that supports, the transfer of memory TLM transactions to Compute SRAM (C-SRAM) tiles, and dynamically broadcasts multiple IMC/NMC instruction transactions to tiles according to the computational vector size. Data memory modifications (when writing or computing) are performed by each C-SRAM tile module receiving a memory or instruction transaction, including when operand transfer is required between different tiles (because the memory is shared). For the GPD model implementation, each TLM transaction payload is evaluated according to the IMC/NMC instruction pipeline status and the vertical transfers between tiles, in order to accept the transaction to the TAM or to configure a layout configuration parameter through the CSRs, as proposed in the Section 5.1.2.

The system bus interconnect dispatches the TLM requests to the corresponding memory sections as defined in Section 5.2.2. For faster simulation and easier memory management, the program (`.text`) and the stack (`.stack`) memories are simulated by the ISS, thus no memory modules has been assigned to them. Nevertheless, their impact in terms of timing and power consumption are included in the overall system evaluation, as discussed in the next section.

7.4 Launchers and Performance Metrics

In order to maintain interoperability between the layers of the ArchSim platform, I developed a Python cockpit that drives the compilation and execution of user benchmarks, hardware models for scalar, SIMD and METEOR architectures, and extracts relevant performance metrics for software and architectural explorations, as presented in the Figure 7.1.

7.4.1 Cross-layer Simulation Launchers

As shown in Figure 7.7, the simulations are scheduled by; a hardware launcher that manages the compilation of emulated architecture using flexible hardware configurations (both static and dynamic) according to the architecture requirements, and a software launcher that manages compilation and execution on virtual or native platforms on the target architectures. These simulations are customizable by defining two exhaustive lists of parameters used for the hardware layer loop Ⓐ and the software layer loop Ⓑ. Finally, the results from the different platforms are formatted with a Python parser library to analyze the performance metrics, as detailed in the next section. This Python cockpit allows the evaluation of a large scope of hardware architectures and various application parameters (using input arguments) with the same user program. The data-intensive application benchmarks proposed in Section 8.1 are evaluated using these launchers for scalar, SIMD and METEOR architectures.

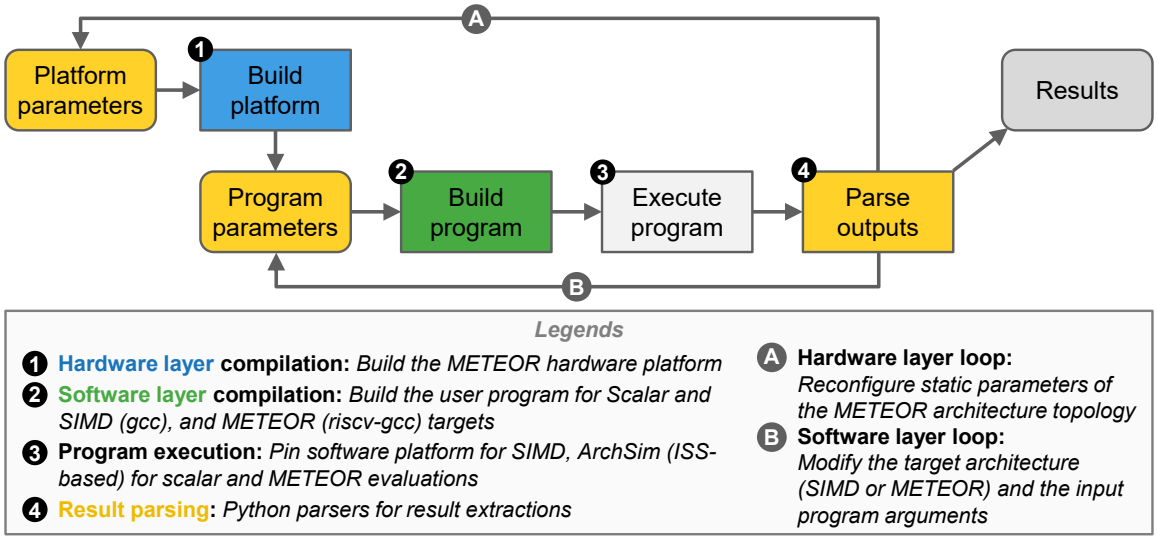


Figure 7.7: ArchSim cross-layer hardware and software simulation launchers.

Furthermore, the *Pin* software [99] is used to evaluate SIMD programs on *Intel*'s Xeon processors. This virtual platform is a dynamic binary instrumentation program developed by *Intel*. Thanks to the set of analysis tools provided, all instructions and memory accesses are detailed and sorted by data access width (from 32 bits up to 512 bits). Using this instruction level profiling, all standard memory accesses, SIMD and CPU instructions are aligned with the energy numbers of equivalent memories and RISC-V for timing and power evaluations.

7.4.2 Hardware Counters, Timings and Power Statistics

To quantify the program execution performance, each hardware module includes counters to measure the number of instructions executed, the number of memory accesses or the impact of processor stalls. All statistics are summarized in a file (*.txt) in order to explore and scale the architecture. These counters are enabled or disabled directly in the user program (software layer) using trigger variables (`tocounter`) to specifically evaluate a subset of the executed program, as shown in Figure 7.8. The `stdlib` counters used for kernel debugging (`printf`, ...) are ignored for the evaluation of the kernel execution time. The total kernel time is the sum of the number of executed instructions, the number of processor stalls, until the end (`tocounter=1`) of the kernel and the end of the memory internal processes (last write).

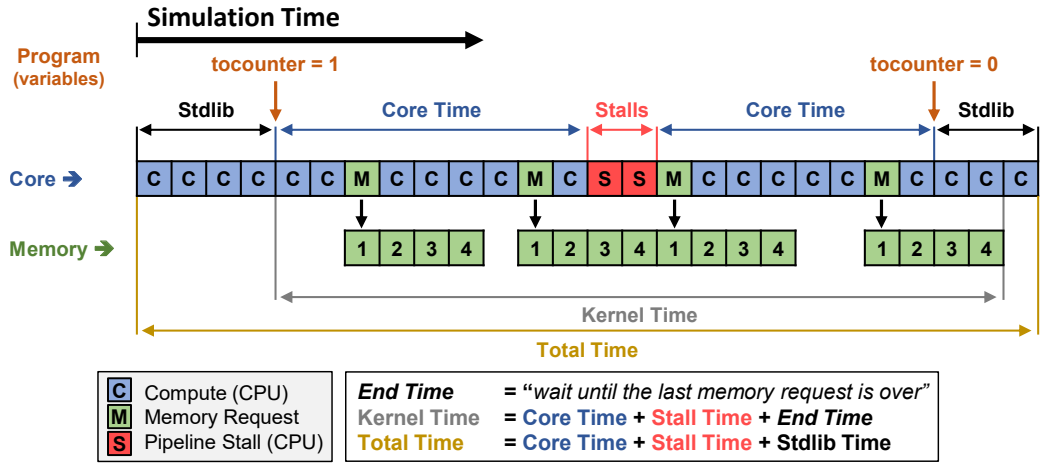


Figure 7.8: Module hardware counters for architecture explorations.

Including the TLM Power library [103] in the platform allows to assign power states to each hardware component according to the type of transaction. A configuration file defines multiple power modes (`ON`, `OFF`, ...) and power phases (`IDLE`, `COMPUTE`, `READ`, `WRITE`, ...) according to values measured on a physical circuit or a post P&R power analysis. This library produces several summary files (*.txt, *.vcd, ...) of kernel energy activity related to the execution time. As shown in Figure 7.9, the energy profiles of each module (`RISC-V`, `CSRAM-0`, ...) are simulated independently, improving the debugging and the accuracy of the architectural evaluation.

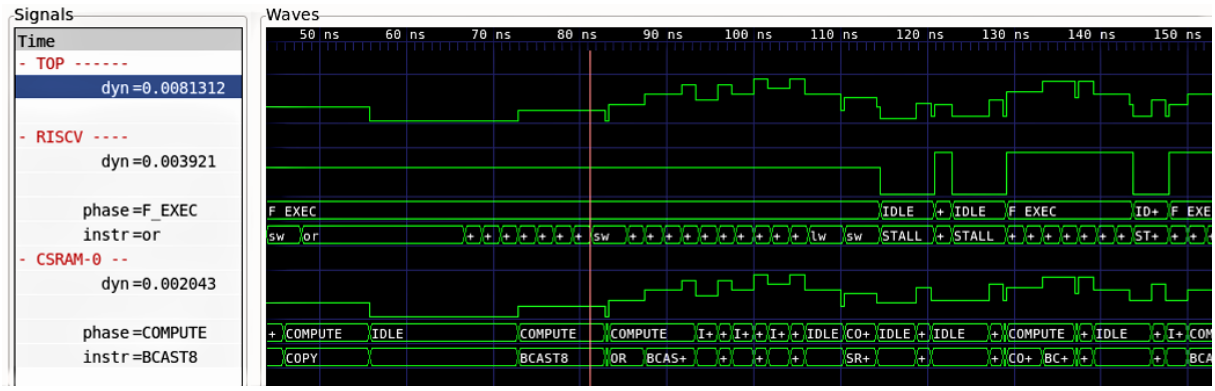


Figure 7.9: Example of VCD power profiles generated by the TLM Power library (GTKWave).

To conclude, the Python cockpit provides simulation monitoring, extraction and formatting of results to perform architectural space exploration across application benchmarks. Since the METEOR architecture model is driven at software and hardware level, the next section presents the calibration of these hardware modules.

7.5 Simulation Platform Calibration

For the exploration of the METEOR architecture (Chapter 4), integrating the NMC concept, the modules of the ArchSim platform are calibrated on circuit measurements and P&R design estimates. In the following sections we detail the calibration numbers for this example. This platform is now supported in the laboratory and in the team to explore other architectures integrating additional components (non-volatile memory, DMA, ...).

7.5.1 Single Core RISC-V System

As an applicative system, we catch all memory accesses of a 32-bit RISC-V core circuit related to an existing design exploration as in [105]. The energy numbers are adapted for different core frequencies in the *Global Foundries* (GF) 22 nm FD-SOI technology node for different processor events. The program memory is integrated into the core module and instruction fetch events are evaluated when a processor instruction is executed.

Table 7.2: RISC-V and its instruction memory numbers in 22 nm FDSOI used for simulations.

32-bit RISC-V core and SRAM instruction values						Comments
Frequency (MHz)	**60**	**120**	**240**	**480**	**720**	
Memory Leakage (µW)	34.56	50.40	73.50	106.40	155.40	Leakage (SRAM inst.)
Core Leakage (µW)	4.67	7.00	10.50	15.40	22.40	Leakage (Core)
Energy / instruction (pJ)	3.06	4.43	6.42	8.17	9.92	Compute
Fetch / instruction (pJ)	1.13	1.63	2.36	3.01	3.65	Fetch
NOP instruction (pJ)	1.85	2.68	3.88	4.94	6.00	Idle
Load instruction (pJ)	5.92	8.58	12.41	15.80	19.18	Fetch + Compute + Pop
Store instruction (pJ)	6.61	9.57	13.85	17.62	21.40	Fetch + Compute + Push

7.5.2 Memory Interconnect Model

Regarding the physical scalability of the multi-tile memory architecture, it is important to evaluate the wiring cost and the correct trade-off between memory tile size and performance. The memory interconnect model and its hardware post P&R analysis studied in Chapter 6 is used to size a larger memory composed of 4 kB tiles, as shown in Table 7.3. As presented in Figure 6.9, the model shows the scalability of multiple SRAM tiles: for a 256 kB of total memory size, composing an 8×8 array of 4 kB tile, the wiring cost between tiles is increased of 61 % in read access time and 42 % in read access dynamic energy, compared to a single 4-kB SRAM tile. Thus these performance and energy numbers are integrated in the TAM memory interconnect module of the ArchSim platform.

Table 7.3: Additional wiring cost of a 4-kB SRAM in a multi-tile architecture.

Total Memory Size (kB)	# of cut	Timing (%)	Dynamic Energy (%)	Leakage (×)
4	1	10 %	+0.1 %	1.02 ×
8	2	17 %	1 %	2.02 ×
16	4	25 %	3 %	4.02 ×
32	8	34 %	7 %	8.02 ×
64	16	43 %	14 %	16.02 ×
128	32	53 %	26 %	32.02 ×
256	64	61 %	42 %	64.02 ×
512	128	71 %	61 %	128.02 ×

7.5.3 In and Near Memory Computing RTL Simulations

The METEOR architecture is composed of C-SRAM tiles, each enabling both IMC logic operations and NMC arithmetic operations (`add`, `mul`). A single tile C-SRAM is designed by the team with the GF 22 nm FD-SOI technology node, implemented with different floor-plans configurations up to final P&R and studied in [90]. The C-SRAM has been optimized to process

energy-efficient vector data thanks to a additional digital wrapper decoding and executing the instructions send through the standard 32-bit access port of the memory tile. The computing capability of the C-SRAM performance impacts is presented in the Table 7.4, for a 4 kB SRAM configuration: the C-SRAM digital wrapper increases the memory tile size to 53 % in area and 73% in static power consumption. However, the read access energy only increases by 12 % on average, whatever the memory size of the tile. Finally, energy costs measurements related to each instruction are configured in the memory module, which receives by the same TLM port, standard memory accesses and computing instructions.

Table 7.4: C-SRAM design numbers versus a 2-Port SRAM (2RW) in GF 22 nm FDSOI.

SRAM (2-Port)				C-SRAM (versus baseline SRAM)			
Size (kB)	Leakage (mW)	Dynamic Energy (pJ)	Density (Mb/mm²)	Leakage (%)	Dynamic Energy (%)	Density (%)	Area (%)
2	0.31	12.04	0.50	+81 %	+14 %	-38 %	+62 %
4	0.34	13.10	0.87	+73 %	+13 %	-35 %	+53 %
8	0.41	14.42	1.35	+57 %	+11 %	-30 %	+42 %
16	0.61	16.10	1.56	+48 %	+11 %	-17 %	+20 %
32	1.03	18.38	1.69	+32 %	+11 %	-10 %	+11 %
64	1.56	22.80	2.17	+30 %	+10 %	-7 %	+8 %

7.6 Conclusion

The Archsim architectural exploration platform facilitates the system parameters study in order to size circuits before designing them. The hardware abstraction protocols provided by SystemC/TLM allow the modeling of physical components on multiple levels of accuracy (cycle-accurate up to transaction-accurate), that affects the simulation speed depending on the model behavior complexity. The software layer integrates and explores different configurations of the ISA proposed in this thesis, thanks to PyISAGen tool that maintains the integrity of the instruction set from the user program to the physical implementation. Coupled with a physical design flow, the platform evaluations can be used for functional validation and unit testing of the studied design. Today, this platform is reused in the laboratory to extend the IMC and NMC explorations into standard architectures in order to evaluate its performance capabilities at system level. The results of the performance simulations evaluated with ArchSim on the METEOR architecture are detailed in the following chapter.

Chapter 8

Architectural Exploration Results

Contents

In this chapter, I propose to explore the performance aspects, including the execution speed up, the energy reduction and the Energy Delay Product (EDP) gains of various kernels running onto different architectures. Thanks to the ArchSim platform, I can explore and compare various IMC and NMC architectures between each other with a set of vector-based kernel applications already compatible with existing SIMD engines. Early evaluations at the circuit level have already shown that a single IMC tile integrated into a conventional architecture achieves speed up gains of $1.8\times$ and energy reduction of $2\times$ compared to a 128-bit SIMD architecture [89]. In this chapter, I will extend the analysis to the NMC approach and to the multi-tile level with the METEOR architecture.

8.1 Application Kernels with Scalable Vectorization

In order to evaluate data-centric computing architectures, I propose to evaluate these architectures with small application kernels, easy to modify and vectorize. In fact, since there is no advanced compiler yet, it is necessary to control the compilation and execution parameters to understand the evaluation and the execution of the program.

Firstly, I converted two open-source programs, Hamming-Weight (`hw`) [22] and Shift-OR (`so`) [22], to support vector processing using specific variable types and vectorized loops. Secondly, I used a set of known applications validated by the polyhedral compilation community to study matrix, vector and scalar products, called *PolyBench* [95]. These kernels have the advantage of having been written in C language in a single file using macros to customize the types of variables, the variable declaration, the loop indices and the kernel instrumentation.

Thanks to this format, I was able to develop the same vectorized code for different compilation environments, such as ArchSim using the RISC-V compiler (version 7.2.0) and x86 Intel architecture using the C compiler (GCC version 7.4.0), all compiled at optimization level 3. For SIMD evaluation, vector intrinsics are forced by macros, which allow the main part of the program to run on 128-bit (SSE), 256-bit (AVX2) and 512-bit (AVX-512) SIMD engines with Intel's Xeon processors.

Table 8.1: Overview of studied kernels. The problem size parameters are assumed to take the same input value **n**, as presented in the *PolyBench* documentation [95].

Application Scope	Kernel Name	Operations	Memory Footprint	Memory Accesses	Description
DNA pattern searching	Shift-OR (so)	41n	n	2.5 %	Match a pattern in a DNA sequence
Information theory	Hamming Weight (hw)	17n	n	9.5 %	Count the number of differences after perform a XOR between two strings
Computer Vision	gesummv	$4n^2 + 3n$	$2n^2 + 3n$	21 %	Scalar, Vector and Matrix Multiplication as: y = alpha.A.x + beta.B.x
Numerical Computing	atax	$4n^2$	$n^2 + 3n$	38 %	Matrix Transpose and Vector Multiplication as: y = A^T (A.x)
	2mm	$5n^3 + n^2$	$4n^2$	26 %	2 Matrix Multiplications as: (alpha.A.B.C + beta.D)
	3mm	$6n^3$	$4n^2$	29 %	3 Matrix Multiplications as: ((A.B).(C.D))
Neural Network	gemm	$3n^3 + n^2$	$3n^2$	60 %	Matrix-multiply as: C = alpha.A.B + beta.C

Kernel Complexity: Linear time Quadratic time Cubic time
Kernel Category: (<15%) compute-bound or (>15%) memory-bound

The Table 8.1 summarizes the collection of benchmarks that we will study in our architectural explorations. Those seven kernels represent different application profiles in terms of memory patterns and computing requirement, extracted from operations in various application domains (linear algebra computations, image processing, statistics). The problem size parameters of each kernels are assumed to take the same input value (n). The kernel complexity is related to the number of nested loops that the kernel must execute (Operations), the memory footprint will limit the dataset size according to the vector width and the memory access ratio indicates whether the kernel will be limited by computation (compute-bound) or by the memory accesses (memory-bound).

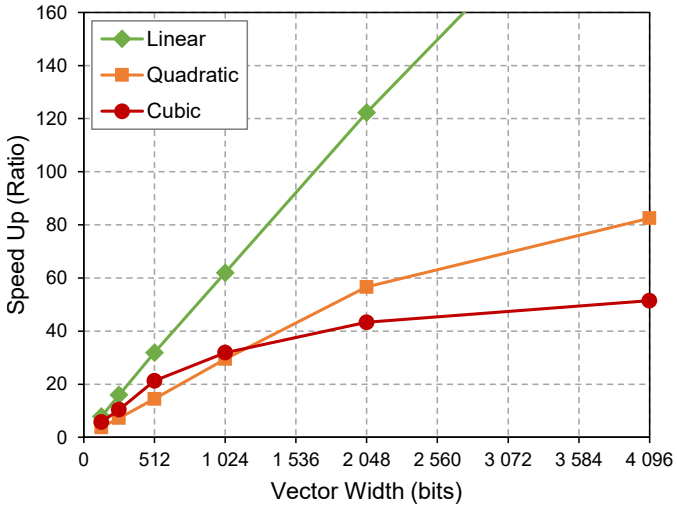


Figure 8.1: Speed up trends of linear, quadratic and cubic time complexity kernels according to the vector width (higher is better).

In order to represent the complexities of kernels (linear, quadratic and cubic), the Figure 8.1 illustrates the acceleration trends of the kernel complexity depending on the vector width. For the cubic complexity, the kernel increases its execution speed when increasing the size of the vector, but this acceleration reaches a plateau. Mainly because the number of loop interleaving reduces vectorization potential and increases data dependencies.

8.1.1 Architecture Benchmarking Set-up

In order to evaluate these kernels, I simulate three types of architecture as presented in Figure 8.2. In all three architectures a 32-bit CPU runs the user program which uses a 32-bit bus to access a 256 kB data memory used by the program. The CPU instruction memory and the stack memory are not shown, but are considered in terms of power consumption. The scalar architecture of Figure 8.2(a) runs the native kernel, used as a baseline in the exploration. As the kernels are vector-based, we used the 512-bit SIMD to compare the execution onto a vector

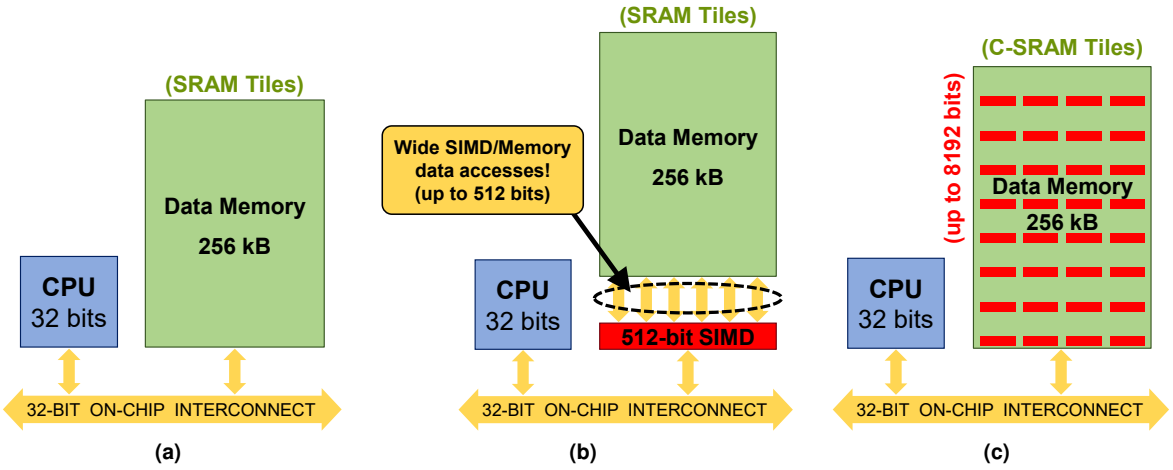


Figure 8.2: Architecture benchmarking set-up. **(a)** 32-bit scalar CPU, **(b)** 512-bit SIMD and **(c)** NMC architectures to explore. (a) and (b) are evaluated using ArchSim and (b) is evaluated using Pin software.

processor architecture (512 bits being the largest SIMD available). In a standard architecture, the SIMD is a hardware accelerator close to the processor and tightly coupled to the L1 cache memory using a 512-bit interconnect, as presented in the Figure 8.2(b). As described in Section 7.4.1, SIMD programs are simulated on the *Pin* platform running on native *Intel*'s Xeon processors. This instruction level profiling provides instruction and memory access counters for CPU and SIMD engines. Thus, 512-bit SIMD programs fetch only one instruction from the program memory but still access 16 words of 32 bits to perform the computation. All SIMD requests including instruction control flow and standard memory accesses are evaluated and energy numbers are aligned with scalar and METEOR architectures for timing and power evaluations. Finally, all the evaluated NMC architectures are considered as a compute memory matrix composed of multiple tiles of NMC using the METEOR architecture principles, as shown in Figure 8.2(c).

Within the scope of our exploration, a few assumptions are used:

1. The 32-bit scalar CPU and the 512-bit SIMD are evaluated according to an instruction-accurate method. Thus each instruction or memory accesses (load or store) require only one clock cycle to be executed. For the CPU architecture, the RISC-V Instruction Set Simulator (ISS) integrated in the ArchSim platform is an instruction-accurate simulator and the SIMD is evaluated with the instruction-accurate Pin software developed by Intel [99]. Due to the diversity of NMC technologies in the state-of-the-art from a system point of view (multiple cycle per operation), it is necessary to use a cycle-accurate platform in order to compare them with each other.
2. The horizontal reconfiguration of the NMC architectures is the same as proposed in the METEOR architecture described in the Chapter 4. In this exploration, for a total memory size of 256 kB, the set-up is a 4×16 array of 4 kB tile (size often used in the state-of-the-art) operating on a 128-bit vector, enabling horizontal scalability of 512-bit up to 8192-bit vector width, performance impacts are presented in Table 7.3.
3. In these architecture proposals, we investigate the system feasibility of such technologies, by sending basic instructions. The CPU has a controller function that can later be considered as part of an hardware accelerator (compute node). Eventually to be managed by a host CPU, which will request the execution of the kernel program on our compute node.
4. For all architectures, all data are pre-loaded in their 256 kB associated data memory. For this system exploration, we measure the gains without considering the data movement from an external (off-chip or on-chip) memory to our Data Memory. The purpose is to evaluate this embedded architecture as a compute node within a larger architecture as presented in Chapter 3. Off-chip memory accesses are still a concern, which will be addressed on a larger scale in future work.
5. The frequency of the system (core and interconnect) will be at 480 MHz in order to compare the performance impact of the pipeline in a multiple-tile architecture. This frequency is aligned with the different existing architectures listed in Table 8.2 and according to the interconnect constraints between tiles, using the wiring model described in Chapter 6.

Facing the large scope of IMC and NMC approaches according to the state-of-the-art, Table 8.2 summarizes the different architectures explored in our study. At the architectural level, bit-serial (`BS`) schemes process bit-by-bit operations to support more complex operations (e.g. floating-point) at a cost of higher latency (up to thousands of cycles for serial multiplier),

Table 8.2: Naming of NMC architectures used in our evaluation.

BS|BP	FC|CS	SV|DV	R?	Architecture Name	Comments
Bit-Serial	Full-Custom	Static Vector		**BS-FC-SV**	Multiple cycles per operations, no pipeline support [56]
Bit-Parallel	Full-Custom	Static Vector		**BP-FC-SV**	3-cycle operations, no pipeline support, shift-and-add algorithm for multiply operation (75 cycles) [89]
	CSRAM	Static Vector		**BP-CS-SV**	5-cycle per operations, no pipeline support, 8-bit digital multiplier [90]
			Register	**BP-CS-SV-R**	5-cycle per operations, pipeline support, 8-bit digital multiplier (operand forwarding, memory access savings) [90]
		Dynamic Vector		**BP-CS-DV**	5-cycle per operations, no pipeline support, 8-bit digital multiplier, horizontal grid reconfiguration [90]
			Register	**BP-CS-DV-R**	5-cycle per operations, pipeline support, 8-bit digital multiplier horizontal grid reconfiguration [90]

and bit-parallel (BP) schemes process larger vector data efficiently but remain limited to arithmetic operations for the moment (integer addition and multiplication). At the design level, implementations of the IMC approach are moving towards full-custom (FC) designs in order to modify the bit-cell array for both bit-serial [56] and bit-parallel [89] schemes, and the NMC approach supports foundry memory cuts for semi-custom C-SRAM (CS) design, maintaining a optimized and dense bit-cell array at the cost of a larger memory periphery area [90].

Thus, all these architectures are integrated and modeled in the ArchSim platform using the generic PyISAGen tool to adapt the various ISA at hardware and software level and calibrating the hardware models on the data extracted at circuit and post P&R level. Moreover, the NMC coupled with the METEOR architecture enables to explore the dynamic vector (DV) reconfiguration and the internal register (R?) due to their additional peripheral logic, while the IMC is evaluated with static vector (SV) only.

8.1.2 Impacts of Cycle Accuracy Effects

Comparing these architectures can be complicated if different simulators are used. With ArchSim, we can measure kernel execution times in a cycle-accurate and instruction-accurate method. These measurements allow to quantify the average Cycles Per Instruction (CPI) (of

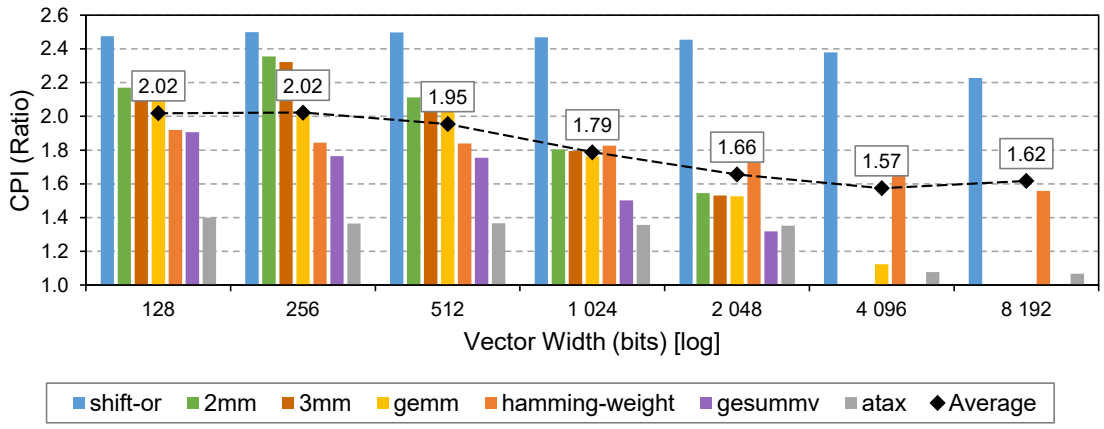


Figure 8.3: Cycles Per Instruction (CPI) ratio of the BP-CS-SV (without registers) by vector width. Not all kernels can be executed with a vector width of 8192 bits because of their memory footprint.

IMC/NMC instructions) during the program execution, which determines the average latency of the workload. When a IMC/NMC instruction has an execution time of one CPU clock cycle, then the CPI will be equal to one.

The evaluated NMC architecture (BP-CS-SV) does not have a pipeline to manage conflicts of NMC instructions coming from the CPU. Therefore, to avoid possible data conflicts, the system stalls the CPU, as presented in Section 4.1.3. This principle is standard, the memory returns a control signal through the system bus, this signal is received by the Load-Store Unit (LSU) of the processor. The control part of the processor inserts bubbles in its instruction pipeline while waiting for data availability. Each NMC instruction in this architecture operates on 5 clock cycles. In Figure 8.3, for a 128-bit vector size, the average latency of the program set execution has an average of 2 CPI, considering these conflicts. If we simulate this architecture with an instruction-accurate simulator, the acceleration gains will have an optimistic error that varies between 60% and 100%. Notice that not all applications can be evaluated with wide vectors because they are limited by the kernel's memory footprint.

To improve the performance and CPI at the system level, adding a pipeline for NMC instructions and an internal register to each memory tile is required. Indeed, the internal register enables to forward operands to subsequent NMC instructions, avoiding memory conflicts, reducing the CPU slowdown and saving memory energy accesses by reducing the read and write interactions, as explained in Section 4.2.1. In Figure 8.4, this BP-CS-SV-R architecture achieves an average of 1.15 CPI by increasing the NMC instruction pipeline throughput and reducing latency by 40% for a 128-bit vector size compared to the previous BP-CS-SV architecture.

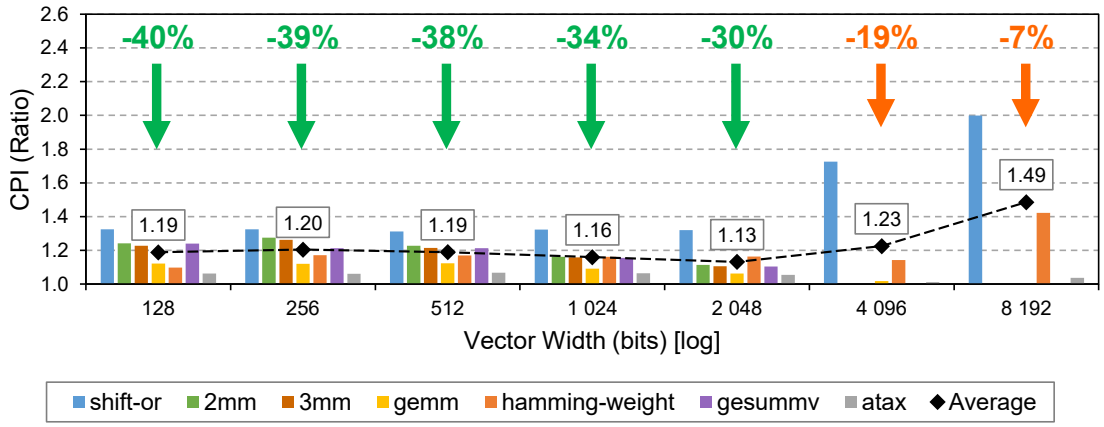


Figure 8.4: Cycles Per Instruction (CPI) ratio of the BP-CS-SV-R (with registers) by vector width.

Furthermore, if this architecture is simulated in an instruction-accurate simulator, the error of the speed up gains will be an average 19% with this approach. Which means that if we compare this architecture with instruction-accurate architectures (CPU and SIMD), the measured performance will be pessimistic by about 19%.

In the reconfigurable architecture, the number of addressable internal registers depends on the grid height of the METEOR architecture, as defined in Section 5.1.2. The smaller the vector, the greater the number of internal registers available and vice versa. The 4096-bit and 8192-bit vectors have 2 and 1 register respectively. Hamming-weight and shit-or kernels would require at least 4 registers, which explains the increase in CPI average curve for large vectors.

8.2 Architecture Benchmarking

With a multi-tile architecture, the IMC and NMC approaches provide a strong Data-Level Parallelism (DLP) by exploiting a scalable vector programming model, as discussed in the Section 5.3. Moreover, the METEOR architecture enables dynamic vector reconfiguration during program execution by using the reduction-operation micro-code to enhance connectivity between tiles and reducing traffic between processor and memory, as presented in Section 5.4.2. In this section, the vector scalability and the reconfigurability of IMC and NMC approaches are evaluated on the METEOR architecture by exploring linear, quadratic and cubic vector-based kernels, as listed in Table 8.1.

8.2.1 Evaluation of the Vector Width Scalability

In this first experiment, we compare four IMC/NMC and 512-bit SIMD architectures to the 32-bit scalar CPU baseline by increasing the vector size of IMC/NMC architectures. As shown in Figure 8.5(a), kernels of linear complexity (here `shift-or`) align perfectly according to the vector size. For a 8192-bit vector size, the BP-FC-SV (IMC full-custom) achieves better speed up performance than the BP-CS-SV-R (NMC with register) due to higher clock cycle latency and a single register to store intermediate results. Despite this, with a 2048-bit vector size, the total energy reduction is 2.9× better for BP-CS-SV-R compared to BP-FC-SV due to the founder's memory design, as shown in Figure 8.5(b).

Furthermore, the bit-serial approach follows the same trend, but multi-cycle operations have a significant impact on overall program acceleration. For large workloads (with wide vector size), this scheme achieves an energy efficiency of 0.56 TOPS/W for 8-bit multiplication [51]. Compared to the 512-bit SIMD architecture, all bit-parallel architectures achieve better speed up performance and energy reduction for a vector size larger than 1024 bits, and 256 bits for the BP-CS-SV-R architecture.

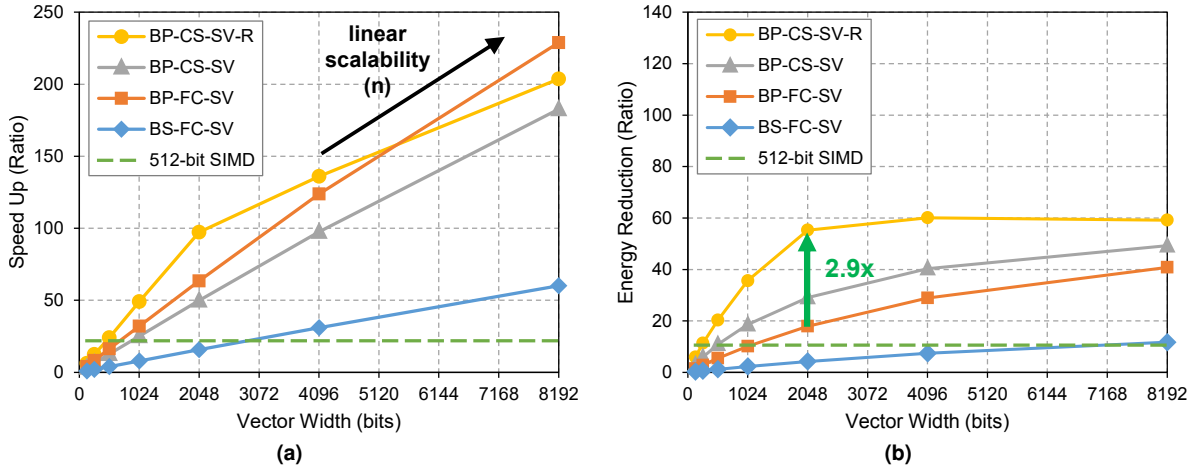


Figure 8.5: Vector width impacts on a linear time complexity kernel (shift-or) (higher is better). **(a)** execution speed up and **(b)** energy reduction gains of the shift-or kernel for NMC and 512-bit SIMD architectures compared to the 32-bit scalar CPU architecture.

In this second experiment, the kernel of quadratic complexity (here `atax`) uses more arithmetic operations, such as additions and multiplications, nested in two loops. For full-custom

architectures, multiplication is an operation performed in several clock cycles in order to avoid the significant addition of a multiplier close to memory. These multipliers add a cost in surface area and static power consumption, but achieves better performance compared to scalar and 512-bit SIMD architectures, as shown in 8.6(a). However, This operation highly increases the energy consumption compared to the baseline architectures, as shown in Figure 8.6(b). Thus, NMC architectures achieve better performance and energy gains compared to IMC architectures because of the reduced IMC instruction set. Thus, the performance benefit of NMC architectures is significant despite the larger memory surface area (and also energy).

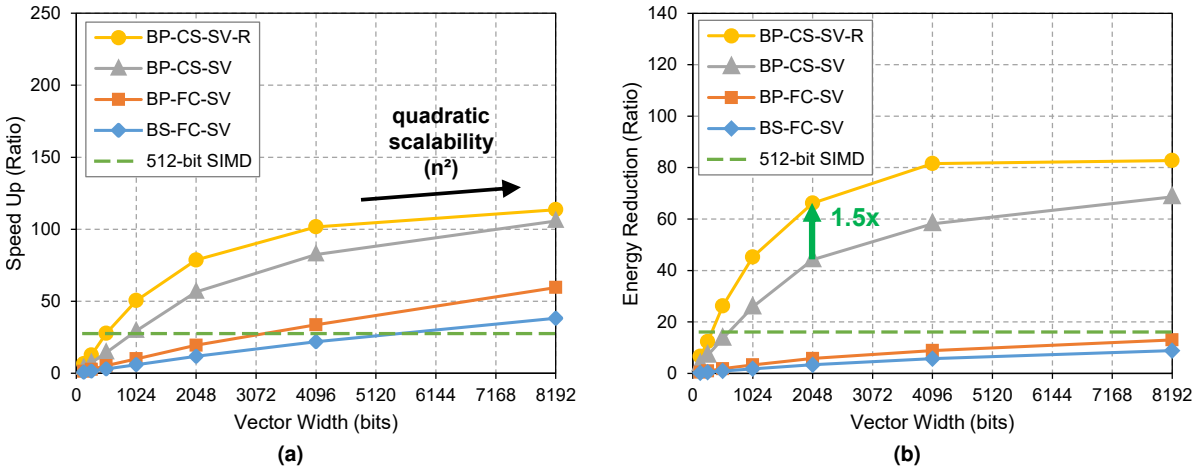


Figure 8.6: Vector width impacts on a quadratic time complexity kernel (atax) (higher is better). **(a)** execution speed up and **(b)** energy reduction gains of the atax kernel for NMC and 512-bit SIMD architectures compared to the 32-bit scalar CPU architecture.

Moreover, the internal register improves performance and CPI, as shown in Section 8.1.2, but also reduces energy consumption by 1.5×, as shown in Figure 8.5(b). According to the results, the data parallelism of kernels is improved as the vector size increases. The internal register has a significant impact both to speed up kernel execution and to reduce the energy cost of the hardware. Furthermore, even if the digital multiplier is energy-intensive, it provides system level gains that justify its implementation.

8.2.2 Evaluation of the Dynamic Reconfiguration

Since the METEOR architecture enables dynamic reconfiguration between vectors inside the memory, internal operations are possible between sub-vectors and between different NMC tiles during the program execution. Thus, this architecture supports the reduction operation micro-code which benefits of various architectural features, such as internal transfer communication and operand forwarding to achieve high throughput, as described in Section 5.4.2.

As shown in Figure 8.7, this operation impact the quadratic kernel. Indeed, for large vectors, the program must perform a high number of sequential memory accesses to compute the reduced sum in the CPU, which explains the speed up trend to reach a plateau. The proposed reduction operation reduces this part of the program by a logarithmic factor. It changes from a factor n, corresponding to sequential accesses, to a factor $log\ n$. Thus, the internal loop of the kernel is replaced from quadratic complexity (n^2) to linearithmic complexity ($n\ log\ n$). This optimization works for very large vectors, but the additional instructions reduces the

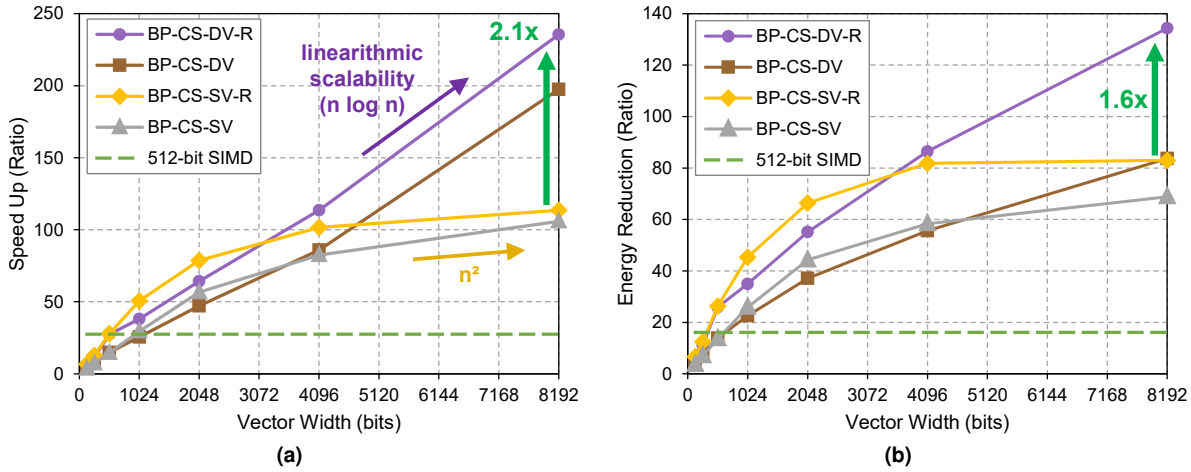


Figure 8.7: Dynamic vector reconfiguration impact on the atax kernel (quadratic complexity). **(a)** speed up gains and **(b)** energy reduction gains compared to the 32-bit scalar CPU architecture.

performance for medium vector sizes (around of 2048 bits). Using the dynamic reconfiguration (BP-CS-DV-R) improves performance by 2.1× and reduces energy consumption by 1.6× compared to a static configuration (BP-CS-SV-R).

However, the maximum vector size used in cubic kernel (here gemm) is limited by the kernel memory footprint. Indeed, it is not possible to increase the vector width (in the inner loop) relative to the available memory contained in the data Memory. To adapt our architecture to our kernel, we can either (1) increase the memory of each tile composing the memory matrix, or (2) reduce the size of the total (maximum) vector. As shown in Figure 8.8, the dashed curves are measured for a larger memory size (512 kB), allowing the use of the 8192-bit vectors.

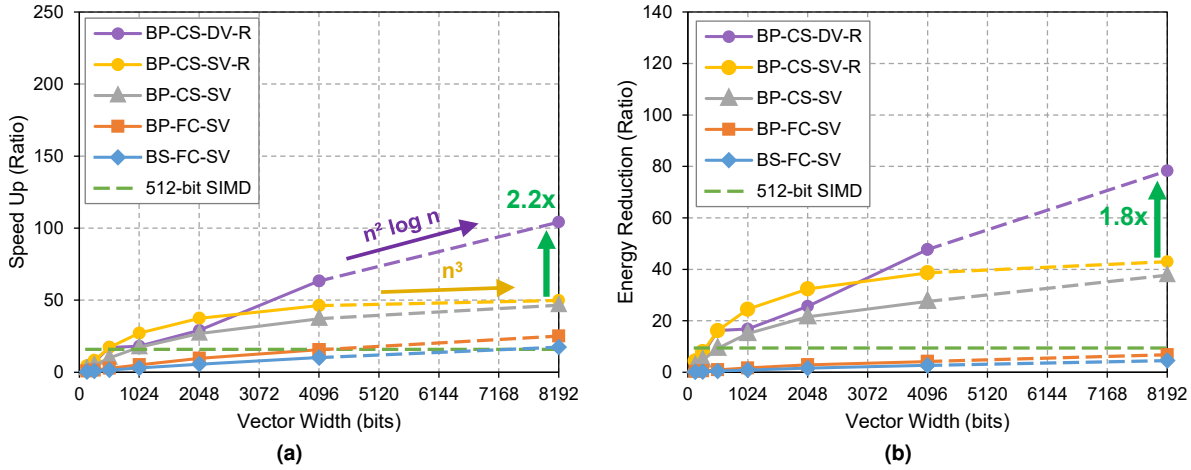


Figure 8.8: Dynamic vector reconfiguration impact on the gemm kernel (cubic complexity). **(a)** speed up gains and **(b)** energy reduction gains compared to the 32-bit scalar CPU architecture.

Thus, to increase performance of algorithm with high operation complexity on the METEOR architecture, it is required to extend the connectivity between tiles in order to compute on very large vectors. The overall performance and energy gains are related to the inter-tile communication scheme and the fast architecture reconfiguration to support an efficient scalable vector processing.

8.2.3 Simulation Results

Figure 8.9 represents the performance extractions of seven kernels using the best vector width (8192-bit when possible) of each architecture according to the memory footprint of their data. For a complete overview between vector architectures with each other, the Figures 8.9 and 8.10 use the 512-bit SIMD architecture as a comparison baseline. All results greater than one are therefore better than the 512-bit SIMD architecture.

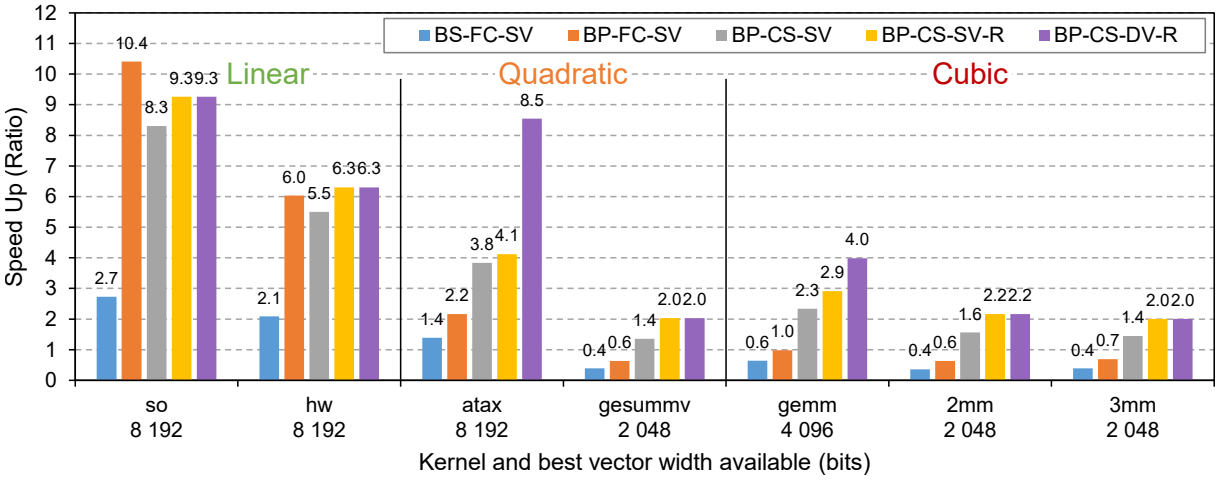


Figure 8.9: Execution speed up of NMC architectures compared to the 512-bit SIMD architecture.

In these figures, all NMC architectures increase its speed up and energy reduction gains using large vectors, larger than 512 bits instead of the SIMD architecture, limited to 512 bits. Kernels using the reduction operation (atax and gemm) benefit from the dynamic vector reconfiguration. For linear kernels (shift-or and hamming-weight) that do not have reduction operations, the dynamic vector architecture provides the same performance and energy savings as static vector architectures. As the memory footprint has an impact the cubic kernels, the vector size limits the performance of the whole system, as discussed in the previous section.

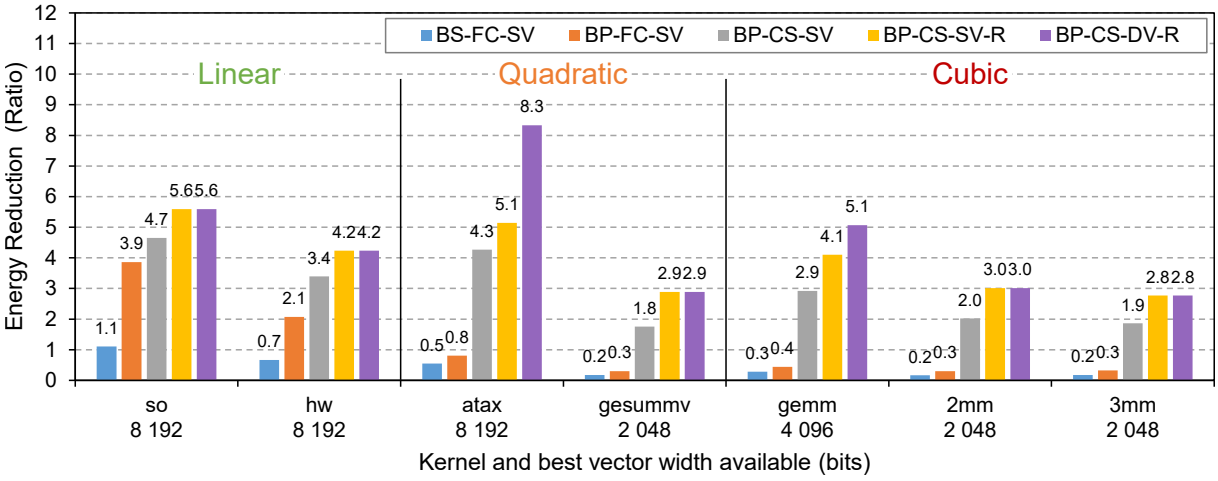


Figure 8.10: Energy reduction gains of NMC architectures compared to the 512-bit SIMD architecture.

Although the speed up depends on the vector width, the energy reduction gains are comparable between the different NMC categories (linear, quadratic and cubic) as presented in Figure 8.10. Overall, full-custom architectures exhibit excessive power consumption due to

the modification of their memory bit-cell array, which reduces performance at the system level. Dynamic vector reconfiguration increases connectivity and reduces the number of CPU memory requests, highlighting the capability of NMC architectures to reduce data movement.

Table 8.3: Summary of best NMC architecture results compared to 512-bit SIMD architecture

Kernel Family	Speed Up			Energy Reduction			Energy Delay Product		
	W_{512}	W_{2048}	W_{Best}	W_{512}	W_{2048}	W_{Best}	W_{512}	W_{2048}	W_{Best}
Linear	0.8	3.7	9.3	1.1	4.3	5.6	0.9	15.8	51.8
Quadratic	0.9	2.4	8.5	1.4	3.5	8.3	1.4	8.6	71.2
Cubic	0.9	2.2	4.0	1.4	3.1	5.1	1.3	6.7	20.2

W_{512} ➔ vector width of 512 bits
W_{2048} ➔ vector width of 2048 bits
W_{Best} ➔ the largest vector width (up to 8192 bits)

The Table 8.3 summarizes the speed up gains, energy reduction and Energy Delay Product (EDP) reduction of NMC architectures (BP-CS-SV-R and BP-CS-DV-R) compared to the 512-bit SIMD architecture. For kernels with linear complexity, an EDP reduction of 52× is achieved, while for kernels with quadratic complexity, where we increase the internal connectivity, an EDP reduction of 71× is achieved compared to 512-bit SIMD architecture. The growing complexity of kernels requires better connectivity, which significantly increases power consumption regardless of the execution time. Indeed, the energy reduction gains of cubic complexity kernels are better than their speed up execution factor.

8.3 Discussions

The METEOR architecture allows wide vector processing for data-intensive applications by enabling dynamic vector reconfiguration within the memory architecture at the software and hardware levels. In this section, I present some perspectives and opportunities to consider at a higher system level.

8.3.1 Efficient Data Placement

The data placement in our data Memory has a major effect on the execution of the program related to the vector acceleration. Structuring the data as vectors allows us to manipulate (and move) the data in our tiled architecture while maintaining memory continuity. C language provides mechanisms (attributes) to ensure the memory alignment of vector data types. However, this language support is not applied automatically by compilers. In addition, the programs produce intermediate results, often stored in the processor registers in the standard architectures. Every proposed kernels need a memory buffer (relative to the vector size) to store all intermediate results. Of course, internal registers can assume this role in some vector configurations, as presented previously. But for large vectors and compute bound kernel (rich in intermediate results), it is necessary to reserve a large memory section to keep a correct acceleration factor.

Nevertheless, data duplication or "memory overhead" is common in High Performance Computing (HPC) applications and especially for neural network algorithms in order to keep high

speed up gains. Indeed, all known deep neural network frameworks include functions to arrange in a vector-format the data (im2col) to optimize the data parallelism of neural network layers [20]. In future work, the exploration of these solutions applied to our architecture is a promising approach.

8.3.2 Memory Allocation

By studying the feasibility of the software environment, we have explored applications using only static data allocation, where the size of the memory positions is determined at the compilation stage. This methodology is sufficient for many applications, including the majority of applications using neural networks, because the topology and memory sizes of each layer are known and fixed after the compilation. Dynamic allocation requires additional mechanisms often linked to an Operating System (OS), but necessary in dynamic movement between data in order to optimize the available memory space. These mechanisms have to be evaluated in a future work as well as the space required for the program memory whose size also depends on the complexity of these mechanisms.

8.4 Conclusion

This architectural exploration study provide key features of the scalable and reconfigurable METEOR architecture composed of multiple NMC tiles. By evaluating kernels for data-intensive applications, the METEOR architecture achieves an EDP reduction of 52× for linear kernel and 71× for quadratic kernel compared to a 512-bit SIMD architecture. Thanks to the internal register integrated in each NMC tile, pipeline throughput is increased due to operand forwarding mechanism, many memory accesses are saved, and data transfers between different tiles are more efficient. The register reduces the latency of the whole architecture by 40%, and reduce energy consumption by 1.5× compared to the same architecture without register. By combining the internal register and the dynamic reconfiguration, the architecture continues to improve performance of quadratic and cubic kernels, using large in-memory vector processing in order to reduce data movement between processor and memory. At application level, the NMC approach delivers better performance due to the enhanced instruction set and intermediate values scheduling compared to full-custom IMC approach.

General Conclusion

Due to the increasing memory needs of modern big data applications, conventional architectures are becoming limited in order to overcome massive data movements in their memory hierarchy, also known as the memory wall problem. The traditional memory cache is no longer energy-efficient enough to satisfy the memory requirements, with a lack of flexibility and scalability caused by the data placement policies and limited connectivity. Moreover, these data-intensive applications presents little locality in computing, and reduced data reuse with very large datasets, for which standard memory hierarchy are inefficient. Emerging energy-efficient data-centric architectures based on advanced technologies are becoming more and more necessary to break down the memory wall.

Therefore, the envisioned architecture based on advanced technologies is proposed to explore emerging research topics and opportunities. Many challenges still remain to be solved, such as advanced 3D-stacked fine-grain integration, efficient thermal dissipation, high-density and high-endurance on-chip emerging non-volatile memory, computation immersed within memory, distributed and reconfigurable architecture and general-purpose programming model. As a first step towards these directions, the contributions of my PhD thesis addresses the exploration of a reconfigurable and scalable tiles of near-memory computing (NMC) architecture for data-intensive applications, presented as METEOR. Each tile enables arithmetic and logic operations within the memory to reduce data movement on the system bus. Moreover, this architecture combines horizontal scalability and vertical communication reconfigurability between tiles to support scalable vector processing in order to compute at the best vector size for maximum performance. These schemes are configurable and programmable by software to resize the memory tiling at run-time and send instructions to the tiles composing the vector. Thus, the same scalable vector programming model is used to explore data-intensive kernels available on existing SIMD engines onto the METEOR architecture.

Data-intensive applications, such as neural networks, machine learning, image processing and database searching, mostly involve vector/matrix operations, reduction operations and specific data pattern accesses. At low-level software integration, I implemented a dedicated ISA through the standard memory interface in order to support NMC instructions in a standard compilation tool chain. Although it is a set of C macros statically adapted at compile time, such proposal provides the ability to interleave processor and memory instructions in the same instruction flow to program vector instructions at any vector size and vector alignment. The same vector-based program proposed for the NMC is compatible for conventional SIMD unis and CPU. This software compatibility is an important feature for adapting the proposed architecture, while allowing easier benchmarking for architecture performance exploration. This study highlights the data movement choreography between NMC tiles, necessary to main-

tain high bandwidth and reduce processor interaction. These mechanisms extend the total distributed memory-computing capacity: using configurability and tile transfers with smaller memory tiles to preserve local performance (frequency and energy), while increasing the overall memory capacity, and enabling computation within tiles and between tiles.

Thus, the proposed reconfigurable METEOR architecture would benefit from this scalable vector processing model. By assembling these NMC tiles horizontally, the same NMC instruction is distributed to the tiles where the data vectors are stored, thus performing vector processing. The combination of a horizontal scalability scheme and a vertical data communication reconfigurability offers an flexible vector size for maximum computing performance and enhanced inter-tile connectivity. In addition, the software integration provides specific processor instructions to configure the METEOR layout to enable computations between distant vectors, while the intermediate vector transfers are supported at the hardware level. Moreover, this architecture is able to interleave standard processor memory accesses and the vertical NMC instruction flow while avoiding data conflicts. Thus, METEOR can be integrated as a standard on-chip high-speed memory and as a vector co-processing unit into any conventional architecture, thanks to its standard memory interface. The hardware memory layout is developed in a scalable fashion, however, additional design challenges remain to ensure an adequate memory architecture scalability to design larger memories. The standard interconnect and the additional vertical communication are two hardware constraints to evaluate in order to estimate the real performance and energy of the METEOR architecture.

Providing a large on-chip memory is a key features of data-centric architecture in order to minimize the off-chip memory wall impact. Large memories are made up of smaller memory tiles to suit the technological trade-offs of performance, power and area (PPA) according to the required specifications. To size the METEOR architecture, I developed an accurate wiring interconnect model to estimate PPA of multiple tile designs based on RTL implementations through a physical design flow, calibrated in a ST 28 nm FD-SOI technology node. The design space evaluation evaluates the scalability of multi-tile memory architectures for various memory configurations and memory sizes in order to estimate the impact of the memory interconnect cost. Thus, by fragmenting a single 32 kB memory into 16 smaller memory tiles, the architecture achieves a 49% performance improvement and reduces energy consumption by 78% at a cost of 1.8$\times$ the surface area. This wiring model is implemented and applied to the METEOR architecture for larger tile configurations. However, as the METEOR architecture provides performance gains through large vectorization and high parallelism proportional to the number of memory tiles, the interconnect cost becomes limiting in access time and energy consumption when the number of tiles is excessive. Since the logical interconnect depends on the timing performance, it becomes necessary to evaluate memory architectures with 3D vertical interconnect by developing an interconnect model for 3D technologies.

Due to the development and design time of a comprehensive architecture, the maturity of the technologies used and the lack of programming model, I developed ArchSim, a hardware and software simulator to evaluate the METEOR architecture running data-intensive applications. ArchSim facilitates the architectural space exploration through three main layers. Firstly, the hardware layer provides an SystemC/TLM abstraction protocol to allow the modeling of physical interconnect, memory and processor units on multiple levels of accuracy. The proposed memory wiring model, NMC and CPU design implementations are integrated as hardware models of the METEOR architecture. Secondly, the software layer integrates data-intensive

benchmarks with scalar and SIMD compatibility and various ISA implementations, generated with PyISAGen, an automatic tool to maintains the integrity of the instruction set development from specification to implementation. Finally, thanks to a Python workflow cockpit, this platform is reused in the laboratory to explore the IMC, NMC and PIM approaches into standard architectures to evaluate performance capabilities at system and software levels.

The architectural exploration study of METEOR under the ArchSim simulator evaluates the impact of the proposed scalable vector processing, involving the specific reduction operation. Contrary to SIMD systems, the METEOR architecture still continue to increase its vector size and reduces the CPU memory accesses thanks to the internal vertical data communication between tiles and the reduction operation functionality. Thus, most of the system bus data movements between processor and memory are avoided for vector-matrix and matrix-matrix multiplication used in linear, quadratic and cubic complexity kernels. The METEOR architecture improves the execution speed up by 200× for linear kernel, 240× for quadratic kernel, and 65× for cubic kernel compared to a scalar CPU architecture. By considering a 512-bit SIMD accelerator with high-speed and large memory access, the METEOR architecture still achieves an EDP reduction of 52× for linear kernel, 71× for quadratic kernel, and 20× for cubic kernel. In conclusion of this study, the performance limitations of the cubic kernel are mainly due to their large memory footprint implemented on a too small METEOR architecture. To solve this problem, an architectural and technological 3D approach is required to increase memory capacity and connectivity between tiles. The METEOR architecture has lead to a patent, and the detailed architecture exploration has been published in several international conferences [84, 88]. As perspective of future work, it is planned to implement the proposed METEOR architecture, validate and measure in details the hardware and software performance in a test-chip.

Toward a 3D Architecture

We have evaluated the advantage of providing an additional and reconfigurable wide interconnect in a multiple memory tile system that can perform near-memory computation. Between the standard distribution interconnect to access data from the processor and this proposed vertical data communication, there are paradoxical constraints. Indeed, as lots of tiles are required to achieve a high-level of data parallelism but limit the standard interconnect performance (for best latency and energy), the 3D-stacked architecture can provide an optimized standard interconnect on 2D layer while 3D vertical connectivity is well suited for large vector processing. As proposed in the Chapter 3, 3D-stacked technologies would solve this problem, by bridging the connectivity gap between memory and computing while still maintaining flexibility and scalability of the architecture.

With our 2D reconfiguration architectural proposal, the implementation of the memory interconnect (TDI) is designed to the optimal performance on 2D layer, while the dynamic vertical transfer interconnect (VTI) is implemented in 3D across 2D layers, as shown in Figure 9. The 3D METEOR architecture allows to organize the memory hierarchy by layer while providing a dense 3D connectivity, combined in the same 3D vault. At the software level, vector manipulation can be performed between the different memory layers without changing the proposed programming model. This proposal, can remind the structures of HMC architectures (using DRAM technologies) which are based on the concept of vertical vaults. The 3D METEOR architecture has the same approach, where each vault transfers large amount of data

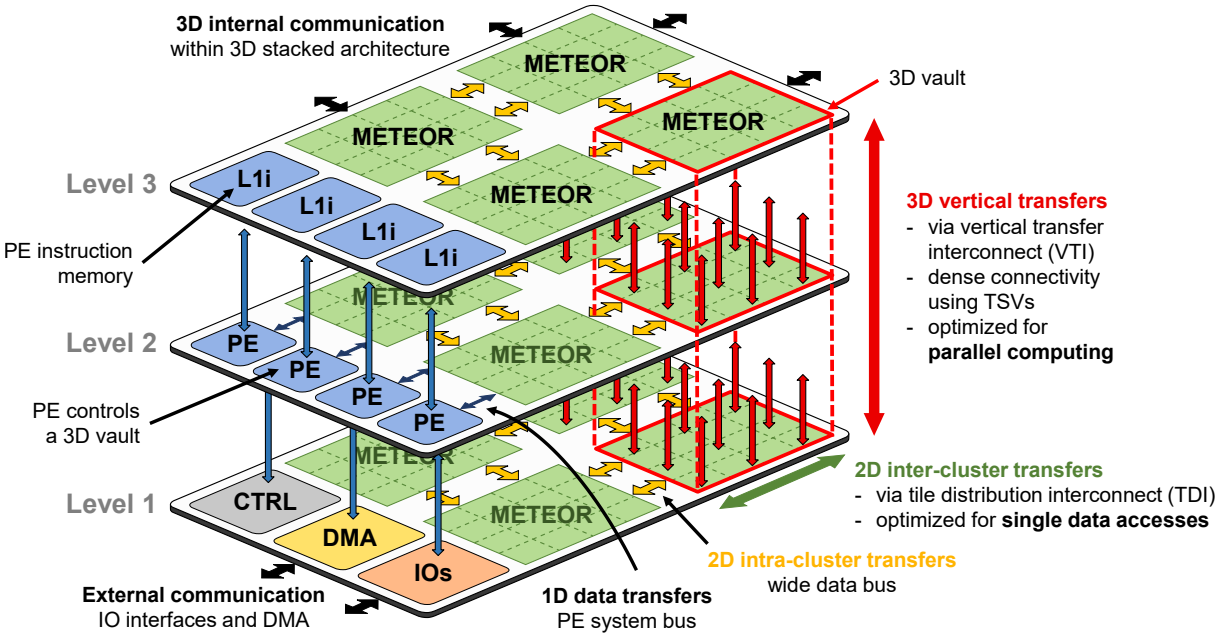


Figure 9: Toward a 3D METEOR architecture proposal.

horizontally on 2D layers and performs vector (or tensor) computations vertically in 3D within memory, driven by an additional control unit (CTRL). Such organized 3D vaults are able to work in parallel (computation/data loading) scheduled by dedicated processing elements (PEs).

The contributions of this PhD focus on the on-chip architecture vision involving high-density memories to minimize the off-chip memory wall problem, however this problem is still a concern. To propose a comprehensive architecture, I contributed to the development of a patent, introducing a dedicated DMA, who is tightly-coupled to the IMC/NMC single tile or multiple tiles, by using an internal and direct communication bus. Thus, data coming from off-chip or on-chip memories will be transferred to the METEOR memory tiles in parallel of the vector processing flow thanks to a dedicated wide data bus. This proposal is suitable for either 2D or 3D implementation. The next step will consist in integrating the DMA in the ArchSim platform to quantify the impact on complete data-intensive applications. Finally, many research opportunities and future works remain, such as architectural evaluation and the distributed programming model, as discussed in the next section.

Perspectives and Future Works

In this section, I present a non-exhaustive list of research topics to address in order to achieve the dream architecture, as proposed in Chapter 3. As a short-term perspective, the "2D" single layer METEOR architecture should be implemented in a digital circuit for further design experiments and measurements. Indeed, many questions remain, such as the accurate measurement of slack time in this vertical communication scheme to ensure the proper scalability of the architecture at interconnect level. The evaluation of instruction pipeline mechanisms is required to measure the impact of interleaved standard data accesses and NMC instruction flow at system level. Moreover, a design space exploration would provide the best physical dimension of the METEOR architecture according to data-intensive application requirements. As a long-term perspective, there are still architectural and technological challenges to be solved, such

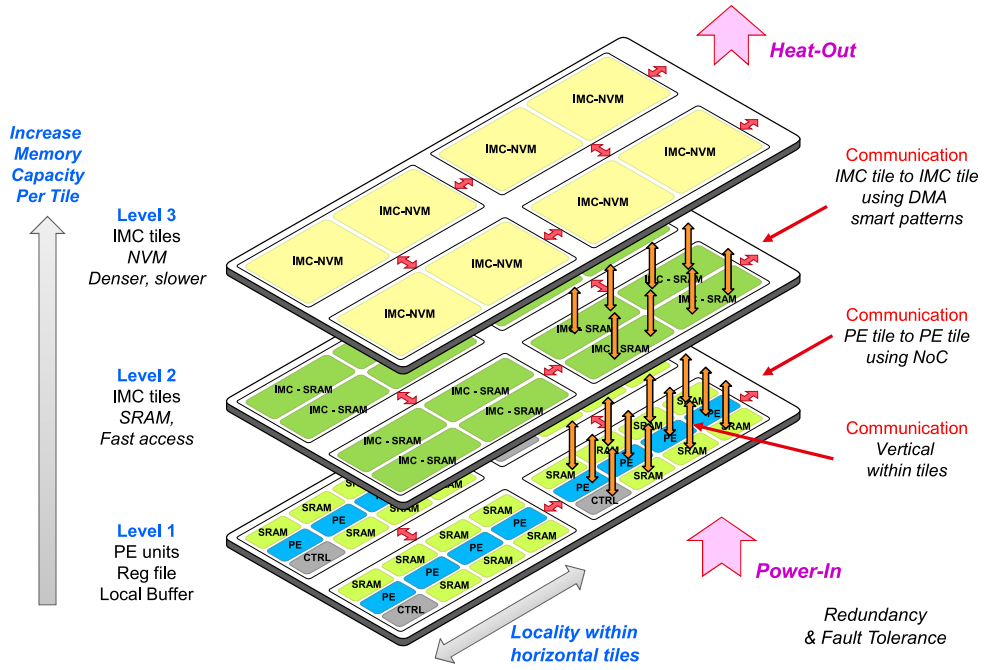


Figure 10: A 3D dream architecture to break the "memory wall" from Chapter 3.

as efficient heat removal in 3D-stacked architectures and the integration of on-chip emerging non-volatile memory (eNVM) coupled with IMC and NMC approaches.

At the software level, many research opportunities lead to the elaboration of a general-purpose programming model dedicated to data-centric architectures (such as METEOR) for both data- and compute-intensive applications. Since my contributions are focused at kernel-level, an application-level evaluation is required to estimate the real performance gains of a full data-intensive workloads. Furthermore, other high-level functions, such as *MapReduce*, *FlatMap* or *Fold*, significantly improve the execution speed of data-intensive applications. For instance, in the software implementation of a neural network application, many memory pattern accesses can be estimated at compile time to improve the energy-efficiency of the memory architecture. At low-level software, the proposed NMC instruction set should be directly integrated into the processor instead of C macros to minimize the impact of instruction preparation, as well as seamless integration into a standard compilation flow. This implies new hardware features and optimization of the computing cores attached in the NMC architecture. During my research work, I noticed the lack of data movement instruction support in processor instruction sets, which is required by the METEOR architecture, would be an opportunity to explore. All these new proposals should be explored and validated using the architectural simulation platform. As a last word, distributed memory-computing architecture are still are at their premise, opening up many hardware, software, architecture and technology challenges and research perspectives.

Appendix A

Résumé en Français

Introduction

Chaque jour, des millions de personnes génèrent d'importants flux de données par le biais d'ordinateurs, de téléphones intelligents, de voitures autonomes ou d'appareils médicaux, ce qui entraîne une explosion massive de données numériques appelée "Big Data". Ces applications gourmandes en données présentent une forte contrainte d'accessibilité aux données par rapport au traitement localisé de ces données. Malheureusement, les architectures d'ordinateurs actuelles peinent à satisfaire les besoins en performance de ces applications en raison des limitations architecturales et technologies des mémoires. Cet écart de performance entre le processeur et la mémoire est appelé le "mur mémoire", comme illustré dans la Figure 2.

Au niveau architectural, la hiérarchie mémoire des architectures des ordinateurs actuels (e.g. Cache mémoire) devient inefficace en matière d'énergie et de performance pour satisfaire les besoins des applications gourmandes en données. De plus, la mise à l'échelle des transistors et les tendances technologiques sont ralenties pour des raisons physiques, marquant aux alentours de 2004, la fin de la loi de Dennard [6]. Ainsi, pour limiter la consommation de puissance excessive des processeurs, la fréquence de fonctionnement est limitée (mur de puissance), réduisant en conséquence les performances des mémoires sur puce, proche du processeur. Les architectures modernes doivent relever deux défis pour réduire ces écarts : en intégrant de nouvelles technologies de mémoire pour résoudre le mur mémoire hors puce, et en intégrant le calcul au sein de la mémoire pour résoudre le mur mémoire sur puce.

Ainsi, cette thèse de doctorat aborde le problème du mur mémoire en proposant une nouvelle vision architecturale et technologique. Dans ce contexte, j'ai développé une architecture mémoire reconfigurable composée de multiple tuiles permettant le calcul vectoriel extensible afin de réduire l'impact du mur mémoire sur puce, détaillée dans le chapitre 4. Chaque tuile étant capable d'exécuter des calculs avec les données stockées dans sa mémoire attachée, comme proposé dans les approches IMC et NMC, mais aussi entre différentes tuiles par le biais d'un interconnect vertical supplémentaire. Pour simplifier le travail du programmeur, j'ai proposé une intégration logicielle souple de cette architecture dans le but de développer un modèle de programmation qui entrelace les accès aux données classiques et les instructions à exécuter dans les tuiles mémoires calculantes, comme détaillée dans le chapitre 5. Afin d'évaluer les performances et les gains énergétiques des architectures mémoires tuilées, j'ai

établi une méthodologie pour explorer et dimensionner efficacement l'interconnect de distribution à travers un flot de conception standard en fonction des paramètres physiques des tuiles mémoires, comme détaillée dans le chapitre 6. Devant le niveau de maturité et la flexibilité d'intégration des technologies émergentes avec les simulateurs architecturaux existants, j'ai développé une plateforme d'exploration architecturale au niveau matériel et logiciel, détaillée dans le chapitre 7. Elle permet d'explorer différentes routines applicatives utilisées dans les applications à forte intensité de données comme le traitement d'images, la recherche dans les bases de données ou les applications de réseaux de neurones. En utilisant cette plateforme, j'ai comparé les gains en matière d'accélération et de réduction d'énergie sur trois architectures différentes utilisant un processeur, un accélérateur SIMD et l'architecture reconfigurable proposée, comme décrit dans le chapitre 8.

Toutes ces contributions ont donné lieu à deux papiers dans des conférences internationales, sur l'étude et la modélisation de l'interconnect de distribution mémoire [88], et sur l'étude de la reconfigurabilité et l'extensibilité de l'architecture proposée [84]. J'ai aussi présenté deux posters [85, 86] sur cette même architecture et déposé un brevet [87] sur les mécanismes de reconfigurabilité décrits dans ce manuscrit. Enfin, pour nous rapprocher de notre vision architecturale, j'ai participé à la rédaction d'un second brevet [92] permettant d'accélérer le déplacement des données entre les couches de la hiérarchie mémoire sans surcharger le bus système du processeur. Par suite, ce résumé détaille les travaux et les contributions réalisés, chapitre par chapitre.

Chapitre 1. Défis des Architectures Économes en Énergie

Le chapitre 1 présente la motivation des travaux de thèse, le contexte des applications gourmandes en données et l'évolution des architectures d'ordinateurs afin d'identifier les problèmes liés aux architectures actuelles. Ces architectures, telles que les CPU, sont conçues pour des applications d'usage général, mais ne permettent pas de surmonter les mouvements massifs de données au sein de leur hiérarchie de mémoire, d'autant plus pour les applications à forte intensité de données (*Big Data*). Ces applications, telles que le traitement des données et des images, impliquent principalement des opérations de multiplication et d'addition (MAC) et des produits matriciels, et les accès spécifiques aux données pourraient être optimisés grâce à des bibliothèques logicielles dédiées. De plus, en réduisant la précision de calcul, les applications de réseaux de neurones quantifiés (QNN) et de fouille de données peuvent bénéficier d'opérations logiques et à largeur de bit réduit pour augmenter l'efficacité énergétique du système et leur temps de traitement.

En outre, les applications à forte intensité de données présentent peu de localité dans le calcul et une réutilisation réduite des données pour lesquels la hiérarchie de mémoire standard devient de plus en plus inefficace. La mémoire cache vise à réduire le mur mémoire sur puce, mais manque de flexibilité et de connectivité, notamment due à sa structure et à ses politiques de placement des données. Comme la mémoire sur puce est conçue pour le calcul haute performance (HPC) et la mémoire hors puce est conçue pour un stockage de grande capacité, le problème du mur mémoire demeure. Pour résoudre ce problème, il faut concevoir de nouvelles architectures efficaces énergiquement en analysant précisément les besoins applicatifs et en optimisant les accès aux données. Les nouvelles technologies mémoire et le calcul distribué sont de nouvelles opportunités pour continuer à repousser ce mur mémoire.

Chapitre 2. État de l'art des Architectures Émergentes Économes en Énergie

Le chapitre 2 décrit les solutions architecturales et technologiques proposées dans la littérature pour améliorer les performances des applications à forte intensité de données. Comme le cache traditionnel n'est plus suffisamment efficace énergétiquement pour répondre aux exigences des applications gourmandes en données, et que les architectures dédiées (ASIC) manquent de souplesse architecturale et de connectivité, il est nécessaire de trouver de nouvelles architectures et technologies. Au niveau architectural, les architectures CGRA offrent cette flexibilité supplémentaire avec une conception rapidement reconfigurable pour intégrer des accélérateurs dédiés et rapprocher le calcul des mémoires. En effet, ces architectures intègrent des accélérateurs spécifiques pour le traitement des tâches redondantes, comme les accélérateurs vectoriels, qui réduisent le temps de traitement à travers un parallélisme massif des données grâce à la vectorisation de l'application. Pour briser le mur mémoire, d'autres solutions architecturales consistent à placer les unités de calcul le plus proche possible de la mémoire afin de réduire les mouvements des données entre le processeur et la mémoire. La plupart de ces architectures centrées sur les données exploitent les approches du calcul dans la mémoire (IMC) et du calcul proche mémoire (NMC), pour exécuter des calculs logiques et arithmétiques directement dans l'unité (ou la tuile) mémoire.

Sur le plan technologique, les approches par empilement en trois dimensions (3D) peuvent combler le fossé de connectivité entre la mémoire et le calcul, en favorisant le couplage entre la mémoire et le calcul tel qu'il est introduit par l'IMC et le NMC. En outre, les mémoires non-volatiles (NVM) émergentes permettraient d'augmenter la densité et l'endurance, pour apporter une plus grande capacité de mémoire sur puce sans transferts mémoires coûteux hors puce. De récents travaux démontrent la possibilité d'effectuer des calculs IMC dans ces NVMs en utilisant le principe de la loi d'Ohm. Les architectures en 3D offrent de nombreuses possibilités de recherche, telle que la portabilité de la programmation des applications à forte intensité de données, la gestion de la mémoire virtuelle et l'amélioration de la dissipation thermique.

Chapitre 3. Un Rêve : une Architecture de Calcul Distribué Empilée en 3D

Dans le chapitre 3, une vision architecturale et technologique rêvée est proposée pour explorer les nouveaux thèmes et les nouvelles possibilités de recherche, illustrée dans la Fig. 3.1. De nombreux défis restent à résoudre, tels que l'intégration 3D avancée à grains fins, la dissipation thermique efficace, les mémoires NVM émergentes sur puce à haute densité et haute endurance, le calcul immergé dans la mémoire (IMC, NMC), les architectures reconfigurables et distribuées, ainsi que leur modèle de programmation générique associé.

Pour aller vers ces directions, les contributions de ma thèse de doctorat abordent les sujets architecturaux de la reconfigurabilité et de l'extensibilité des tuiles de calcul proche mémoire utilisant les technologies CMOS standard. Au niveau architectural, les architectures reconfigurables compensent le manque de connectivité de la hiérarchie traditionnelle des mémoires, ainsi que le manque de flexibilité des architectures ASIC grâce à leurs chemins de données polyvalents. Ces architectures permettent une mise en œuvre souple et efficace entre les accélérateurs matériels hétérogènes et augmentent le couplage entre le calcul et la mémoire. Ainsi, différentes approches de calcul proche mémoire (IMC et NMC) sont intégrées pour rapprocher le calcul de la mémoire et réduire le trafic de mémoire sur le bus système, grâce à la

flexibilité d'interconnexion de l'architecture. Actuellement, les architectures mémoires souffrent d'un manque de connectivité tant au niveau matériel que logiciel, et les accélérateurs de calcul proche mémoire (ou mémoires calculantes) sont rarement adaptés aux applications à forte intensité de données en raison d'un manque d'intégration logicielle.

Chapitre 4. METEOR : une Architecture Tuilée Reconfigurable de Mémoire Calculante

Dans le chapitre 4, je propose une architecture reconfigurable pour évaluer les gains du calcul proche mémoire (IMC et NMC). METEOR est une architecture en grappe de tuiles de calcul, reconfigurable de deux manières : soit par extension horizontale de la mémoire pour calculer sur des vecteurs plus larges, soit par une extension verticale de la mémoire pour utiliser plus de vecteurs. La reconfigurabilité offre la possibilité de modifier ces chemins de données internes afin de déplacer des vecteurs très larges entre les tuiles mémoires sans surcharger le bus système du processeur, et l'extensibilité de l'interconnexion matérielle permet de satisfaire aux exigences de l'application en matière de mémoire. Ces tuiles de calcul utilisent les approches existantes de l'IMC et du NMC qui consiste à exécuter des calculs logiques et arithmétiques à l'intérieur ou proche de la tuile mémoire, respectivement. Avec METEOR, ces approches sont intégrées dans un schéma de vectorisation flexible permettant un fort parallélisme de données, détaillé dans le chapitre suivant.

La conception de l'architecture permet d'entrelacer les accès mémoires standard et le flux d'instructions IMC/NMC par le biais d'un système de pipeline, pour calculer entre les différentes tuiles en évitant les conflits éventuels entre les données. Les mécanismes connus de blocage et transfert des opérandes entre les étages du pipeline sont utilisés pour augmenter le débit du pipeline, réduisant ainsi la consommation globale de l'architecture. Grâce à son interface mémoire générique, METEOR peut être intégré comme une mémoire étroitement couplée ou comme un co-processeur faiblement couplé dans n'importe quelle architecture conventionnelle. Enfin, cette architecture est évaluée au niveau logiciel dans le chapitre 5, et plusieurs compromis de performance et d'énergie entre le partitionnement des tuiles et l'interconnexion standard de la mémoire sont étudiés dans le chapitre 6.

Chapitre 5. Intégration Logicielle pour le Calcul Vectoriel Extensible

Le chapitre 5 décrit l'intégration logicielle du jeu d'instruction IMC/NMC dédié (ISA), ainsi les paramètres de configuration de l'architecture METEOR pour permettre le traitement vectoriel extensible. La définition d'un nouveau type de données vectorielles est proposée pour permettre au compilateur d'allouer, de placer et d'aligner les vecteurs de données dans l'architecture METEOR et pour simplifier le développement du programmeur logiciel. Du point de vue de l'exécution, envoyer une instruction IMC/NMC aux mémoires calculantes est équivalent à écrire une donnée à une adresse spécifique dans une section de mémoire dédiée (une interface de contrôle). Ainsi, le même bus système est utilisé pour l'accès aux données, pour l'envoi d'instructions IMC/NMC et pour reconfigurer l'architecture à larges vecteurs.

En outre, le même programme vectorisé est utilisé pour l'architecture METEOR, les accélérateurs SIMDs et les CPUs conventionnels grâce aux bibliothèques proposées qui s'adaptent au modèle d'exécution de la cible. Cette compatibilité logicielle est une caractéristique importante pour intégrer l'architecture proposée, tout en permettant une évaluation comparative

plus facile pour l'exploration des performances de l'architecture. De plus, l'implémentation de l'opération de réduction, utilisée dans de nombreuses applications gourmandes en données, profite des optimisations matérielles de l'architecture METEOR, tel que le calcul proche mémoire, la flexibilité des chemins de données par la reconfigurabilité et l'extensibilité des vecteurs de calcul. Bien qu'il s'agisse d'un ISA de bas niveau, qui est implémenté comme un ensemble de macros C, et résolu durant l'étape de compilation, la bibliothèque actuelle permet d'évaluer les routines d'applications à forte intensité de données en exploitant un parallélisme important des données sans modifier la conception du processeur, tout en assurant le support d'autres architectures vectorielles. Dans une perspective à long terme, cette bibliothèque de haut niveau sera étendue pour explorer des applications complètes à forte intensité de données afin de mesurer les avantages des approches IMC et NMC à une plus grande échelle.

Chapitre 6. Exploration de la Conception de L'Interconnect Mémoire

Dans le chapitre 6, j'ai évalué les contraintes matérielles des systèmes composés de multiples tuiles mémoires (multi-tuile) et j'ai développé un modèle d'interconnect mémoire afin de dimensionner tout type d'architecture mémoire multi-tuile, comme METEOR. Étant donné qu'une seule tuile mémoire de grande taille ne peut satisfaire les exigences en matière de capacité et de performance des applications gourmandes en données, une architecture multi-tuile est nécessaire pour concevoir des mémoires plus grandes. Les performances des temps d'accès, les puissances consommées et la surface des circuits (PPA) sont mesurées pour plusieurs configurations et plusieurs types de mémoires extraits depuis un générateur mémoire constructeur pour un nœud technologique en 28 nm FD-SOI. Ces diverses architectures sont conçues à travers un flot de conception standard, puis les PPA sont extraites pour calibrer le modèle d'interconnect mémoire proposé.

L'étude des résultats montre qu'il existe plusieurs topologies optimisées en matière de performance et d'énergie entre une seule grande tuile et plusieurs petites tuiles pour une taille de mémoire donnée, comme présenté dans la Figure 6.9. En fragmentant une grande mémoire en plus petites tuiles mémoires, l'architecture multi-tuile permet d'améliorer les performances de 49% et de réduire la consommation d'énergie de 78%, pour un coût de $\times$1,8 la surface. Ces estimations servent à la calibration de la plateforme de simulation, présentée dans le chapitre 7, afin d'améliorer la précision des explorations architecturales. Comme l'architecture METEOR apporte des gains de performance par vectorisation et parallélisme de manière proportionnel au nombre de tuiles mémoires, le coût de l'interconnect mémoire devient limitant lorsque le nombre de tuiles est excessif. Ce modèle sera donc couplé à l'évaluation du système de l'architecture proposée, telle que présentée dans le chapitre 8.

Chapitre 7. ArchSim : une Plateforme de Simulation Logiciel-Matériel pour l'IMC/NMC

Le chapitre 7 détaille la plateforme d'exploration architecturale Archsim qui facilite l'étude des paramètres du système aux niveaux logiciel et matériel afin de dimensionner le circuit de l'architecture avant de le concevoir. Les protocoles d'abstraction matériels fournis par SystemC/TLM permettent de modéliser des composants physiques à plusieurs niveaux de précision (au niveau du cycle processeur jusqu'aux transactions mémoires). Les accès mémoires, les instructions processeurs et les instructions IMC/NMC envoyées aux mémoires calculantes sont mesurées par le biais de compteurs matériels, calibrés par des mesures extraites de circuits in-

tégrés ou de premières évaluations RTL. La couche logicielle intègre et explore les différentes configurations des ISA proposées dans cette thèse, grâce à l'outil PyISAGen qui maintient l'intégrité du jeu d'instructions du programme utilisateur à la mise en œuvre physique. De plus, un ensemble de routines vectorisées, décrites en langage C, sont utilisées dans les applications à forte intensité de données, pour évaluer l'impact architectural de METEOR et autres architectures vectorielles (e.g. SIMD). Les résultats des simulations de performances évaluées avec ArchSim sont détaillés dans le chapitre suivant.

Chapitre 8. Résultats de l'Exploration Architecturale

Le chapitre 8 expose l'étude complète de l'architecture METEOR pour un ensemble de sept routines applicatives gourmandes en données, face à une architecture conventionnelle et une architecture équipée d'un accélérateur vectoriel SIMD. Les sept routines peuvent être classées selon leur complexité algorithmique (linéaire, quadratique et cubique), liée au nombre de boucles imbriquées que la routine doit exécuter. L'architecture METEOR améliore la vitesse d'exécution de 200× pour les routines de complexité linéaire, de 240× pour les routines de complexité quadratique et de 65× pour les routines de complexité cubique par rapport à une architecture CPU conventionnelle. Comparé à un accélérateur SIMD de 512 bits utilisant un accès mémoire large, l'architecture METEOR atteint une réduction de l'EDP (produit de l'énergie par le délai) de 52× pour une routine linéaire, de 71× pour une routine quadratique, et 20× pour une routine cubique.

Contrairement aux systèmes SIMD, l'architecture METEOR continue d'augmenter sa taille de vecteur et réduit les accès à la mémoire CPU grâce à la communication interne supplémentaire entre les tuiles et à la fonctionnalité de l'opération de réduction. Ainsi, la plupart des mouvements de données du bus système entre le processeur et la mémoire sont évités pour la multiplication vecteur-matrice et matrice-matrice utilisées dans les noyaux à complexité linéaire, quadratique et cubique. En conclusion, les limites de performance du noyau cubique sont principalement dues à leur grande empreinte mémoire implémentée sur une architecture METEOR trop petite. Pour résoudre ce problème, une approche architecturale et technologique 3D est nécessaire pour augmenter la capacité de mémoire et la connectivité entre les tuiles.

Conclusion Générale et Perspectives

Dans la perspective de travaux futurs, il est prévu d'implémenter l'architecture METEOR proposée, de valider et de mesurer en détail les performances matérielles et logicielles sur une puce de test. Cette architecture est capable d'exploiter les bénéfices des technologies de calcul proche mémoire (IMC et NMC) grâce au réseau de communication interne supplémentaire et reconfigurable dans un système mémoire multi-tuile. Entre l'interconnect de distribution mémoire standard pour accéder aux données du processeur et cette proposition de communication verticale des données, il existe des contraintes paradoxales. En effet, étant donné que de nombreuses tuiles sont nécessaires pour atteindre un niveau élevé de parallélisme de données et que cela limite les performances de l'interconnect standard (pour une latence et une énergie optimales), les architectures à empilement 3D sont capables de fournir un interconnect standard optimisé sur les couches 2D tandis que la connectivité verticale 3D est bien adaptée pour le déplacement interne des vecteurs larges, comme proposé dans la Figure A.1. Ainsi, l'architecture METEOR en 3D permet d'organiser la hiérarchie de la mémoire par couche tout

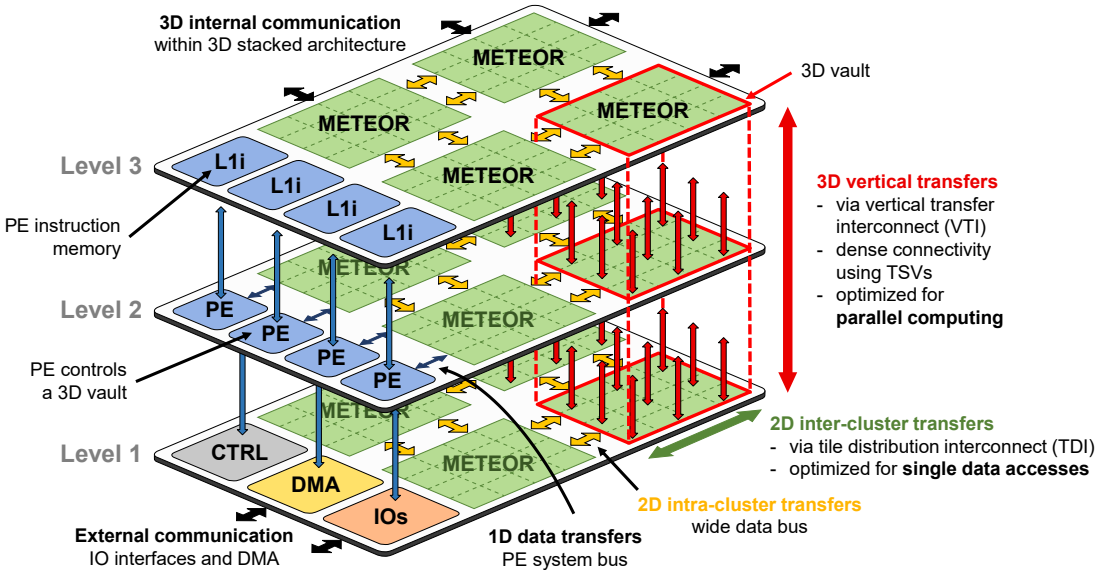


Figure A.1: Vers une architecture METEOR en 3D comme proposé dans la conclusion.

en fournissant une connectivité verticale dense, combinée dans le même "pilier" vertical en 3D. De tels piliers 3D sont capables de travailler en parallèle (calcul/chargement de données) et sont programmés par des éléments de traitement (PEs). Chaque tuile METEOR transfère horizontalement une grande quantité de données sur des couches 2D et effectue des calculs vectoriels (ou tenseurs) verticalement en 3D dans les mémoires, pilotés par l'unité de contrôle (CTRL). Au niveau du logiciel, la manipulation des vecteurs peut être effectuée entre les différentes couches de la mémoire sans changer le modèle de programmation proposé.

Les contributions de ce doctorat se concentrent sur la vision d'une architecture sur puce impliquant des mémoires à haute densité pour minimiser le problème du mur mémoire hors puce, mais ce problème reste préoccupant. Pour proposer une architecture complète, j'ai contribué au développement d'un brevet, en introduisant un DMA dédié, qui est étroitement couplé à la IMC/NMC tuile unique ou à plusieurs tuiles, en utilisant un bus de communication interne et direct. Ainsi, les données provenant de mémoires hors puce ou sur puce seront transférées vers les tuiles mémoires de METEOR en parallèle du flux de traitement vectoriel grâce à un large bus de données dédié. Cette proposition convient pour une implémentation en 2D ou en 3D. La prochaine étape consistera à intégrer le DMA dans la plateforme ArchSim pour quantifier l'impact sur des applications complètes à forte intensité de données. Pour conclure, il reste de nombreuses possibilités de recherche et des travaux futurs, tels que l'évaluation architecturale et le modèle de programmation distribué.

Glossary

List of Abbreviations

ALU Arithmetic Logic Unit, referring to a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers.

CPI Cycles Per Instruction, referring to the average number of clock cycles per instruction for a program (or program fragment) in order to evaluate the processor's performance. It is the multiplicative inverse of Instructions Per Cycle (IPC).

DMA Direct Memory Access (Controller), referring to a device that can transfer data between system memory and other peripherals without involving the processor. The processor is only interrupted at the end of the transfer, which reduces its workload.

DRAM Dynamic Random Access Memory, referring to a type of memory which is randomly accessible but whose content is lost over time. DRAM cells need to be refreshed periodically to keep its content.

DRC Design Rule Checking, referring to geometric constraints imposed on integrated circuit designers to ensure their designs function properly, reliably, and can be produced with acceptable yield.

FPGA Field-Programmable Gate Array, is an integrated circuit designed to be configured by a customer or a designer after manufacturing.

FPU Floating-Point Unit, is a system specially designed to carry out operations on floating-point numbers. The first FPU were co-processors, but most are now integrated into the CPU.

FSM Finite State Machine, referring to a machine that has states and transitions from one state to another state. The transition in an FSM depends only on the type of event and the current state.

ISA Instruction Set Architecture, referring to a hardware/software interface that describes the design of a computer in terms of the basic operations it must support from the programmer's perspective.

RAW Read-After-Write, referring to pipeline data hazard which occurs when an instruction tries to read a results that has not yet been calculated.

RISC Reduced Instruction Set Computer, referring to a computer with a small, optimized set of instructions involving a large number of registers and a regular instruction pipeline that can perform one operation in one instruction.

RTL Register-Transfer Level, is a design abstraction used to create high-level representations of a circuit architecture.

SIMD Single Instruction Multiple Data, referring to a computing parallel architecture in which a single instruction operates on multiple data simultaneously. SIMD is one category of machines under the Flynn's taxonomy.

SRAM Static Random Access Memory, referring to a type of memory that can be accessed randomly with data persistence, but the data are lost when the memory is not powered.

VHDL VHSIC (Very High Speed Integrated Circuit) Hardware Description Language, is a hardware description language used to describe the behaviour as well as the architecture of a digital electronic system.

WAR Write-After-Read, referring to pipeline data hazard which occurs when an instruction tries to write the destination before it is read by a previous instruction.

WAW Write-After-Write, referring to pipeline data hazard which occurs when an instruction tries to write an operand before it is written by a previous instruction.

List of Acronyms

AES Advanced Encryption Standard. 23
AI Artificial Intelligence. 1
ASIC Application-Specific Integrated Circuit. 18, 20, 28, 35, A-3
AT Approximately-Timed. 80, 81

BL Bit-Line. 21, 23, 67, 72
BNN Binary Neural Network. 11

C-SRAM Compute SRAM. 59, 81, 82, 85, 86, 91
CAM Content Addressable Memory. 9, 13, 21, 23
CGRA Coarse-Grained Reconfigurable Architecture. 18, 20–22, 28, 35, A-3
CMOS Complementary Metal Oxide Semiconductors. 33, 34, A-3
CNN Convolutional Neural Network. 2, 10–12, 26, 61, 62
CPU Central Processing Unit. 11, 12, 14, 15, 18, 20, 22, 25, 26, 49, 50, 58, 63, 83, 89, 90, 92–94, 97, 99–101, A-2
CSR Control and Status Register. 40, 45, 47, 48, 55, 57, 58, 82

DBT Dynamic Binary Translation. 76
DK Design-Kit. 67
DL Deep Learning. 1
DLP Data-Level Parallelism. 18, 19, 25, 54, 93
DNN Deep Neural Network. 10, 11
DSL Domain-Specific Language. 20, 32, 33
DSP Digital Signal Processing. 18, 20

EDA Electronic Design Automation. 33
EDP Energy Delay Product. 87, 97, 98, 101, A-6
ELF Executable and Linkable Format. 78
eNVM emerging Non-Volatile Memory. 21, 34, 103

FC Fully-Connected. 11, 61, 62
FD-SOI Fully Depleted Silicon On Insulator. 46, 68, 70, 73, 85, 100, A-5
FET Field-Effect Transistor. 27
FF Flip-Flop. 69
FFT Fast Fourier Transform. 12
FinFET Fin Field-Effect Transistor. 3
FLOPS Floating-point Operations Per Second. 26
FRAM Ferroelectric Random-Access Memory. 21

GCC GNU Compiler Collection. 60, 78, 88
GPD Global Pipeline Dispatcher. 40–44, 46–48, 56, 82
GPU Graphics Processing Unit. 2, 11, 12, 18–20, 24–26, 33, 35

HBM High Memory Bandwidth. 25, 26
HDD Hard Disk Drive. 14, 21
HDL Hardware Description Language. 20
HMC Hybrid Memory Cube. 25, 26, 101

HPC High Performance Computing. 2, 9, 97, A-2

IC Integrated Circuit. 66
ILP Instruction-Level Parallelism. 18
IMC In-Memory Computing. 22–29, 32, 34–37, 39, 40, 44–48, 51, 54, 56–59, 61, 63, 64, 66, 68, 75, 76, 78, 79, 81, 82, 85–87, 90–94, 98, 101–103, A-5
IO Input/Output. 20
IPC Instructions Per Cycle. I
ISS Instruction Set Simulator. 81, 82, 90

LD Linker Descriptor. 78, 79
LLC Last Level Cache. 14
LLVM Low Level Virtual Machine. 76
LSB Least Significant Bit. 58
LSTM Long Short-Term Memory. 10
LSU Load-Store Unit. 92

M3D Monolithic 3D. 27, 33
MAC Multiply-ACcumulate. 11, 15, 20, 57, A-2
METEOR Matrix of Elementary Tiles Enabling Optimal Reconfigurability. 35, 39, 40, 43, 45, 46, 48–51, 53–67, 73, 75, 78, 79, 81–87, 90–95, 97–103, A-4
MIV Monolithic Inter-tier Via. 27, 33
ML Machine Learning. 1
MLC Multi-Level Cell. 21
MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor. 3
MSB Most Significant Bit. 58

NMC Near-Memory Computing. 22–25, 28, 29, 32, 34–37, 39, 40, 44–51, 54, 56–59, 61, 63, 64, 66, 68, 75, 78, 79, 81, 82, 84–87, 90–94, 96–103, A-6
NN Neural Network. 1, 9–11, 15, 18, 20, 21, 24–26
NoC Network-on-Chip. 20, 26, 32
NVM Non-Volatile Memory. 21, 27, 29, 32, 33, 37, A-3

OS Operating System. 76, 98

P&R Place and Route. 68–70, 77, 84, 85, 91
PCI Peripheral Component Interconnect. 15
PCM Phase-Change Memory. 21, 34
PE Processing Element. 11, 32, 33, 35, 43, 44, 48
PIM Processing-In-Memory. 26–28, 101
PPA Power, Performance and Area. 23, 36, 72, 100, A-5

QNN Quantized Neural Network. 9, 11, 15, A-2

RAM Random-Access Memory. 14
ReLU Rectified Linear Unit. 10
RNN Recurrent Neural Network. 2, 10
RRAM Resistive Random-Access Memory. 21, 34

SCM Storage Class Memory. 14, 21, 25, 34, 37

SIMT Single Instruction Multiple Threads. 18
SPM Scratch-Pad Memory. 22, 49, 50
SSD Solid-State Drive. 14, 21, 25, 34
STA Static Timing Analysis. 69, 70
STT-RAM Spin-Transfer Torque Random-Access Memory. 21, 34

TAM Tile Address Mapper. 40–42, 44, 45, 47, 48, 54, 60, 81, 82, 85
TCDM Tightly-Coupled Data Memory. 50
TCM Tightly-Coupled Memory. 49
TCPM Tightly-Coupled Program Memory. 50
TDI Tile Distribution Interconnect. 40, 41, 46–48, 66, 67, 101
TLM Transaction-Level Modeling. 37, 77, 80–82, 84, 86, 100, A-5
TLP Task-Level Parallelism. 18–20
TSV Through-Silicon Via. 26, 27, 33
TTC Tile and Transfer Control. 40, 46–48

VLIW Very Long Instruction Word. 18
VTI Vertical Transfer Interconnect. 41, 46, 67, 101
VTID Vertical Transfer Interconnect Down. 40, 42, 46
VTIU Vertical Transfer Interconnect Up. 40, 42, 46
VTU Vertical Transfer Unit. 40, 42, 44–46, 48, 81

WL Word-Line. 21, 23, 45, 67, 72
WNS Worst Negative Slack. 69, 70

List of Publications

Articles published in refereed international conference and journals

[1] V. Egloff, J.-P. Noel, M. Kooli, B. Giraud, L. Ciampolini, **R. Gauchi**, et al. "Storage Class Memory with Computing Row Buffer: A Design Space Exploration". *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. accepted paper. 2021.

[2] J.-P. Noel, M. Pezzin, **R. Gauchi**, J.-F. Christmann, M. Kooli, H.-P. Charles, et al. "A 35.6TOP-S/W/mm2 3-Stage Pipelined Computational SRAM with Adjustable Form Factor for Highly Data-Centric Applications". *IEEE Solid-State Circuits Letters (L-SSC)*. 2020. DOI: 10.1109/LSSC.2020.3010377.

[3] **R. Gauchi**, V. Egloff, M. Kooli, P. Vivet, J.-P. Noel, B. Giraud, et al. "Reconfigurable Tiles of Computing-In-Memory SRAM Architecture for Scalable Vectorization". *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 2020. DOI: 10.1145/3370748.3406550.

[4] J.-P. Noel, V. Egloff, M. Kooli, **R. Gauchi**, J.-M. Portal, H.-P. Charles, et al. "Computational SRAM Design Automation using Pushed-Rule Bitcells for Energy-Efficient Vector Processing". *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020. DOI: 10.23919/DATE48585.2020.9116506.

[5] **R. Gauchi**, M. Kooli, P. Vivet, J.-P. Noel, E. Beigné, S. Mitra, et al. "Memory Sizing of a Scalable SRAM In-Memory Computing Tile Based Architecture". *IFIP/IEEE International Conference on Very Large Scale Integration and System-on-Chip designs (VLSI-SoC)*. 2019. DOI: 10.1109/VLSI-SoC.2019.8920373.

Patents

[6] M. Kooli, **R. Gauchi**, and P. Vivet. "Module mémoire adapté à mettre en oeuvre des fonctions de calcul". FR2014174. 2020.

[7] **R. Gauchi**, P. Vivet, H.-P. Charles, and S. Mitra. "Module mémoire reconfigurable adapté à mettre en oeuvre des opérations de calcul". FR2008272. 2020.

Workshop and poster presentations

[8] **R. Gauchi**, V. Egloff, M. Kooli, J.-P. Noel, B. Giraud, P. Vivet, et al. "Exploration of a Scalable In-Memory Computing SRAM-based Vector Architecture via a System-on-Chip Evaluation Framework". *ACM/IEEE Design Automation Conference: Work-in-Progress (DAC-WiP)*. Poster session. 2020.

[9] **R. Gauchi**, V. Egloff, M. Kooli, J.-P. Noel, B. Giraud, P. Vivet, et al. "Exploration of a Scalable Vector-Tile-based In-Memory Computing Architecture". *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Computation-In-Memory Workshop (CIMW). 2020.

References

[1] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, et al. "Improved protein structure prediction using potentials from deep learning". *Nature*. 2020. DOI: 10.1038/s41586-019-1923-7.

[2] J. L. Hennessy and D. A. Patterson. "Computer Architecture: A Quantitative Approach". Sixth Edition. Morgan Kaufmann, 2017. ISBN: 9780128119051. URL: https://b-ok.cc/book/3423170/795bf3.

[3] A. Gara, M. A. Blumrich, D. Chen, G. L. Chiu, P. Coteus, M. E. Giampapa, et al. "Overview of the Blue Gene/L system architecture". *IBM Journal of Research and Development*. 2005, DOI: 10.1147/rd.492.0195.

[4] M. Horowitz. "1.1 Computing's energy problem (and what we can do about it)". *IEEE International Solid-State Circuits Conference (ISSCC)*. 2014. DOI: 10.1109/ISSCC.2014.6757323.

[5] G. Moore. "Cramming More Components Onto Integrated Circuits". *Proceedings of the IEEE*. 1998. Reprinted from *Electronics*, April 19, 1965. DOI: 10.1109/JPROC.1998.658762.

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

[7] "Executive Summary - Technology Trend Targets, 2013 Edition". *IEEE International Technology Roadmap for Semiconductors (ITRS)*. 2013. URL: http://www.itrs2.net/2013-itrs.html.

[8] "More Moore - Ground Rules Roadmap for Logic Devices, 2020 Edition". *IEEE International Roadmap for Devices and Systems (IRDS)*. 2020. URL: https://irds.ieee.org/images/files/pdf/2020/2020IRDS_MM.pdf.

[9] C. Philip Chen and C.-Y. Zhang. "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data". *Information Sciences*. 2014. DOI: 10.1016/j.ins.2014.01.015.

[10] A. Sebastian, M. Le Gallo, and R. Khaddam-Aljameh. "Memory devices and applications for in-memory computing". *Nature Nanotechnology*. 2020. DOI: 10.1038/s41565-020-0655-z.

[11] V. Sze, Y. Chen, T. Yang, and J. S. Emer. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". *Proceedings of the IEEE*. 2017. DOI: 10.1109/JPROC.2017.2761740.

[12] Z. C. Lipton, J. Berkowitz, and C. Elkan. "A Critical Review of Recurrent Neural Networks for Sequence Learning". *ArXiv, Computer Science, Machine Learning*. 2015. arXiv: 1506.00019.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, et al. "ImageNet Large Scale Visual Recognition Challenge". *International Journal of Computer Vision*. 2015. DOI: 10.1007/s11263-015-0816-y.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks". *Communications of the ACM*. 2017. DOI: 10.1145/3065386.

[15] H. Park, D. Kim, J. Ahn, and S. Yoo. "Zero and data reuse-aware fast convolution for deep neural networks on GPU". *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. 2016, DOI: 10.1145/2968456.2968476.

[16] X. Lin, C. Zhao, and W. Pan. "Towards Accurate Binary Convolutional Neural Network". *Advances in Neural Information Processing Systems (NIPS)*. 2017. URL: http://papers.nips.cc/paper/6638-towards-accurate-binary-convolutional-neural-network.pdf.

[17] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong. "LogNet: Energy-efficient neural networks using logarithmic computation". *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017. DOI: 10.1109/ICASSP.2017.7953288.

[18] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". *European Conference on Computer Vision (ECCV)*. 2016. DOI: 10.1007/978-3-319-46493-0_32.

[19] A. Skodras, C. Christopoulos, and T. Ebrahimi. "The JPEG 2000 still image compression standard". *IEEE Signal Processing Magazine*. 2001. DOI: 10.1109/79.952804.

[20] M. Cho and D. Brand. "MEC: memory-efficient convolution for deep neural network". *JMLR International Conference on Machine Learning (ICML)*. 2017, URL: http://proceedings.mlr.press/v70/cho17a/cho17a.pdf.

[21] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw. "A 28 nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T Bit Cell Enabling Logic-in-Memory". *IEEE Journal of Solid-State Circuits (JSSC)*. 2016. DOI: 10.1109/JSSC.2016.2515510.

[22] S. Faro and T. Lecroq. "The Exact Online String Matching Problem: a Review of the Most Recent Results". *ACM Computing Surveys (CSUR)*. 2013. DOI: 10.1145/2431211.2431212.

[23] S. Faro and M. O. Külekci. "Fast Multiple String Matching Using Streaming SIMD Extensions Technology". *International Symposium on String Processing and Information Retrieval (SPIRE)*. 2012. DOI: 10.1007/978-3-642-34109-0_23.

[24] Y. Yu, C. Zhang, W. Wang, J. Zhang, and K. Letaief. "Towards Dependency-Aware Cache Management for Data Analytics Applications". *IEEE Transactions on Cloud Computing*. 2019. DOI: 10.1109/TCC.2019.2945015.

[25] R. S. Lakhdar, H.-P. Charles, and M. Kooli. "Data-layout optimization based on memory-access-pattern analysis for source-code performance improvement". *ACM International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. 2020, DOI: 10.1145/3378678.3391874.

[26] B. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, et al. "The Vector-Thread Architecture". *IEEE Micro*. 2004. DOI: 10.1109/MM.2004.90.

[27] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, et al. "The ARM Scalable Vector Extension". *IEEE Micro*. 2017. DOI: 10.1109/MM.2017.35.

[28] H. Fatemi, B. Mesman, H. Corporaal, T. Basten, and P. Jonker. "Run-time reconfiguration of communication in SIMD architectures". *IEEE International Parallel Distributed Processing Symposium (IPDPS)*. 2006. DOI: 10.1109/IPDPS.2006.1639470.

[29] H. Fatemi, B. Mesman, H. Corporaal, T. Basten, and R. Kleihorst. "RC-SIMD: Reconfigurable communication SIMD architecture for image processing applications." *Journal of Embedded Computing*. 2006. URL: http://content.iospress.com/articles/journal-of-embedded-computing/jec00032.

[30] I. Kuon, R. Tessier, and J. Rose. "FPGA Architecture: Survey and Challenges". *Foundations and Trends in Electronic Design Automation*. 2007. DOI: 10.1561/1000000005.

[31] R. Hartenstein. "A Decade of Reconfigurable Computing: a Visionary Retrospective". *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2001. DOI: 10.1109/DATE.2001.915091.

[32] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, et al. "A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications". *ACM Computing Surveys*. 2020. DOI: 10.1145/3357375.

[33] A. Podobas, K. Sano, and S. Matsuoka. "A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective". *IEEE Access*. 2020. DOI: 10.1109/ACCESS.2020.3012084.

[34] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". *IEEE Journal of Solid-State Circuits (JSSC)*. 2017. DOI: 10.1109/JSSC.2016.2616357.

[35] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis. "TANGRAM: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators". *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2019. DOI: 10.1145/3297858.3304014.

[36] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines". *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 2013. DOI: 10.1145/2491956.2462176.

[37] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, et al. "Plasticine: A Reconfigurable Architecture For Parallel Paterns". *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 2017. DOI: 10.1145/3079856.3080256.

[38] T. Agerwala and M. Perrone. "Data Centric Systems, The Next Paradigm in Computing". *International Conference on Parallel Processing (ICPP)*. Keynote. 2014. URL: http://icpp.cs.umn.edu/agerwala.pdf.

[39] W. H. Kautz. "Cellular Logic-in-Memory Arrays". *IEEE Transactions on Computers*. 1969. DOI: 10.1109/T-C.1969.222754.

[40] H. S. Stone. "A Logic-in-Memory Computer". *IEEE Transactions on Computers*. 1970. DOI: 10.1109/TC.1970.5008902.

[41] 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*. 2018. DOI: 10.1145/3131848.

[42] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, et al. "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars". *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. ACM/IEEE International Symposium on Computer Architecture (ISCA). 2016. DOI: 10.1109/ISCA.2016.12.

[43] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie. "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories". *ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2016. DOI: 10.1145/2897937.2898064.

[44] A. Haj-Ali, R. Ben-Hur, N. Wald, R. Ronen, and S. Kvatinsky. "IMAGING: In-Memory AlGorithms for Image processiNG". *IEEE Transactions on Circuits and Systems I: Regular Papers*. 2018. DOI: 10.1109/TCSI.2018.2846699.

[45] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, et al. "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory". *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 2016. DOI: 10.1109/ISCA.2016.13.

[46] K. C. Akyel, H.-P. Charles, J. Mottin, B. Giraud, G. Suraci, S. Thuries, et al. "DRC2: Dynamically Reconfigurable Computing Circuit based on memory architecture". *IEEE International Conference on Rebooting Computing (ICRC)*. 2016. DOI: 10.1109/ICRC.2016.7738698.

[47] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das. "Compute Caches". *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2017. DOI: 10.1109/HPCA.2017.21.

[48] Q. Dong, S. Jeloka, M. Saligane, Y. Kim, M. Kawaminami, A. Harada, et al. "A 4 + 2T SRAM for Searching and In-Memory Computing With 0.3-V VDDmin". *IEEE Journal of Solid-State Circuits (JSSC)*. 2017. DOI: 10.1109/JSSC.2017.2776309.

[49] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy. "X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories". *IEEE Transactions on Circuits and Systems I: Regular Papers*. 2018. DOI: 10.1109/TCSI.2018.2848999.

[50] G. H. Loh, N. Jayasena, M. H. Oskin, M. Nutter, D. Roberts, M. Meswani, et al. "A Processing-in-Memory Taxonomy and a Case for Studying Fixed-function PIM". *Workshop on Near-Data Processing (WoNDP) at IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. 2013. URL: https://www.cs.utah.edu/wondp/wondp2013-paper2-final.pdf.

[51] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, et al. "A 28-nm Compute SRAM With Bit-Serial Logic/Arithmetic Operations for Programmable In-Memory Vector Computing". *IEEE Journal of Solid-State Circuits (JSSC)*. 2019. DOI: 10.1109/JSSC.2019.2939682.

[52] Y. Zhang, L. Xu, Q. Dong, J. Wang, D. Blaauw, and D. Sylvester. "Recryptor: A Reconfigurable Cryptographic Cortex-M0 Processor With In-Memory and Near-Memory Computing for IoT Security". *IEEE Journal of Solid-State Circuits (JSSC)*. 2018. DOI: 10.1109/JSSC.2017.2776302.

[53] F. Schuiki, M. Schaffner, F. K. Gürkaynak, and L. Benini. "A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets". *IEEE Transactions on Computers*. 2019. DOI: 10.1109/TC.2018.2876312.

[54] Y. Zhang, L. Xu, K. Yang, Q. Dong, S. Jeloka, D. Blaauw, et al. "Recryptor: A reconfigurable in-memory cryptographic Cortex-M0 processor for IoT". *IEEE Symposium on VLSI Circuits*. 2017. DOI: 10.23919/VLSIC.2017.8008501.

[55] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, et al. "Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks". *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 2018. DOI: 10.1109/ISCA.2018.00040.

[56] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, et al. "A Compute SRAM with Bit-Serial Integer/Floating-Point Operations for Programmable In-Memory Vector Acceleration". *IEEE International Solid-State Circuits Conference (ISSCC)*. 2019. DOI: 10.1109/ISSCC.2019.8662419.

[57] W. A. Simon, Y. M. Qureshi, A. Levisse, M. Zapater, and D. Atienza. "BLADE: A BitLine Accelerator for Devices on the Edge". *ACM Great Lakes Symposium on VLSI (GLSVLSI)*. 2019. DOI: 10.1145/3299874.3317979.

[58] M. Rios, W. Simon, A. Levisse, M. Zapater, and D. Atienza. "An Associativity-Agnostic in-Cache Computing Architecture Optimized for Multiplication". *IFIP/IEEE International Conference on Very Large Scale Integration and System-on-Chip designs (VLSI-SoC)*. 2019. DOI: 10.1109/VLSI-SoC.2019.8920317.

[59] G. Singh, L. Chelini, S. Corda, A. Javed Awan, S. Stuijk, R. Jordans, et al. "A Review of Near-Memory Computing Architectures: Opportunities and Challenges". *Euromicro Conference on Digital System Design (DSD)*. 2018. DOI: 10.1109/DSD.2018.00106.

[60] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, et al. "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing". *IEEE International 3D Systems Integration Conference (3DIC)*. 2013. DOI: 10.1109/3DIC.2013.6702348.

[61] J. T. Pawlowski. "Hybrid memory cube (HMC)". *IEEE Hot Chips 23 Symposium (HCS)*. 2011. DOI: 10.1109/HOTCHIPS.2011.7477494.

[62] D. U. Lee, K. W. Kim, K. W. Kim, K. S. Lee, S. J. Byeon, J. H. Kim, et al. "A 1.2 V 8 Gb 8-Channel 128 GB/s High-Bandwidth Memory (HBM) Stacked DRAM With Effective I/O Test Circuits". *IEEE Journal of Solid-State Circuits (JSSC)*. 2015. DOI: 10.1109/JSSC.2014.2360379.

[63] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu. "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost". *ACM Transactions on Architecture and Code Optimization (TACO)*. 2016. DOI: 10.1145/2832911.

[64] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, et al. "Gather-Scatter DRAM: In-DRAM address translation to improve the spatial locality of non-unit strided accesses". *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2015. DOI: 10.1145/2830772.2830820.

[65] Dominique Lavenier, Charles Deltel, David Furodet, and Jean-François Roy. "BLAST on UPMEM". Research Report. INRIA Rennes - Bretagne Atlantique, 2016. URL: https://hal.archives-ouvertes.fr/hal-01294345.

[66] S. Ghose, A. Boroumand, J. S. Kim, J. Gomez-Luna, and O. Mutlu. "Processing-in-memory: A workload-driven perspective". *IBM Journal of Research and Development*. 2019. DOI: 10.1147/JRD.2019.2934048.

[67] E. Azarkhish, C. Pfister, D. Rossi, I. Loi, and L. Benini. "Logic-Base Interconnect Design for Near Memory Computing in the Smart Memory Cube". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2017. DOI: 10.1109/TVLSI.2016.2570283.

[68] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis. "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory". *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2017. DOI: 10.1145/3037697.3037702.

[69] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, et al. "Active Memory Cube: A processing-in-memory architecture for exascale systems". *IBM Journal of Research and Development*. 2015. DOI: 10.1147/JRD.2015.2409732.

[70] E. Azarkhish, D. Rossi, I. Loi, and L. Benini. "Neurostream: Scalable and Energy Efficient Deep Learning with Smart Memory Cubes". *IEEE Transactions on Parallel and Distributed Systems (TPDS)*. 2017. DOI: 10.1109/TPDS.2017.2752706.

[71] "UPMEM company website". URL: https://www.upmem.com/.

[72] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay. "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory". *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 2016. DOI: 10.1109/ISCA.2016.41. URL: https://iscaconf.org/isca2016/wp-content/uploads/2016/07/6-2.pdf.

[73] S. K. Samal, D. Nayak, M. Ichihashi, S. Banna, and S. K. Lim. "Monolithic 3D IC vs. TSV-based 3D IC in 14nm FinFET technology". *IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. 2016. DOI: 10.1109/S3S.2016.7804405.

[74] Y. Yu and N. K. Jha. "Energy-Efficient Monolithic 3D on-Chip Memory Architectures". *IEEE Transactions on Nanotechnology*. 2017. DOI: 10.1109/TNANO.2017.2731871.

[75] K. Chang, D. Kadetotad, Y. Cao, J. s. Seo, and S. K. Lim. "Monolithic 3D IC designs for low-power deep neural networks targeting speech recognition". *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 2017. DOI: 10.1109/ISLPED.2017.8009175.

[76] L. Millet, S. Chevobbe, C. Andriamisaina, L. Benaissa, E. Deschaseaux, E. Beigne, et al. "A 5500-frames/s 85-GOPS/W 3-D Stacked BSI Vision Chip Based on Parallel In-Focal-Plane Acquisition and Processing". *IEEE Journal of Solid-State Circuits*. 2019, DOI: 10.1109/JSSC.2018.2886325.

[77] M. M. S. Aly, M. Gao, G. Hills, C. S. Lee, G. Pitner, M. M. Shulaker, et al. "Energy-Efficient Abundant-Data Computing: The N3XT 1,000x". *Computer*. 2015. DOI: 10.1109/MC.2015.376.

[78] L. Brunet, C. Fenouillet-Beranger, P. Batude, S. Beaurepaire, F. Ponthenier, N. Rambal, et al. "Breakthroughs in 3D Sequential technology". *IEEE International Electron Devices Meeting (IEDM)*. 2018, DOI: 10.1109/IEDM.2018.8614653.

[79] S. Thuries, O. Billoint, S. Choisnet, R. Lemaire, P. Vivet, P. Batude, et al. "M3D-ADTCO: Monolithic 3D Architecture, Design and Technology Co-Optimization for High Energy Efficient 3D IC". *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, DOI: 10.23919/DATE48585.2020.9116293.

[80] J. Jourdon, S. Lhostis, S. Moreau, J. Chossat, M. Arnoux, C. Sart, et al. "Hybrid bonding for 3D stacked image sensors: impact of pitch shrinkage on interconnect robustness". *IEEE International Electron Devices Meeting (IEDM)*. 2018, DOI: 10.1109/IEDM.2018.8614570.

[81] M. M. Sabry, A. K. Coskun, D. Atienza, T. Š. Rosing, and T. Brunschwiler. "Energy-Efficient Multiobjective Thermal Control for Liquid-Cooled 3-D Stacked Architectures". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 2011, DOI: 10.1109/TCAD.2011.2164540.

[82] D. R. B. Ly, J. Noel, B. Giraud, P. Royer, E. Esmanhotto, N. Castellani, et al. "Novel 1T2R1T RRAM-based Ternary Content Addressable Memory for Large Scale Pattern Recognition". *IEEE International Electron Devices Meeting (IEDM)*. 2019, DOI: 10.1109/IEDM19573.2019.8993621.

[83] T. F. Wu, B. Q. Le, R. Radway, A. Bartolo, W. Hwang, S. Jeong, et al. "14.3 A 43pJ/Cycle Non-Volatile Microcontroller with 4.7μs Shutdown/Wake-up Integrating 2.3-bit/Cell Resistive RAM and Resilience Techniques". *IEEE International Solid-State Circuits Conference (ISSCC)*. 2019. DOI: 10.1109/ISSCC.2019.8662402.

[84] R. Gauchi, V. Egloff, M. Kooli, P. Vivet, J.-P. Noel, B. Giraud, et al. "Reconfigurable Tiles of Computing-In-Memory SRAM Architecture for Scalable Vectorization". *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 2020. DOI: 10.1145/3370748.3406550.

[85] R. Gauchi, V. Egloff, M. Kooli, J.-P. Noel, B. Giraud, P. Vivet, et al. "Exploration of a Scalable Vector-Tile-based In-Memory Computing Architecture". *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Computation-In-Memory Workshop (CIMW). 2020.

[86] R. Gauchi, V. Egloff, M. Kooli, J.-P. Noel, B. Giraud, P. Vivet, et al. "Exploration of a Scalable In-Memory Computing SRAM-based Vector Architecture via a System-on-Chip Evaluation Framework". *ACM/IEEE Design Automation Conference: Work-in-Progress (DAC-WiP)*. Poster session. 2020.

[87] R. Gauchi, P. Vivet, H.-P. Charles, and S. Mitra. "Module mémoire reconfigurable adapté à mettre en oeuvre des opérations de calcul". FR2008272. 2020.

[88] R. Gauchi, M. Kooli, P. Vivet, J.-P. Noel, E. Beigné, S. Mitra, et al. "Memory Sizing of a Scalable SRAM In-Memory Computing Tile Based Architecture". *IFIP/IEEE International Conference on Very Large Scale Integration and System-on-Chip designs (VLSI-SoC)*. 2019. DOI: 10.1109/VLSI-SoC.2019.8920373.

[89] J.-P. Noel, M. Pezzin, R. Gauchi, J.-F. Christmann, M. Kooli, H.-P. Charles, et al. "A 35.6TOP-S/W/mm2 3-Stage Pipelined Computational SRAM with Adjustable Form Factor for Highly Data-Centric Applications". *IEEE Solid-State Circuits Letters (L-SSC)*. 2020. DOI: 10.1109/LSSC.2020.3010377.

[90] J.-P. Noel, V. Egloff, M. Kooli, R. Gauchi, J.-M. Portal, H.-P. Charles, et al. "Computational SRAM Design Automation using Pushed-Rule Bitcells for Energy-Efficient Vector Processing". *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020. DOI: 10.23919/DATE48585.2020.9116506.

[91] V. Egloff, J.-P. Noel, M. Kooli, B. Giraud, L. Ciampolini, R. Gauchi, et al. "Storage Class Memory with Computing Row Buffer: A Design Space Exploration". *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. accepted paper. 2021.

[92] M. Kooli, R. Gauchi, and P. Vivet. "Module mémoire adapté à mettre en oeuvre des fonctions de calcul". FR2014174. 2020.

[93] ARM, ed. "AMBA 3 AHB-Lite Protocol Specification v1.0". 2010. URL: https://developer.arm.com/docs/ihi0033/a/amba-3-ahb-lite-protocol-specification-v10.

[94] M. Kooli, H.-P. Charles, C. Touzet, B. Giraud, and J.-P. Noel. "Software Platform Dedicated for In-Memory Computing Circuit Evaluation". *IEEE/ACM International Symposium on Rapid System Prototyping (RSP)*. 2017. DOI: 10.1145/3130265.3130322.

[95] L.-N. Pouchet. "PolyBench/C, the Polyhedral Benchmark suite". 2015. URL: http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/.

[96] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, et al. "The Gem5 Simulator". *ACM SIGARCH Computer Architecture News*. 2011. DOI: 10.1145/2024716.2024718.

[97] T. E. Carlson, W. Heirman, and L. Eeckhout. "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulations". *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 2011. DOI: 10.1145/2063384.2063454.

[98] D. Sanchez and C. Kozyrakis. "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems". *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 2013. DOI: 10.1145/2485922.2485963.

[99] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, et al. "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation". *ACM SIGPLAN Notices*. 2005. DOI: 10.1145/1064978.1065034.

[100] M. Chiang, T. Yeh, and G. Tseng. "A QEMU and SystemC-Based Cycle-Accurate ISS for Performance Estimation on SoC Development". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*. 2011. DOI: 10.1109/TCAD.2010.2095631.

[101] "IEEE Standard for Standard SystemC Language Reference Manual". *IEEE Standard 1666-2011*. 2012. DOI: 10.1109/IEEESTD.2012.6134619.

[102] J. Asynsley. "OSCI TLM-2.0 Language Reference Manual". 2009. URL: https://www.accellera.org/images/downloads/standards/systemc/TLM_2_0_LRM.pdf.

[103] H. Lebreton and P. Vivet. "Power Modeling in SystemC at Transaction Level, Application to a DVFS Architecture". *IEEE Computer Society Annual Symposium on VLSI*. 2008. DOI: 10.1109/ISVLSI.2008.71.

[104] Western Digital, ed. "Open Source RISC-V SweRV Instruction Set Simulator". 2018. URL: https://github.com/westerndigitalcorporation/swerv-ISS.

[105] J.-F. Christmann, F. Berthier, D. Coriat, I. Miro-Panades, E. Guthmuller, S. Thuries, et al. "A 50.5 ns Wake-Up-Latency 11.2 pJ/Inst Asynchronous Wake-Up Controller in FDSOI 28 nm". *Journal of Low Power Electronics and Applications (JLPEA)*. 2019. DOI: 10.3390/jlpea9010008.