COLLEGE DOCTORAL PAYS DE LA LOIRE
ED MATHEMATIQUES STIC

Nantes Université

THÈSE DE DOCTORAT DE

NANTES UNIVERSITÉ

ÉCOLE DOCTORALE N° 641
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Électronique*

Par
Quentin DARIOL

Early Timing and Energy Prediction and Optimization of Artificial Neural Networks on Multi-Core Platforms

Thèse présentée et soutenue à Nantes, France, le 27/11/2023
Unité de recherche : IETR UMR CNRS 6164

Rapporteurs avant soutenance :

Prof. Dr. Matthias JUNG	Full professor, Würzburg Universität, Germany
Dr. Angeliki KRITIKAKOU	Associate professor - HDR, IRISA/INRIA, Université de Rennes, France

Composition du Jury :

Président :	Prof. Dr. Frédéric PÉTROT	Full professor, TIMA, Université Grenoble Alpes, France
Examinateurs :	Dr. Kim GRÜTTNER	Head of department, German Aerospace Center (DLR), Germany
	Prof. Dr. Matthias JUNG	Full professor, Würzburg Universität, Germany
	Dr. Angeliki KRITIKAKOU	Associate professor - HDR, IRISA/INRIA, Univ. Rennes, France
	Prof. Dr. Gregor SCHIELE	Full professor, Duisburg-Essen Universität, Germany
Dir. de thèse :	Prof. Dr. Sébastien PILLEMENT	Full professor, IETR, Nantes Université, France
Encadrant :	Dr. Sébastien LE NOURS	Associate professor - HDR, IETR, Nantes Université, France

Invité :

Dr. Domenik HELMS	Principal scientist, German Aerospace Center (DLR), Germany

Table of Contents

List of Figures

List of Tables

RÉSUMÉ LONG

Contexte: La croissance importante du domaine de l'Internet des objets (IoT) s'accompagne du besoin d'applications reposant sur l'utilisation d'algorithmes d'Intelligence Artificielle (IA) et en particulier de Réseaux de Neurones artificiels (NNs). Habituellement, les NNs sont exécutés au niveau du cloud des applications IoT, car il contient une quantité suffisante de ressources de calcul pour permettre une exécution rapide et efficace. Toutefois, cela nécessite la transmission des données récoltées par les capteurs du niveau edge jusqu'au cloud pour le traitement. Les résultats du NN sont ensuite retransmis du cloud jusqu'à l'edge pour être communiqués à l'utilisateur de l'application. Les tendances actuelles visent plutôt à déployer les algorithmes d'IA au niveau edge en supprimant ainsi les transferts de données coûteux entre edge et cloud, permettant notamment l'amélioration du temps d'exécution et de la consommation énergétique.

Cela n'est néanmoins pas une tâche aisée, car les plates-formes embarquées disponibles au niveau edge ont des ressources de calcul et de mémoire limitées ainsi qu'un budget strict en termes de temps et d'énergie, et les NNs sont gourmands en calcul et en mémoire. Pour trouver des solutions qui optimisent les performances, l'énergie et l'utilisation des ressources, plusieurs approches d'évaluation et optimisation des NNs au niveau edge ont déjà été proposées. Beaucoup se concentrent sur le prototypage rapide, qui vise à déployer et caractériser par la mesure les déploiements de NNs sur cible réelle. Ce type d'approche requiert cependant un effort conséquent car de nombreuses solutions potentielles doivent être déployés et testés sur la plateforme cible. La technologie de mise en œuvre est également figée en raison de la nécessité d'avoir la plate-forme réelle dans la boucle, ce qui limite les possibilités en ce qui concerne l'exploration architecturale.

Pour ces raisons, d'autres approches ont été proposées, se basant sur des modèles analytiques purs qui permettent une exploration rapide et efficace des accélérateurs matériels pour les NN. Ces approches peuvent par contre difficilement être appliquées sur les plateformes multicœurs, qui sont des cibles d'implémentation privilégiées au niveau edge. Sur les plateformes multicœurs, les accès concurrents des cœurs de traitement aux ressources partagées occasionnent des contentions, qui impactent le temps d'exécution et l'énergie consommée. Pour l'exécution en pipeline des NNs sur les plates-formes multicœurs, plusieurs aspects rendent difficile l'évaluation des propriétés non fonctionnelles :

(1) l'expression et l'utilisation de différents parallélismes de NNs tels que le parallélisme intra- et inter-couche (pipeline) qui doivent être convenablement modélisés pour permettre l'optimisation de l'exécution.

(2) les conflits des cœurs de traitement pour accéder aux ressources partagées telles que le bus et la mémoire qui peuvent également survenir lorsque plusieurs cœurs tentent d'y accéder simultanément, ce qui entraîne des sur-coûts importants en termes de temps et d'énergie qui doivent être correctement modélisés,

(3) L'architecture de la plateforme : le nombre et les types de composants (cœurs, mémoire, bus, périphériques) ont un effet important sur les propriétés temporelles et la consommation d'énergie qui doit être correctement modélisé.

(4) le comportement dynamique du système, où les cœurs exécutent différentes phases (calcul, communication) et peuvent être activés ou désactivés (par exemple via l'utilisation du clock gating).

(5) les différentes charges de calcul/communication liées au NN d'entrée, les NNs pouvant avoir différents nombre, types et tailles de couches.

Outre les défis liés à l'évaluation, il existe un grand nombre de déploiements possibles d'un NN sur une plateforme multicœur. Les plateformes au niveau edge ont souvent de fortes contraintes de temps et d'énergie, et une exploration intensive du vaste espace de conception est donc nécessaire pour trouver des déploiements qui respectent ces contraintes. Il est nécessaire de proposer un flot automatisé d'exploration de l'espace de conception (DSE) permettant une évaluation rapide et fiable des différents déploiements, en tenant compte des enjeux susmentionnés, afin d'identifier des solutions optimisées. Ce flot doit notamment permettre de :

(1) Trouver des déploiements optimisés de NNs sur une plateforme fixe spécifiée par l'utilisateur,

(2) Optimiser conjointement les dimensions de la plateforme matérielle (nombre de cœurs, taille des mémoires) et le déploiement logiciel des NNs.

Enjeux de recherche: Le travail présenté dans le cadre de cette thèse vise à répondre aux questions de recherche suivantes :

(I) Comment fournir une évaluation rapide et précise en amont des phases de conception des propriétés temporelles et de l'énergie des déploiements de NNs sur des plateformes multicœurs ?

(II) Est-ce qu'une approche basée sur des modèles est plus pertinente que le prototypage rapide ?

(III) Une approche basée sur les modèles est-elle adaptée à l'exploration rapide et fiable de l'espace de conception (DSE) des déploiements de NNs sur des plateformes multi-cœurs ?

Proposition: Nous présentons un flot complet de prédiction et d'optimisation des propriétés temporelles et de l'énergie qui combine plusieurs approches de modélisation. Le flot proposé permet d'optimiser l'occupation des ressources sans dégrader les performances des NNs mis en œuvre. Ces travaux permettent d'aboutir aux contributions suivantes :

(1) Un flot de modélisation hybride pour les propriétés temporelles. Ce flot s'appuie des modèles analytiques décrivant les phases de calcul et de communication des cœurs exécutant des NNs. Ces modèles sont calibrés par la mesure et utilisés dans une simulation de haut niveau qui permet de prendre en compte les ressources partagées.

(2) Un flot de modélisation de la puissance et de l'énergie, qui se base sur la caractérisation par la mesure et les traces d'exécution estimées par le flot de prédiction du temps.

(3) Un flot d'Exploration de l'Espace de Conception (DSE) qui utilise des modèles analytiques de haut niveau d'abstraction pour trouver des implémentations de NNs optimisées. Le flot de modélisation proposé en (1) et (2) est ensuite utilisé pour correctement classer les solutions sélectionnées et ainsi retourner les meilleurs implémentations candidates.

Hypothèses de travail: Dans cette thèse, nous nous concentrons sur l'étude des Multi-Layer Perceptrons (MLPs) aussi appelés réseaux entièrement connectés ainsi que sur les Réseaux Neuronaux Convolutifs (CNN). Afin de faciliter le processus de déploiement, d'analyse et d'optimisation des NNs, nous les modélisons dans un Modèle de Calcul (MoC) orienté flot de données appelé Synchronous DataFlow (SDF). Nous expliquons notamment comment nous décrivons les NNs avec différents niveaux de granularité, appelés *clusterings*, qui définissent le nombre de groupes de neurones générés par couche du NN. Le *clustering* exprime le parallélisme intra-couche des NNs, permettant l'exécution des neurones en parallèles, mais occasionnant davantage de communications.

Nous utilisons un Modèle d'Architecture (MoA) basée sur des tuiles pour les plateformes multicœurs, qui comprend un nombre de tuiles défini par l'utilisateur, ainsi qu'un bus et une mémoire partagée. Le MoA peut aussi intégrer l'utilisation de gestion de la consommation par le biais du "clock gating" et l'utilisation d'un signal d'interruption. Les NNs sont programmés sur les plates-formes qui souscrivent à notre MoA en affectant aux cores les phases de calcul des neurones et les canaux de communication en mémoire. Cette étape s'appelle le *mapping* et elle permet d'exploiter le parallélisme intra-couche issu du clustering et également le parallélisme inter-couche (exécution en pipeline) des NNs - en mappant les différentes couches sur des cœurs différents.

Pour mener nos expériences (calibration et validation des modèles), nous avons mis au point un prototype de plateforme multicœur sur FPGA UltraScale. Cette plateforme est constituée d'un ensemble de tuiles comportant un coeur de calcul MicroBlaze et sa mémoire locale privée de données et d'instructions, d'un bus partagé AXI et d'une mémoire partagée. Cette plateforme est dotée d'une infrastructure de mesure du temps et de la puissance. Pour la validation de notre approche, nous considérons 4 NNs dont 3 MLPs aux caractéristiques différentes et 1 CNN. Nous avons testé un total de 54 mappings différents issus de ces 4 NNs.

Prédiction des propriétés temporelles: Nous présentons une méthodologie de modélisation permettant de prédire les propriétés temporelles, telles que la latence et le débit pour les NN déployés sur des plates-formes multicœurs qui respectent notre modèle d'architecture. La principale contribution de la méthodologie proposée est d'offrir des prédictions rapides et précises des propriétés temporelles en étant scalable vis à vis:

— du NN considéré en entrée,
— du clustering/mapping de ce NN,
— du nombre de tuiles utilisées,
— de la quantité de calcul/communication,
— de différentes procédures de communication (scrutage ou interruption).

Ce flot s'appuie sur des modèles analytiques décrivant les phases calculatoires et de communications sur la plateforme lors de l'exécution des NNs, sur la calibration par la mesure, et sur la simulation pour modéliser les ressources partagées. La combinaison de ces différentes approches de modélisation permet d'aboutir à un modèle hybride rapide à exécuter et offrant une grande précision et modularité. Les modèles analytiques de temps de calcul pour les NN décrivent les opérations exécutées dans les couches. Les prédictions sont confrontées à des mesures obtenues lors de l'implémentation sur cible réelle. Pour les 54 mappings testés, les modèles proposés ont une précision de plus de 97%. Ils prennent une durée de l'ordre de la centaine de millisecondes pour évaluer un mapping. Ce temps d'évaluation monte à 20 s si on prend en compte le temps nécessaire pour compiler le modèle exécutable. Pour comparaison, nous observons que notre infrastructure de mesure automatique du temps d'exécution prend 40 s pour évaluer un mapping. Dans ce cas là, les modèles permettent un gain de temps d'un facteur 2 vis à vis d'une approche de prototypage rapide. Il est important de noter que le délai fourni pour le prototypage rapide ne prend pas en compte le temps d'entrainement du NN et génération des fichiers sources en C, de synthèse du FPGA, de génération du BSP et de compilation des librairies pour supporter l'exécution de software sur la plateforme. De même le temps de calibration des modèles n'est pas pris en compte.

Prédiction de l'énergie: Nous présentons un flot de prédiction de la puissance et de l'énergie pour les NNs déployés sur des plateformes multicœurs qui respectent notre MoA. Les modèles proposés s'appuient sur notre flot de simulation pour la prédiction des propriétés temporelles. Il est utilisé pour prédire la puissance et la consommation d'énergie en fonction des phases exécutées par les tuiles et de l'éventuelle contention des ressources. Le modèle est obtenu par une phase de caractérisation par la mesure pour offrir une modélisation précise de la consommation d'énergie dans les plateformes multi-cœurs. Notre approche de modélisation combine la simulation et la caractérisation par la mesure, ce qui permet d'obtenir un modèle scalable vis à vis du clustering, du mapping, de la charge de travail en calcul et communication ainsi que l'utilisation de stratégies de gestion de l'énergie. Le flot de prédiction de l'énergie peut être utilisé à deux fins:

(1) Évaluer et trouver des mappings de NNs optimisés sur une plateforme multi-cœur fixe.

(2) Optimiser conjointement les dimensions de la plateforme multicœur (nombre de tuiles, taille des mémoires) et le déploiement des NNs.

Nous présentons une étape de calibration complémentaire qui permet d'étendre l'applicabilité du modèle aux plateformes avec différents nombres de tuiles et tailles de mémoire privée, au prix d'un effort de caractérisation plus intensif. Nous évaluons notre flot de modélisation de la puissance et de l'énergie sur les 54 mappings des 4 NN considérés. Nous testons également son applicabilité sur 7 versions de plateformes différentes. Les résultats montrent que le flot permet de prédire la puissance et l'énergie avec plus de 93 %

de précision. Les prédictions sont rapides, le modèle étant implémenté sous la forme d'un script Python utilisé en post-processing du flot de modélisation des propriétés temporelles. Le temps d'exécution du script est négligeable vis à vis du temps de compilation des modèles de simulation, le temps d'évaluation restant donc de 20 s approximativement pour la prédiction du temps et de l'énergie par mapping. Le flot de prédiction des propriétés temporelles et de l'énergie que nous proposons permet de répondre aux questions de recherche **(I)** et **(II)**.

Exploration de l'Espace de Conception (DSE): Nous présentons un flot de DSE permettant de chercher efficacement et d'optimiser les mappings de NNs sur les plateformes multicœurs respectant notre MoA sous des contraintes définies par l'utilisateur pour les propriétés temporelles et l'énergie. Le flot peut être utilisé de deux manières :

(1) Rechercher des clusterings/mappings optimisés pour les NNs déployés sur une plateforme fixe spécifiée par l'utilisateur,

(2) Optimiser conjointement la plateforme concernant le nombre de cœurs et la taille des mémoires, ainsi que l'implantation du logiciel (clustering/mapping des NNs).

L'objectif dans la mise en œuvre de ce flot est de démontrer comment des modèles à haut niveau d'abstraction peuvent être utilisés pour rapidement explorer l'espace des solutions. Le flot de DSE comprend 3 étapes:

(1) Explorer et sélectionner les clusterings optimisés du NN considéré. Bien que le nombre de clusterings soit relativement limité, le nombre de mappings qui peuvent être générés pour chaque clustering est très grand. Une limitation du nombre de clusterings considérés est donc nécessaire. Cela est fait à l'aide de modèles analytiques rapides d'exécution et à l'aide de l'utilisation de l'algorithme de Branch & Bound.

(2) Explorer et sélectionner les mappings optimisés pour chaque clustering. L'espace de conception des mappings est très large, cette étape est donc à nouveau menée à l'aide de modèles analytiques rapides à exécuter et de l'algorithme de Branch & Bound.

(3) Évaluer les mappings sélectionnés à l'aide du flot de modélisation proposé basé sur la simulation. Ce dernier permet une évaluation rapide et précise des mappings, et ainsi de les classer par temps d'exécution et énergie croissants. La liste des mappings triée de cette façon est retournée à l'utilisateur. Notre flot offre également la possibilité de générer le code C permettant l'inférence des mappings sélectionnés.

Nous appliquons notre flot de DSE à 5 NNs différents et discutons les résultats. Nous comparons les résultats de notre flot vis à vis de l'exploration exhaustive pour un nombre de cœurs réduits afin d'évaluer sa capacité à trouver les mappings les plus optimisés. Les résultats mis en évidence par le flot de DSE proposé permettent de répondre à la question de recherche **(III)**.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to all those who have contributed to the completion of this doctoral thesis. This journey has been as challenging as rewarding, and I could not have succeeded without the support and encouragement of numerous individuals. First and foremost, I am profoundly grateful to my advisors at Nantes Université, Prof. Dr. Sebastien Pillement and Dr. Sebastien Le Nours, for their guidance, expertise, trust and patience throughout the entire PhD program. Their insights and feedback have been invaluable, shaping not only the content of this thesis but also my development as a researcher. I extend my sincere appreciation to Dr. Kim Grüttner, Dr. Domenik Helms, and Ralf Stemmer for their ongoing support and feedback throughout the program, and for welcoming me in Oldenburg at the German Aerospace Center (DLR). I extend my thanks to the members of my doctoral committee, Prof. Dr. Matthias Jung, Dr. Angeliki Kritikakou, Prof. Dr. Gregor Schiele and Prof. Dr. Frédéric Pétrot for their constructive criticism and insightful suggestions when reviewing my dissertation and during the PhD defense. The collaborative environment at the IETR lab and DLR has been a crucial aspect of my research experience, and I am grateful for the generosity and professionalism demonstrated by my colleagues. In particular I express my gratitude to Sandrine Charlier, Marc Brunet and Inge Kuper for their administrative and technical support.

Heartfelt appreciation goes to my father Rodolphe, my mother Sandrine, my sister Oriane, and all other members of my family for their unwavering support and understanding throughout this academic journey. Their encouragement has been a constant source of strength, and I am grateful for their belief in my abilities. Last but not least, I extend my gratitude to my great friends and colleagues, for their encouragement, camaraderie, and occasional distractions that provided much-needed breaks during intense periods of research: Rafal, Romain, $(\text{Alexis D.})^2$, Gaby, Gaël, Safouane, Hai Dang, May, Tamar, Guillaume, Antoine, Corentin, Nolwenn, Juliette, Reem, Fatima, Oriane, Jules, Gourav, Armel, Filippos, Magat, Katerina, Angela, Sony, Benedek, Sat, Linda, Ishan, Adrian, Avinaash, Patrick, Jan, Thomas, Bewoayia, Jannick, Rolf, Georg, Henning, Frank, Gregor, Bernd... I dedicate this work as well to all those I couldn't mention, who have contributed in one way or another to making this experience so enriching.

To my friend Simon

Chapter I

Introduction

I.1 Context

I.1.1 Artificial Intelligence (AI) and Neural Networks (NNs)

The interest for AI has grown tremendously in the last decade. As shown in the 2023 AI Index Report published by Stanford Institute for Human Centered-AI (HAI) [1], the total number of publications on the AI topic has more than doubled since 2010. AI is a concept first introduced in 1956 [2] to designate the simulation of human intelligence in machines, allowing them to perform tasks that typically require human intelligence. Products that implement AI have become part of our daily lives, as in 2022, one-third of the US consumers owned a smart speaker [3], that integrate virtual assistants used to perform home automation actions using AI-driven key word spotting in human speech [4]. Since the 2010s, we've been experiencing the third wave of AI:

1. The first wave of AI took place between the 40s and 60s. In 1943, the behavior of human brain neurons was described for the first time by two psychologists in [5] as a logical model. In the late 50s, the psychologist F. Rosenblatt [6, 7] did the first hardware implementation of a single neuron called the Perceptron. Many consider the work of F. Rosenblatt as the first record in human history of a machine that implemented human-like learning process, through trial and error. This subset of AI was later called Machine Learning (ML) [8]. It includes all algorithms which are not programmed directly to solve a specific problem, but rather to learn by themselves how to solve the problem. During this first wave, AI algorithms remained however relatively simple, as it was impossible to train complex algorithms without advanced training methods.
2. The second wave of AI in the 80-90s was triggered by the introduction of back-propagation learning algorithm [9]. It rendered possible the adjustment of weights from all layers of Multi-Layer Perceptrons (MLPs), thus rendering the training of

more complex NNs possible. MLPs are NNs that are constituted of several layers of percetrons (aka neurons). Using this new learning approach, Y. Le Cun introduced in 1989 the first Convolutional Neural Network (CNN) [10] called LeNet. CNNs use first a set of convolution and pooling layers for extracting data from input images. The results of these layers are then passed to a MLP to classify the data. Y. Le Cun showed that CNNs could be used reliably for the recognition of hand written characters, and introduced several versions of the LeNet as well as the handwritten database MNIST [11]. However, due to the lack of computational power of hardware in this era and due to the lack of training data, more complex AI applications were still impossible.

3. The third wave of AI started around 2010 and is still ongoing. It was rendered possible by the fast evolution of hardware platforms, GPUs in particular. This evolution of hardware enabled the training of much more complex NNs such as AlexNet [12], a CNN featuring 62 million of parameters, against 60 thousand for LeNet5. AlexNet started the third wave of AI by being the first winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [13] [1] in 2012. The third wave of AI was also enabled thanks to such contests and the massive sharing of data-sets on the Internet on websites like Kaggle [2].

NNs have become the reference ML algorithm nowadays to handle data classification and regression problems. Their shape is inspired by biological NNs, in which large sets of neurons are interconnected. Overall, NNs have revolutionized the field of ML and enabled the development of a wide range of applications across various domains by providing a flexible, data-driven approach to problem-solving that complements classical programming techniques. They show high performance for pattern recognition and generalization in large data-sets, are inherently capable of modeling non-linear functions, which are involved in many real-world problem, and show high adaptability due to their resilience to noise and variability in tested data and their ability to learn. NNs are applications with a high degree of parallelism. They mainly have two types of parallelism [3]:

— Intra-layer parallelism: NN layers are composed of neurons. The neurons are inde-

[1] ILSVRC is held yearly since 2010, and makes software programs compete to reach the highest classification accuracy on the ImageNet data-set introduced in 2009 [14]. ImageNet contains 1000 different classes of objects organized in sub-trees, with numerous images for each class.

[2] `https://www.kaggle.com/`, last accessed: 27.09.2023.

[3] Some types of NNs have other forms of parallelism due to the nature of the calculations carried out in the layers. This is discussed, for example, in [15].

pendent of each other and can be executed in parallel.

- Inter-layer parallelism: although consecutive layers depend on each other, it is possible to run different iterations of the NN simultaneously, in order to activate a *streaming* (or in other words *pipeline*) execution of the NN.

Leveraging the two forms of parallelism is necessary to optimize the execution time and consequently the energy of NNs. This is especially important for the execution of NNs on embedded platforms, which have strong timing and energy constraints.

I.1.2 Internet of Things (IoT) and TinyML

The important growth of the Internet-of-Things (IoT) field comes with the need for smart applications using NNs. The number of connected devices by 2030 is expected to reach 29.4 billion [3]. This would represent a 350 % increase in the ongoing decade, as their estimated number in 2019 was 8.6 billion. Usually NNs are executed at the cloud level of IoT applications, as it provides sufficient amount of resources to support fast and efficient execution. However, as shown in Figure I.1, executing NNs in the cloud requires transmitting the data harvested by sensors at the edge through the fog level up to the cloud (❶-❷). The results of the NN executions must then be transmitted again to the edge through the fog (❸-❹) to finally be returned to the user (❺). Current trends aim at bringing the AI algorithm at the edge and thus removing transfers ❶ to ❹ [16]. Deploying NNs at the edge represents an opportunity for improvement in many areas:

1. Response times with effect on latency (execution time) and throughput (number of processed data per second), as data transmissions are costly in terms of time.
2. Power and energy consumption improvements, as data transmissions between the different levels of IoT applications as well as the processing on the cloud are energy intensive.
3. Security, as transmitted data can be intercepted.
4. Alleviation of data privacy issues, as data harvested by sensors can be stored on the cloud and utilized without user's knowledge,
5. Reduced bandwidth problems, as connectivity to the network vary depending on the location and cannot be always ensured. Bandwidth shows also limitations for transferring heavy data packets.
6. In line with the points mentioned above, for safety-critical systems, deployment

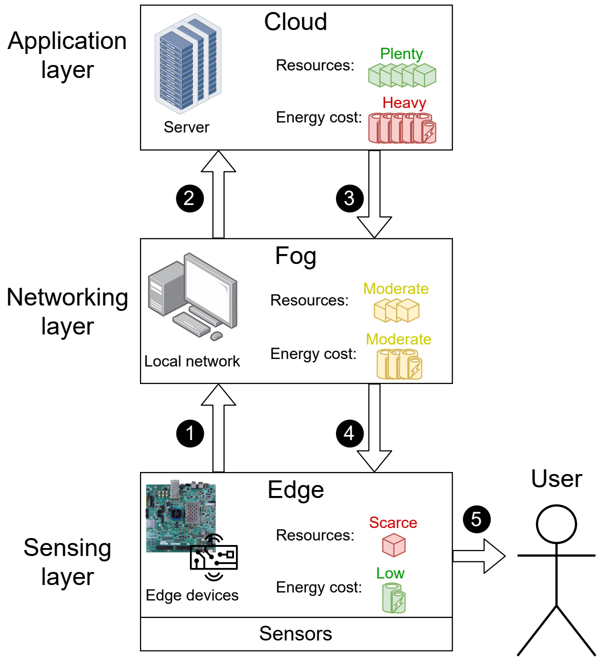


Figure I.1 – Typical organization of an IoT application. At the edge, data is read from sensors. Usually the data is sent through the fog to the cloud (transfers ❶ and ❷), where it is processed using AI algorithms. The results of the AI algorithm are then transmitted through the fog to the edge (❸ and ❹) to be returned to the user (❺). This figure was inspired by work presented in [17, 18].

at the edge is an absolute necessity to satisfy the stringent requirements of these applications, e.g. real time constraints.

We mainly focus on points 1,2 and 6 in this work.

I.1.3 Available platforms at the edge

Various types of embedded platforms are available at the edge:

1. MicroController Units (MCUs): MCUs are low-power, cost-effective embedded systems with limited processing power and memory. Notable MCU chip providers are ST Microelectronics, NxP Semiconductors and Microchip. Other providers such as GreenWaves Technologies have emerged to offer specialized MCUs for AI processing.
2. MultiProcessor Units (MPUs): MPUs often offer more computing power than MCUs, as they integrate multiple processing cores, GPU, memory, and peripherals onto a

single chip. They offer a good balance of performance and power efficiency. Notable examples include the GreenWave Technologies GAP9 SoC[4], the NVIDIA Jetson series, such as the NVIDIA Jetson Nano[5], which is often used in academic work [19, 20], and the Qualcomm Snapdragon Series.

3. FPGAs (Field-Programmable Gate Arrays): FPGAs are programmable hardware devices that can implement logical circuits. They offer high parallel processing capabilities, making them attractive to implement hardware accelerators for NN inference [21, 22, 23]. They also offer versatility due to their re-programmable nature. Although FPGAs are sometimes used at the edge, their overall higher power consumption as well as the important timing and energy overheads caused by their reprogramming [24, 25] make them arguably more appropriate to be used in the cloud. AMD and Intel (formerly Xilinx and Altera) are leading FPGA manufacturers. It is worth noting that some products, called Multi-Processor SoCs (MPSoCs), integrate a FPGA as well as a multi-core system onto the same chip e.g. AMD Zynq7000 and UltraScale series, and Intel Cyclone series.
4. Edge AI accelerators: New chips have also emerged specifically for the processing of NNs. Notable examples are the Intel Movidius Myriad aka Neural Compute Stick, used in many academic works [26, 24, 27], the NVIDIA EGX Edge AI platform and the Meta Training and Inference Accelerator (MTIA)[6].

Most of the aforementioned platforms are based on multi-core systems, with the exception of the majority of MCUs, which are single-core. FPGA-based MPSoCs are multi-processor, edge AI accelerators feature clusters of processing units, which executes very similarly to multi-core platforms, the NVIDIA Jetson Nano features 4 ARM A57 cores and the GAP9 has 10 RISC-V cores. Multi-core systems are quite naturally at the heart of all these platforms, because they can perform calculations in parallel, which is a key feature for the efficient execution of NNs. Multi-core platforms are thus a promising implementation target for NNs. Figure I.2 gives a general schematic of the architecture of multicore platforms. They contain several processing cores, which have their own private memory

[4] The GAP9 features 10 RISC-V, 9 for AI acceleration, and 1 used as a controller. `https://greenwaves-technologies.com/gap9_processor/`, last accessed 09.10.2023.

[5] The NVIDIA Jetson Nano features 4 ARM A57 processing cores and a 128-cores NVIDIA Maxwell GPU, `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`, last accessed: 09.10.2023.

[6] The MTIA, as presented by Meta in May 2023, is an Application-Specific Integrated Circuits (ASIC) dedicated to AI acceleration. It is based on an array of 64 PEs connected via a mesh network. `https://ai.meta.com/blog/meta-training-inference-accelerator-AI-MTIA/`, last accessed: 09.10.2023.

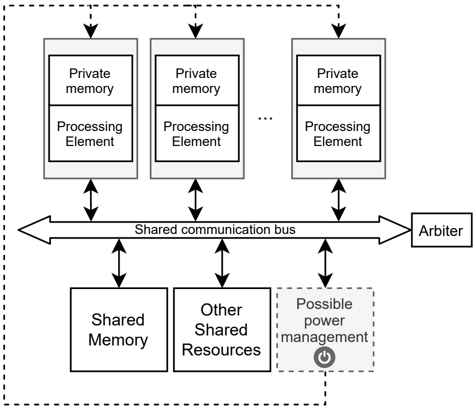
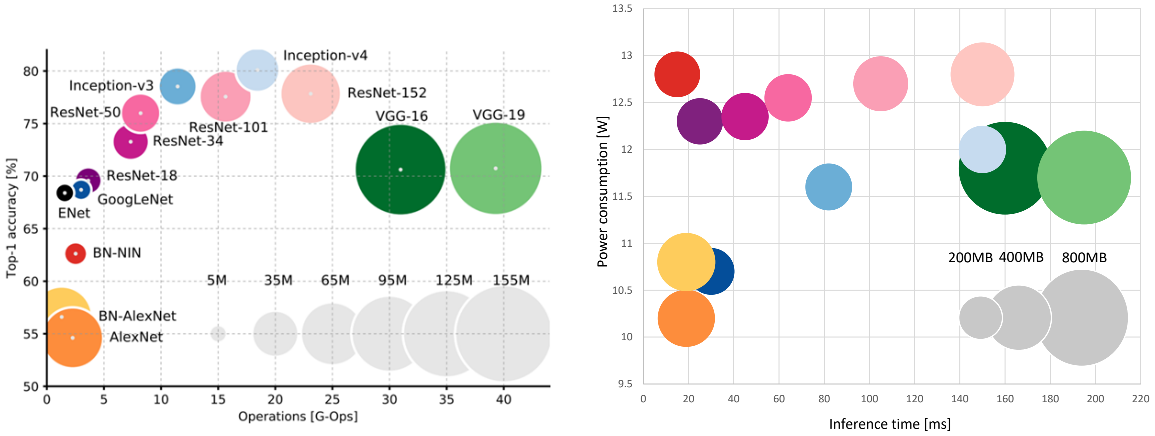


Figure I.2 – Schematic of generic multi-core platform architecture



(a) CNNs sorted by operations, classification accuracy and number of parameters

(b) CNNs sorted by inference time (ms), power consumption (W) and memory cost (MB)

Figure I.3 – Plots sorting notable CNN classifier architectures from the state of the art. Graph (a) shows commonly used quantities to classify NNs. Graph (b) shows preferred quantities to classify NNs deployed onto edge devices. The data presented in this figure was obtained through the implementation and measurement of the CNNs on a NVIDIA Jetson TX1. These results come from the paper [28] (please refer to this paper for more information about the presented CNNs).

for data and instructions. They also include a shared communication bus, that allow cores to access shared resources such as shared memories used for communication between cores. They can also integrate power management to optimize their power consumption in runtime.

I.1.4 NN deployment on embedded platforms

The deployment of NNs on embedded platforms is difficult. Platforms available at the edge have limited processing and memory resources as well as strict timing and energy budget, whereas NNs are computation and memory intensive. Optimizing the resource use, latency and energy without degrading the classification accuracy represents a major but necessary challenge. Around this problematic, a number of communities have sprung up, including the TinyML (for Tiny Machine Learning) Fundation [7], which aims to bring together researchers and industrials working on the implementation of ML algorithms on ultra-low power and resource-constrained platforms. An important trend of TinyML aims at proposing workflows for the evaluation and benchmarking [8] of NN deployments. In TinyML, the priority is put on the optimization of non-functional properties: timing, energy and cost in regards to resource usage, especially memory, and secondarily on functional properties (classification accuracy) [29]. As shown in Figure I.3, the position of the regarded NNs can differ (in particular GoogLeNet, BN-NIN and Inception-v4) based on the plot (a) or (b), (a) representing usually considered metrics to evaluate NNs, and (b) representing metrics that matter in the scope of TinyML.

It is necessary to perform an evaluation of NN deployments under timing and energy constraints to find solutions that meet the constraints imposed on the system. Several evaluation flows for edge NN deployment have been proposed. They can be divided in two categories:

1. Rapid prototyping: these approaches focus on systematic implementation and characterization of NNs on a real platform. Notables examples are: [26, 19, 29]. The two main drawbacks of these approaches is that:
   (a) they do not allow an evaluation of candidate solutions early in design phases, prior to the deployment on the real hardware.
   (b) they require an important characterization effort as each configuration must be systematically compiled, deployed and tested.
2. Modeling flows: these approaches rely on the building of models to predict timing and energy properties of candidate solutions. Due to the abstraction necessary to establish the models, they are less accurate than rapid prototyping. However, they

[7] `https://www.tinyml.org/`, last accessed: 01.10.2023

[8] See for example the ML Commons Inference Tiny Benchmark: `https://mlcommons.org/en/inference-tiny-10/`, Last accessed: 01.10.2023

require less evaluation effort and can be performed early in design phases. Notable examples are [23, 30, 31, 32].

Instead of rapid prototyping, models with fast execution time and highly confident prediction are preferable to quickly identify optimized solutions. State-of-the-art modeling approaches for NNs deployed on edge devices show however limitations in fully covering the scope of NNs on multi-core platforms. We plan in this work to propose adapted models for such platforms.

On multi-core platforms, several aspects render the evaluation of non-functional properties tedious. The proposed models must be scalable, i.e. must have good and invariable prediction accuracy and evaluation time, in regards to these aspects. The models must also offer scalability to all aspects simultaneously, which adds to the complexity of the task.

1. The different sources of parallelism: NNs have different sources of parallelism (intra and inter-layer). Leveraging these sources of parallelism is necessary to enhance non-functional properties. However, the proposed models must allow exploring the effect of leveraging both parallelisms simultaneously.
2. The contentions for shared resources: on multi-core platforms, contentions occur when multiple processing cores try to access shared resource such as bus and memory simultaneously. Contentions lead to important timing overheads, with consequence on energy, and it is necessary to properly model them.
3. The architecture of the platform: the number and types of components (cores, memory, bus, peripherals) have an important effect on timing properties and power consumption, and must be properly modeled.
4. The dynamic behavior of the system: on multi-core platforms, cores execute different phases (computation, communication). When using power management techniques, such as clock gating [33], cores may additionally be enabled or disabled (i.e. clock gated) over time. The effect of dynamic behavior and power management on energy but also on timing, as the enabling of low power modes can introduce timing overheads, must be properly accounted for in the models.
5. Different computation/communication workloads linked to the input NN: NN classifiers differ in architecture. They can have different numbers, types and size (number of neurons) of layers. The modeling flow must provide accurate predictions in regards to the different computation/communication workloads of NNs.

The design space for NNs deployed on multi-core platforms is tremendous. In addition to having a fast yet accurate prediction flow for timing properties and energy, a Design Space Exploration (DSE) flow must be proposed to efficiently use the models to discard lesser solutions while selecting most optimized ones. The design space of NNs deployments is indeed very large, and evaluating every possible solution is not feasible, even with very fast models. The DSE flow is also necessary to find a solution that meet the strong constraints of edge platforms. The DSE flow must allow two different objectives:

1. Finding deployments of NNs onto a user specified fixed multi-core platform,
2. Jointly design the hardware platform's dimensions and software deployment of the NN.

I.2 Research challenges

The presented work aims at answering the following research questions:

1. How to provide fast yet accurate evaluation early in design phases of timing and energy properties for streaming NNs deployments on multi-core platforms?
2. Is a model-based approach more relevant than rapid prototyping?
3. Is a model-based approach suited for early, fast and confident Design Space Exploration (DSE) of streaming NNs deployments on multi-core platforms?

I.3 Contributions

To address these research challenges we propose the modeling flow depicted in Figure I.4. On this figure, the contributions regarding early evaluation of NN mappings onto multi-core platforms are highlighted in orange. They are the following:

1. A hybrid timing modeling flow, which relies on analytical models, characterization through measurement, and simulation. This combination of modeling approaches allows delivering fast yet accurate timing predictions with scalability in regards to the aforementioned challenging aspects,
2. A power and energy modeling flow, which offers accurate power predictions with scalability considering dimensions of multi-core platforms. The power model uses the simulated execution traces issued by the timing modeling flow in order to predict power and energy.

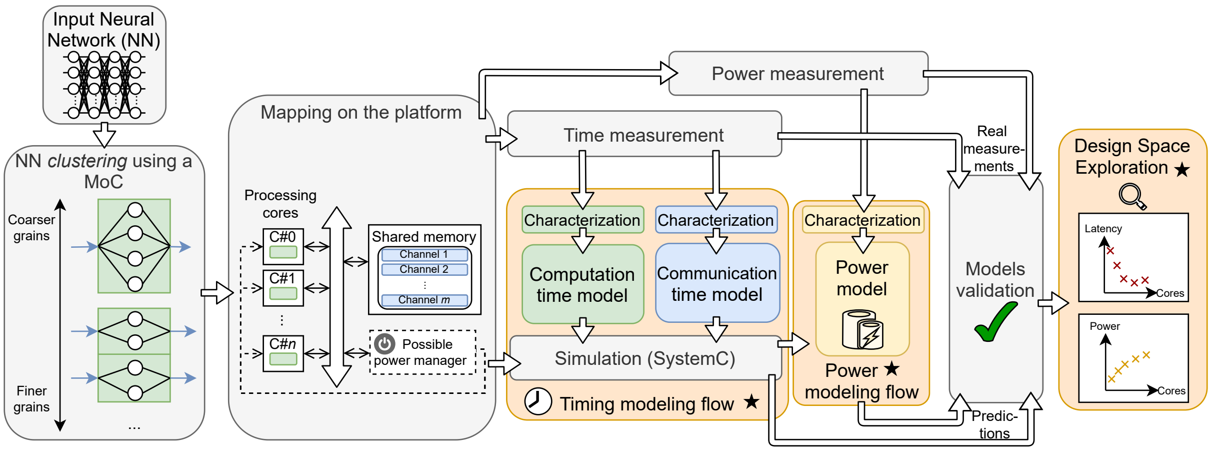


Figure I.4 – Proposed modeling flow for the prediction of timing and power properties of NNs deployed on multi-core platforms. The three main contributions of this work are depicted in orange.

3. An efficient DSE flow for high level models. The flow first prunes the vast design space through the use of pure analytical models with very fast evaluation time and Branch and Bound algorithm to progressively optimize solutions and disregard sub-optimal branches. The second phase aims then at evaluating the selected mappings using the proposed timing and power modeling flow to provide more accurate and reliable evaluation but on a limited number of solutions.

The implemented flow and experimental results presented in this manuscript are available on our GitLab repository: `https://gitlab.univ-nantes.fr/lenours-s/pssim4ai`.

I.4 Organization

We first provide in Chapter II a review of the published literature in the scope of evaluation and DSE for NNs deployed onto edge devices. The provided review justifies the contributions of our approach.

We then present in Chapter III the technical background and work hypotheses under which this work is conducted. We present the types of NNs considered, the dataflow-oriented Model of Computation (MoC) used to describe their execution, our Model of Architecture (MoA) and how applications are mapped on platforms compatible with our MoA. The implementation of a platform prototype for model calibration and validation is also presented, including details about the neural networks used for validation.

In Chapter IV, we introduce our primary contribution, which is our timing prediction and analysis flow as shown in Figure I.4. Our methodology combines measurements, analytical models, and simulations to offer fast yet accurate predictions of timing properties with scalability in regards to various factors: NN parallelism expression, core usage, computation and communication workload and type of communications. Our models are validated against real timing measurements of 54 different mappings of 4 NNs and show an accuracy of more than 97 % on end to end latency and throughput, for an evaluation time per mapping of 20 s. We also provide a comparison of our modeling flow against pure analytical models with respect to related work.

In Chapter V we present the power and energy prediction flow. It leverages the simulation from our timing modeling flow as well as analytical models calibrated through measurements, to accurately model power and energy consumption. It offers flexibility for optimizing NN mappings on fixed platforms in regards to the identified challenging factors, as well as allowing conjointly optimizing platform dimensions and NN deployment. Similarly to the timing models, we validated our power and energy prediction flow on the 54 NN mappings, and they show an accuracy 93 % and again an evaluation time of 20 s for both energy and timing prediction. We also validate the scalability of our models regarding platforms of different dimensions.

We present the third and last contribution of our work in Chapter VI. In this chapter, we introduce a Design Space Exploration (DSE) flow that utilizes the models proposed and validated in Chapters IV and V. This DSE flow allows for the optimization of NN mappings on multi-core platforms while adhering to user-defined constraints for both timing and energy properties. It provides efficient pruning of design space using analytical models and subsequent accurate evaluation leveraging the proposed simulation-based modeling flow. We test the DSE flow on different NNs and evaluate its limitations.

The Chapter VII provides our conclusions regarding the research work presented in the manuscript. It also identifies prospects on interesting directions that could be led to extend the scope of the proposed flow in future work.

Chapter II

Related work

This chapter provides a review of the research work in the field of evaluation and optimization of NNs deployed on edge devices. First we provide a review of approaches focused on the evaluation of NN deployments regarding timing properties and energy. Then we provide a review of optimization techniques used in the field of embedded systems development.

II.1 Evaluation of NN deployments on edge platforms

As shown in Figure I.3 in Chapter I and as discussed in [29], the focus of works regarding the deployment of NNs on edge devices is put on optimizing non functional properties such as inference time and energy, without sacrificing *Quality of Service (QoS)*. In this chapter, when using the term *QoS*, we refer to the NN's classification/regression accuracy and other relevant functional properties dependent on the application. The non-functional properties often considered for the optimization of edge NN implementations are the following:

1. **Execution time**, also called inference time, which designates the duration between the start and the end of NN execution.
2. **Throughput**, which designates the number of inputs a NN deployment can process per a quantity of time. High throughput is particularly important for applications focused on the processing of a streaming flow of data, e.g. key word spotting [4].
3. **Power consumption** in Watts, which refers to electrical energy consumed by the system per second. Optimizing the power consumption is necessary as edge devices are often operating using batteries, which have limited supply and must thus be preserved. Power consumption in electronic systems is split into [33]:
   - **Static power consumption**, which is due to the leakage current between the source and drain of Field Effect Transistors (FETs) used to implement electronic circuits. This current is observed regardless of the state of the transistor, as long

as the circuit is supplied in power. For this reason, static power consumption is usually obtained by measuring the power consumption of a circuit when not clocked.

- **Dynamic power consumption**, which is principally due to the switching of FETs, when the circuit is operating in normal conditions.

4. **Energy**, which corresponds to the power consumption of the system integrated over time. For NN evaluation, the quantity that is typically considered is the energy cost per inference.

5. **Resource usage**, which is usually considered for cost reasons, as using more components in a circuit design leads to higher cost of production. For NNs, what is often regarded is **memory usage**. NNs are indeed memory intensive, and reducing memory cost can be necessary to enable the execution on devices that lack such resources. It can also positively impact latency and power, as memory accesses in edge devices cause important overheads on these quantities.

Approaches have been proposed for the evaluation of NN deployments in regards of these quantities, in order to find optimized solutions. These approaches can be separated into two main trends:

1. **Rapid prototyping,** which focus on systematic implementation of the targeted hardware and measurement of tested metrics during the execution. These approaches offer the best evaluation quality, but require important characterization effort. We provide a review of such approaches in Section II.1.1.

2. **Modeling flows,** which propose models to predict non-functional properties. The time and effort is lesser than rapid prototyping, at the cost of less confident evaluations. We provide a review of these approaches in Section II.1.2.

II.1.1 Rapid prototyping

Many approaches focus on the evaluation of NN deployments through systematic implementation on a real platform and characterization using measurements. Measuring directly the quantities at test, such as latency and energy, offers the highest possible evaluation accuracy. In Figure II.1 we provide a view of the main steps and process that can be found in most of these approaches. Neural Architecture Search (NAS) engines are often used in these flows during Step ➊. NAS [34] aims at automating the design of NNs in regards to their architectural features (i.e. number of layers, number of neurons inside

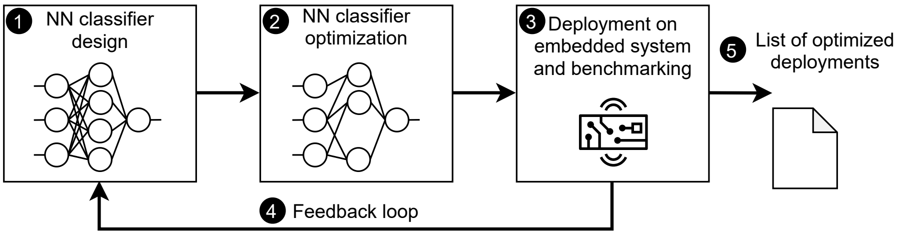

1 NN classifier design
2 NN classifier optimization
3 Deployment on embedded system and benchmarking
5 List of optimized deployments
4 Feedback loop

Figure II.1 – Diagram showing the main steps in a measurement-based approach aiming at evaluating NN deployment on embedded platforms. This diagram takes inspiration from [29].

layers). The design is done in such a way to optimize functional (classification accuracy) and non-functional (timing, energy) properties. In the reminder of this chapter, we will refer to classifier accuracy as *Quality of Service (QoS)*, as in [19]. In step ❷, compression is applied to the NN model. The compression of NNs constitutes an entire field of research that aims at proposing methods to reduce the memory footprint (number of weights) and the computation/communication workload of NNs while minimizing the loss in NN classification accuracy [35, 36, 37, 38]. The NN is then trained, compiled and deployed on the real embedded platform, on which perceptible metrics such as timing properties (latency, throughput) and energy are measured ❸. The measured data is then fed forward back to ❶ in order to guide the optimization of the NN. Deployments that meet user defined constraints are then returned to the user ❺.

The approach in [29] provides an exploration flow for NN deployments under QoS, latency and energy constraints. It focuses on the measurements of tested quantities after the implementation of NNs on a ST Microelectronics MCU. The presented flow corresponds to the one in Figure II.1. This paper provides intermediate results, showing the effect of using certain ISA (Instruction Set Architecture) extensions such as Floating Point Units (FPUs) on the tested quantities.

The authors in [19] propose an evaluation and exploration flow for NNs deployed on NVIDIA Jetson TX2 GPU under timing, energy and QoS constraints. It respects the phases presented in Figure II.1. NAS is performed with a feedback loop with QoS, timing and energy measurements obtained after training and implementation on the GPU. Experimental results show up to 79 % reduction in energy consumption and 36 % in inference time, with a loss of QoS up to only 2 %, against the default implementation.

In [39], and exploration flow for CNN classifier architectures for semantic segmentation of aerial images with QoS and timing constraints is presented. The proposed methodology takes the specifications of the hardware target into account when performing NAS, to ensure hardware architecture and algorithm adequation and consequently improve performance. The experiments were conducted on a NVIDIA V100 Tensor Core GPU [1] even though the approach can arguably be transposed to edge devices. The results show that NN complexity could be reduced by 88 % with significant gains in performance and memory use and minimal loss of QoS (3 %), against the default implementation.

The authors of [26] propose a DSE flow to find CNN deployments that optimize latency and energy on Intel Movidius Myriad 2 (USB Neural Compute Stick). The Myriad is instrumented to obtain timing and energy measurement used to rank possible deployments. While this approach does not support NAS, it puts the emphasis on exploring the clustering, mapping and scheduling of the NN on the targeted hardware unlike [29, 19, 39]. In our approach we will also highlight that the exploration of NN partitioning, mapping and scheduling using a Model of Computation (MoC) is a key element to properly take advantage of NN intra- and inter-layer parallelism with latency and energy enhancements.

The approach in [20] focuses on exploring NN deployments on the heterogeneous NVIDIA Jetson boards, which feature a 4 ARM cores MPU and a Maxwell GPU. Similarly to [26], the emphasis is not on NAS but rather on the optimization of hardware implementation. To this end, this approach evaluates the effect of modifying the clock frequency of the devices. It offers up to 66.3 % performance improvement compared to the default implementation, with a reduction of power consumption of up to 61.5 %.

As a general criticism, exploring NN deployments through measurement requires an important effort as numerous mappings must be deployed and tested on the real platform. When also considering NAS, the training and compression of the NN must also be systematically done (steps ❶ and ❷ of Figure II.1), which takes additional effort. For this reason such approaches do not focus on the exploration of different levels of granularity for the partitioning/mapping of the NN. We will show in our approach that this is an important aspect to consider for the optimization of NN deployments. Evaluation through measurement also requires having fixed the real implementation platform, which restrict the possibilities in regards to architectural exploration. To avoid these drawbacks, other approaches propose models that ease the exploration of the design space, at the detriment of a loss in evaluation accuracy.

[1] This device is found at cloud level of IoT applications - see `https://www.nvidia.com/en-us/data-center/v100/`, last accessed 19.09.2023.

II.1.2 Evaluation using models

Modeling flows for NNs deployed on edge devices are often characterized through linear regression [40, 41] of timing and power measurements. They are mainly based on analytical formulas. The timing models are, for the most part, derived from operations executed inside NN layers. The power models describe the static power consumption of the circuit when the clock is disabled, and the dynamic power consumption of NN computation phases on the platform and memory accesses. The average power consumption of the system is predicted and integrated over the estimated time to acquire the energy. These models are said to be of *high level of abstraction*, which means that they abstract numerous details regarding the implementation of the NN to offer simplified formulas with fast evaluation time [42].

The approach in [30] focuses on the characterization of analytical timing and energy models by measurement on two edge multi-core platforms, by manually varying the number of layer parameters (i.e. neurons for dense layers). Models are derived from measurements using multi-linear regression. The characterization phase is performed for two different compiler settings, which allows obtaining high accuracy with both, as measurements allow modeling the behavior of both compiler and hardware. Both for timing and power, they use a simple model memory access cost (network data retrieval and layer input and output communications between cores). Their models remains at the layer level of NNs, and do not allow the exploration a finer grained deployments issued from the partitioning of layers, as we will explain in Chapter III, Section III.2.2.

AutoDice [32] is a framework for fast exploration of NNs deployed onto multiple heterogeneous edge devices on the same network (e.g. GPUs, MCUs, multi-core CPUs). The exploration is done using high level analytical models for throughput, memory and energy evaluation. The execution time and consequently the energy is established through an estimation of the number of MAC operations executed inside the targeted edge devices to process the NN's layers. The approach does not allow custom partitioning with intermediate granularity for NN layers. The framework allows automatically generating the C code for the execution of optimized NN mappings.

The aforementioned modeling approaches [30, 32] share the inconvenient of not offering strong scalability in regards of the fine grained partitioning of the NN. The models from [30, 32] in their current form are limited for the prediction of non-trivial NN intra-layer parallelism expression and use. In general, this observation can be done for approaches that rely on the most used Deep Learning frameworks featuring a Python API, such as

Tensorflow [43, 44], PyTorch [45], Caffe [46] and Keras [47], and NN [2] compiler for edge inference such as TVM [48] and OpenVino [49]. These tools offer a high level of abstraction to the engineer developing NNs for edge devices, which highly reduce the development effort, but limits the custom deployment possibilities, such as using intermediate partitioning and mapping of NNs. In our approach, we show that non-trivial partitioning and mappings of NNs on multi-core edge devices must be considered to find more optimized solutions.

It is also worth noting that some Deep Learning frameworks and NN compiler tools for edge inference propose a latency and sometimes also energy estimation tool. This is the case for example of TVM [48], TensorFlow Lite Micro [44] and N2D2 [50]. The estimators are however calibrated for dedicated compiler settings and platform specifications, and tend to bear inaccurate estimations for unsupported platforms. As shown in [27], which proposes analytical timing models for NN layers on Intel Movidius Myriad2 while using OpenVino, the compiler selects optimization settings unbeknownst to the developer with tremendous effect on timing, that can hardly be modeled. Building confident models when entirely relying on these tools can be tedious. To alleviate this issue and consequently ease the modeling effort, the authors of [32] train NNs using Deep Learning frameworks and then carry out their deployment using a custom library in C programming language.

The approach in [15] focuses on the design and development of an FPGA-based accelerator for CNNs. The flow allows DSE regarding several sources of parallelism withing these applications: inter and intra-layer, but also inter and intra-kernel for convolution layers. Analytical formulas are proposed for the different computations performed inside layers. Their results show an important speedup of AlexNet's execution [12] compared to the state-of-the-art. This approach is however only focused on timing evaluation.

fpgaConvNet [51] relies on a fine-grained description of CNNs in the Synchronous Dataflow (SDF) [52] Model of Computation (MoC) to optimize their deployment on FPGAs. As shown in the paper [53] by the same authors, analytical models for timing prediction are proposed by using the formulas obtained from the SDF paradigm, complemented by the delays required to perform CNN computations on FPGA. The proposed models have a prediction accuracy of more than 92.9 % on timing properties. The authors use the fast analytical models and transformations of CNN mappings described in SDF to efficiently explore the design space.

Timeloop [54] is a framework used to explore the design space of NN mappings on GPUs and hardware accelerators by evaluating solutions using analytical models. It uses

[2] Often in the Open Neural Network eXchange (ONNX) format.

an analytical model for performance (computation time), area and energy prediction and a mapper (dataflow mapping explorer) to explore mappings on the targeted architecture.

The approach in [55] proposes an optimization flow based on analytical models for timing and energy prediction of CNNs mapped on CPU-GPU MPSoC. The proposed flow allows optimizing power management use in the platform when executing CNNs using voltage and frequency scaling. Their approach offers high accuracy (95 % on timing, 91 % percent on power/energy) on several mappings of three different CNNs from the state-of-the-art executed on the NVIDIA Jetson TX2 board, and allows finding optimized settings for these applications. While this approach explores the mapping/scheduling on the CPU, the GPU provides most of the computational power, with limited effect of shared resource contention, which is not the case on all multi-processor based systems. Also the proposed models in their current form cannot be easily extended to allow exploring the hardware platform's dimensions (e.g. number of cores) under timing and power constraints.

HALF (Holistic Auto machine Learning for FPGAs) [56] proposes a multi-criteria optimization approach for NNs deployed on FPGA. Optimized hardware accelerator for NNs, which are similar to FINN [21]-based solutions, are obtained by exploring computational IPs for NN layers in an IP bank, considering application, algorithm, architecture, and platform criterias. Communications between IPs are implemented using AXI streams. The approach leads to solutions that can outperform the baseline implementation on a NVIDIA Jetson AGX Xavier board. The resulting solutions demonstrate superior performance compared to baseline implementations on a NVIDIA Jetson AGX Xavier board. However, when using AXI stream, the communication time is relatively marginal, and thus in this approach, the timing prediction flow does not model communication times, and while the power consumption of idle IPs (i.e. which are waiting for data) is modeled, the effect on power of communications is disregarded. On multi-core platforms, more versatile communication mediums are preferred, which tend to have non-marginal communication times. On such platforms, contentions also occur when several cores access shared resources simultaneously, which bear important effects on non-functional properties and must be correctly modeled.

NNest [57] is an early-stage design space exploration tool that can rapidly and accurately estimate the area/performance/energy of different NN accelerator architectures, also using analytical models. This approach allows both fine tuning of the hardware architecture and high intra- and inter-layer parallelism use from NNs. It involves the use of compression techniques.

MAESTRO [58] (for Modeling Accelerator Efficiency via Spatio-Temporal Resource Occupancy) is an analytical modeling approach to optimize the timing properties and energy of NNs deployed on hardware accelerators. It offers a data-centric approach that put the emphasis on data re-use, while evaluating the use of the different phases of processing of the NN. The approach uses a MoC to describe NNs.

ZipCNN [31, 59] is based on an analytical modeling flow for energy, latency and memory prediction of NNs deployed on MCUs. It offers scalability in regards to the operating frequency, and help finding the frequency that allow the most savings in timing and power consumption for the inference of NNs. ZipCNN's models achieve an average prediction accuracy of more than 91 % on latency, 93 % on energy, and 96 % on memory for the LeNet5 [11] and ResNet [60] NNs with various operating frequencies.

The aforementioned approaches are however all based on pure analytical modeling flows, which demonstrate very high evaluation speed and accurate prediction of timing properties for NN implementation on hardware accelerators. However they have limited scalability for multi-core platforms due to the possible influence of shared resources (bus, memory). As the number of cores and shared resource contentions increase, the complexity of analytical models grows exponentially, rendering the prediction of non functional properties in a fast yet accurate manner tedious. Simulation can model how cores contend for shared resources, thus providing high scalability in regards to the number of cores and amount of communications on the platform.

The authors of [61] propose a platform-aware NAS flow for NNs implemented using accelerators in the form of NPUs (Neural Processing Units), i.e. arrays of Processing Elements (PEs), on FPGA. This flow uses a NPU Dataflow Simulator to evaluate possible HW/SW deployments of NNs on the platform. However, this simulation approach does not aim at modeling shared resources, but rather the complex dataflow inside arrays of PEs.

NNSim [62] is a SystemC/Transaction Level Modeling (TLM) simulator for NN accelerators using the Eyeriss accelerator architecture [63, 22], which corresponds to an array of PEs interconnected through a Network-on-Chip. Compared to RTL [3] implementation, the proposed model has a prediction accuracy of more than 97 % on timing properties, with 3000 up to 13 000 times simulation speedup. While NNSim enables timing predictions, it aims primarily at providing a fast functional model to help designers perform the verification and validation of NN implementations using the Eyeriss architecture.

[3] The Register Transfer Level (RTL) is an abstraction for electronic system design, which is focused on the level of registers and instruction sets, and more specifically on the dataflow inside synchronous hardware circuits.

The approach in [23] provides a DSE flow to find optimized accelerator architectures for the inference of CNNs, which relies on system-level modeling and IP-based design methodology. NNs are described into the Kahn Process Network (KPN) MoC [64] to model the dataflow between NN layers and strictly separate computation and communication phases. Computations are implemented as functional IPs featuring a set number of PEs and communications are implemented as interconnect IPs. Both are picked from a user defined IP bank. As shown in [65], they use SystemC simulation to model shared resource contention with an evaluation time in the order of seconds for simple NNs up to the order of hundreds of seconds for NNs with larger layers (e.g. input layer containing more than 2 million pixels). The model achieves more than 92 % accuracy on timing prediction.

The main difference between our work and [23, 65] is that they focus on IP search, while we focus more on the fine grained SW implementation of the NN on processing cores. Our models also offer a higher accuracy of 97 % on timing and much faster evaluation time. In the case of our flow, the evaluation time is also observed to be unaffected by the complexity of the NN or the architecture. Our modeling flow also offers power consumption and energy prediction, which [23, 65] does not offer.

The authors of [66] propose a framework aimed at optimizing the mapping of concurrent dataflow applications on heterogeneous multi-core platforms. The flow relies on the description of targeted applications' dataflow in KPN, and predicts non functional properties using simulation and analytical models. It can be used to optimize mappings in regards to several metrics such as timing and memory use. However the proposed flow doesn't allow power or energy prediction. The timing analytical models are calibrated using estimates from RTL simulation of time spent executing different instructions. Several heuristics are proposed to quickly find solutions that optimize user criteria. They show that their approach works on three different applications, and that the heuristic offering fastest convergence towards the most optimized points is highly dependent on the application. When targeting NNs, the models can be simplified, which allows enhancing the evaluation speed without losing prediction accuracy.

In previous papers, our work team proposed a message level timing modeling flow for the streaming execution of SDF graphs on multi-core platforms [67, 68]. The flow is based on SystemC simulation, timing measurement and the use of Statistical Model Checking (SMC) [69, 70]. A communication model for multi-core platforms using AXI interconnect shared bus has been proposed [68], and is still used in this thesis work.

When executing NNs, we have observed marginal execution variability. We thus do not rely on SMC, which allows saving consequential effort, as it requires the analysis of large amounts of data and add complexity to the simulation. Instead, we propose an approach which offers more scalability in regards to NN layer partitioning into actors, without the need to re-calibrate for every generated actor. The evaluation time is decreased to the order of hundreds of milliseconds[4]. In [67] the evaluation time ranges from 1 min 44 s up to more than 25 min in some cases[4]. The prediction accuracy of timing properties has remained in the same order of magnitude, although it is observed to be slightly more accurate and it is validated on a larger number of mappings in this thesis work: 97 % for 54 mappings against 95 % for 6 mappings. In this thesis we also extend the modeling flow to power and energy prediction, and enable its use both for finding optimized deployments of NNs onto a set multi-core platform, and jointly optimize the NN deployment and the multi-core platform's dimensions in regards to core number and memory sizes. We also propose an automatized DSE flow that uses high level models, which is entirely new compared to work led in the past.

Table II.1 summarizes the review of state-of-the-art approaches. In this table, the relevance of approaches regarding different aspects is evaluated using a system of more or less filled circles. Full circles ● correspond to the highest grade, while empty circles ○ correspond to the lowest grade. For example the grade ● for evaluation speed means that the flow is very fast. The grade ⦿ for intra-layer parallelism means that the flow allows scalability regarding this aspect with limited exploration possibilities, whereas the grade ● means that the flow offers complete scalability regarding this aspect. Related to the state-of-the-art, our evaluation flow offers the following contributions:

1. Confident prediction of timing and power properties: The proposed timing and energy modeling flow offers an accurate evaluation of candidate deployments.

2. Scalability in consideration of:
   — NN intra- and inter-layer parallelism expression and use,
   — Modeling of shared resources,
   — Effect of power management,
   — Platform dimensions in regards to the number of cores and memory sizes.

3. Fast evaluation time: The proposed modeling flow offers fast evaluation time. This allows saving crucial effort over other simulation modeling flows like [23, 67] and rapid prototyping approaches like [19, 26].

[4] When only considering the simulation time and excluding compilation time.

Table II.1 – Summary of main features of approaches from the state-of-the-art. In this table, evaluated quantities are provided in initials: QoS designates the Quality of Service (i.e. functional properties, and especially classifier's accuracy), T designates the timing properties (inference time, throughput), E designates the power consumption and energy, M designates memory, and A designates area (relevant for approaches using FPGAs).

Work	HW target	Approach type	Evaluated metrics	MoC	Evaluation speed	Evaluation Accuracy	Considers shared resources	Inter-layer parallelism	Intra-layer parallelism	Power management	HW dimensions
[29]	MCU	R.P.	QoS, T, E	×	⊙	●	N.A.	N.A.	N.A.	○	N.A.
[19]	GPU	R.P.	QoS, T, E	×	⊙	●	N.A.	◉	●	○	○
[39]	GPU	R.P.	QoS, T, E	×	⊙	●	N.A.	◉	●	○	⊙
[26]	VPU	R.P.	T, E	×	◉	●	N.A.	●	●	○	⊙
[20]	Multicore + GPU	R.P.	T, E	×	◉	●	N.A.	◉	⊙	●	◉
[30]	Multicore	Analytical	T, E	×	●	◍	⊙	●	⊙	○	◉
[32]	All	Analytical	T, E, M	SDF	●	◉	⊙	●	⊙	○	◉
[43, 44, 48, 50]	All	Analytical, estimates	T, M	×	●	⊙	○	◉	◉	○	⊙
[27]	VPU	Analytical	T	×	●	⊙	⊙	◉	◉	○	⊙
[15]	FPGA	Analytical	T	×	●	N.C.	⊙	●	●	○	●
[51]	FPGA	Analytical	T	SDF	●	N.C.	⊙	●	●	○	⊙
[54]	FPGA, GPU	Analytical	T, E	✓	●	⦿	⊙	●	●	○	⦿
[57]	FPGA	Analytical	T, E, A	✓	●	⦿	⊙	●	●	○	●
[58]	FPGA	Analytical	T, E, M	✓	●	◍	⊙	●	●	○	●
[31, 59]	MCU	Analytical	QoS, T, E, M	×	●	⦿	⊙	N.A.	N.A.	●	N.A.
[71]	Multicore, GPU	LUT	E	×	●	◍	⊙	N.A.	N.A.	●	◉
[61]	NPUs	Simulation	T, E	✓	◍	N.A.	⊙	●	●	○	◉
[62]	SoC	Simulation	QoS	×	◍	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
[23, 65]	GPU	Measurement	T, A	KPN	◉	⦿	●	●	●	○	●
This work	Multicore	Hybrid	T, E	SDF	◍	◍	●	●	●	●	●

II.2 Design Space Exploration (DSE)

To reduce the time spent in exploration of NN deployments onto edge devices, engineers and researchers use optimization algorithms. The design space of NN deployments on edge devices is tremendous. As presented in [72], optimization algorithms are commonly used in the field of embedded system development to quickly converge towards solutions that optimize multiple intertwined properties. As shown in the previous section, NN deployment optimization focus on properties such as QoS, timing quantities (execution time, throughput), power and energy, and resource usage (area, memory). The optimization algorithms can be separated into two main categories:

1. Exhaustive search algorithms, that allow performing an enhanced search of the design space usually by pruning less optimized design points. These algorithms always converge towards the optimal solution, but they are computation-intensive and have a higher evaluation cost than their counterpart. In general, exploration with these algorithms is done with a time limit. Once the time limit has been reached, the results are returned to the user. However, the user still has the option of continuing the search, until the entire design space has been explored. Notable examples of such algorithms used in our field are:
   - Branch & Bound (BB) [73]. A notable example of the use of the BB algorithm to optimize DSE for streaming applications on embedded systems is [74].
   - Integer Linear Programming (ILP) [75], used for example in [76].
2. Heuristics search algorithms, that allow searching the design space using heuristics and randomly generated solutions. They perform usually faster searches than the exhaustive ones, but they have the drawback of converging towards local optimum and not always global ones. Notable examples of such algorithms are:
   - the Genetics Algorithm (GA) [77]. It is the most prominent optimization algorithm for DSE of NN deployments on edge devices. It used for example in [72] for SDF graphs deployed onto MPUs, and in [61, 32] for the deployment of NNs on edge devices. The GA is often paired with the use of a MoC (e.g. SDF) and the introduction of an encoding of NN mappings on the platform. Mappings undergo successive mutations inspired by those of chromosomes and genes observed in the living world over generations. Mutated mappings that optimize tested metrics are selected, and the proposed methodology allows eventually converging to an optimized mapping.

- Tabu search [78],
- Simulated annealing. [79] uses for example the simulated annealing to help optimize NN classification accuracy in a NAS setup.
- Ant colony optimization approach [80], e.g. [81],
- Iterative racing (iRACE) [82], e.g. [83].

In our approach, we focus on the optimization of timing, energy and cost (in regards to the number of cores used and private memory sizes). Our contribution regarding the DSE aspect is to provide a demonstration of an efficient DSE flow using high level models with high scalability for NNs deployments on multi-core platforms. The proposed flow uses the Branch & Bound algorithm to quickly optimize intra and inter-layer layer parallelism use. Our search stops when user constraints are satisfied. Since the BB algorithm is exhaustive, the user always has the possibility to run additional iterations to try finding more optimized solutions. We explain in Chapter VI how we use the BB algorithm in our work. Our models are built to offer scalability to the platform's dimensions regarding the number of PEs and memory sizes. For this reason, the proposed DSE flow can be used for two objectives:

1. Finding deployments of NNs onto a user specified fixed multi-core platform,
2. Jointly design hardware platform's dimensions and software deployment of the NN.

Chapter III

Work hypothesis

This chapter aims at presenting the hypothesis we formulate in order to delimit the scope of our study. We first present the types of NNs that have been considered. We then present how we model NNs in a dataflow-oriented Model of Computation (MoC) to ease their analysis and optimization process. The following sections present what hypotheses we formulate about our Model of Architecture (MoA), and how we map the applications described in the MoC onto platforms that subscribe to our MoA. The final section of this chapter presents the implementation of a platform that respects our MoA and includes timing and power measurement infrastructures, which are used for the calibration and validation of our models. This chapter also presents which NNs have been used and how they have been trained and deployed onto the platform in order to perform the validation of the modeling flow.

III.1 Considered types of NNs

As presented in the introduction of this manuscript, many different NN algorithms have emerged with the raise of interest for AI. In this work we consider two of the main algorithms considered in both industry and academia: the Multi Layer Perceptrons (MLPs) [6, 7] and the Convolutional Neural Networks (CNNs) [10]. MLPs are composed of only one type of layer: the dense layer, also called fully-connected layer, which allows classifying data. In addition to dense layers, CNNs also contain convolutional and pooling layers generally placed before the dense ones. These additional layers extract features from the input data to improve the classification process of dense layers. CNNs can also contain other types of layers such as normalization layers and branch and merge layers. In this work, we choose to focus on convolutional, pooling and dense layers. An example of MLP is shown in Figure III.1 and an example of CNN is provided in Figure III.2. Our approach focuses on evaluating the performance and energy cost of the inference of NNs on multi-core platforms. The training of the NN is assumed to be done beforehand.

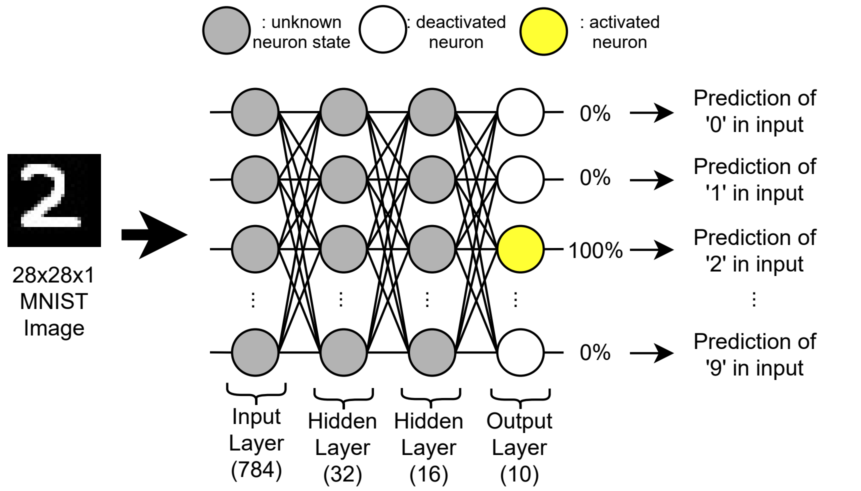


Figure III.1 – Example of a MLP. This NN is entirely constituted of dense layers, which are composed of a set of neurons fully connected to the previous layer. In this example, the MLP predicts that the input MNIST [11] image is a 2.

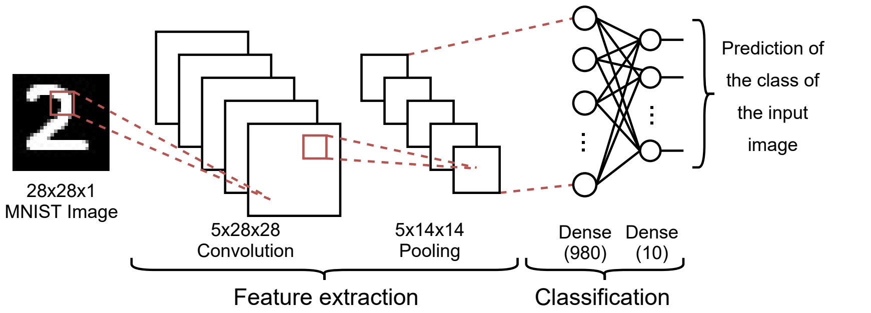


Figure III.2 – Example of a CNN. This NN is composed of convolution and pooling layers, used to perform feature extraction in order to ease the classification process performed by the dense layers placed afterwards.

Dense layers are composed of a set of base elements called neurons. Each neuron produces a single scalar as output, which indicates whether the neuron is activated based on the inputs processed. Every neuron inside a dense layer process all inputs of the layer. MLPs are also commonly called fully-connected networks because in such a configuration all neurons from a layer are connected to all neurons from the previous layer. We denote the output of the neuron ν_m, with $m \in (1, 2 \ldots M)$ and M being the number of neurons inside the considered layer. ν_m depends on the input $X = (x_1, x_2 \ldots x_N)$ with N being the total number of inputs, the weights $W_m = (w_{m,1}, w_{m,2} \ldots w_{m,N})$ and bias B_m of the neuron, and the activation function φ. The weights W_m and the bias B_m are obtained as a result of the training of the NN. The activation function of the neuron φ works as a threshold function to determinate if the neuron is activated by the inputs. The equation III.1 provides the relation that gives the output of a neuron ν_m, and the equation III.2 provides the output of an entire dense layer denoted $\mathcal{D}$. Because neurons from a same dense layer are independent from one another, it is possible to parallelize their execution.

$$\nu_m(X) = \varphi\left(\sum_{i=1}^{N} w_{m,i}x_i + B_m\right) \tag{III.1}$$

$$\mathcal{D}(X) = \begin{pmatrix} \nu_1(X) \\ \nu_2(X) \\ \vdots \\ \nu_M(X) \end{pmatrix} \tag{III.2}$$

In CNNs, convolution layers perform a set of *convolution* operations called *kernel convolution* on an input image. A *convolution* operation constitute a convolution layer's neuron. To help the reader make the difference between *convolution* and *kernel convolution*, we choose in this thesis to denote *convolutions* with the operator $*$ and *kernel convolutions* with the operator $\circledast$. Equation III.3 provides the relation to obtain the result of a *convolution* of two matrices of size (m, n). The result of a *convolution* is a scalar.

$$X * Y = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{pmatrix} * \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m,1} & y_{m,2} & \cdots & y_{m,n} \end{pmatrix} = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} x_{m-i,n-j} \cdot y_{1+i,1+j} \tag{III.3}$$

We define a *kernel convolution* as the operation of performing convolutions between a set of matrices obtained from the input image denoted I of size (m, n) by a matrix denoted K of size (k, l) with $k < m$ and $l < n$. K is referred as the convolution kernel or filter. The input image is split in a set of matrices $I_{i,j}$ of size (k, l) (same size as K) with $i \in \{1, 2 \cdots m\}$ and $j \in \{1, 2 \cdots n\}$. The relation that defines the $I_{i,j}$ is given in Equation III.4. In this relation, $I(\lambda, \mu)$ designates the element at index (λ, μ) of the input image I and $\lfloor x \rfloor$ designates the integer part of the real number x.

$$I_{i,j} = \begin{pmatrix} I\left(i - \lfloor \frac{k}{2} \rfloor, j - \lfloor \frac{l}{2} \rfloor\right) & I\left(i - \lfloor \frac{k}{2} \rfloor, j - \lfloor \frac{l}{2} \rfloor + 1\right) & \cdots & I\left(i - \lfloor \frac{k}{2} \rfloor, j + \lfloor \frac{l}{2} \rfloor\right) \\ I\left(i - \lfloor \frac{k}{2} \rfloor + 1, j - \lfloor \frac{k}{2} \rfloor\right) & I\left(i - \lfloor \frac{k}{2} \rfloor + 1, j - \lfloor \frac{l}{2} \rfloor + 1\right) & \cdots & I\left(i - \lfloor \frac{k}{2} \rfloor + 1, j + \lfloor \frac{l}{2} \rfloor\right) \\ \vdots & \vdots & \ddots & \vdots \\ I\left(i + \lfloor \frac{k}{2} \rfloor, j - \lfloor \frac{l}{2} \rfloor\right) & I\left(i + \lfloor \frac{k}{2} \rfloor, j + 1\right) & \cdots & I\left(i + \lfloor \frac{k}{2} \rfloor, j + \lfloor \frac{l}{2} \rfloor\right) \end{pmatrix} \tag{III.4}$$

The splitting is done as such: the first matrix produced starts with the element $I\left(\lfloor \frac{k}{2} \rfloor, \lfloor \frac{l}{2} \rfloor\right)$. Then at every step of the splitting, the resulting matrix is obtained by moving 1 pixel to the right of the input image. When it reaches the end of the line, the resulting matrix is obtained by moving to the beginning of the next line and moved by 1 pixel to the bottom (to the following line). During every step, the convolution of the matrix at the current location of the input image by the kernel is performed. The result of a kernel convolution is a matrix of size $\left(n - \lfloor \frac{k}{2} \rfloor, m - \lfloor \frac{l}{2} \rfloor\right)$ containing all the results of the individual convolutions $I_{i,j} * K$. This process is illustrated in Figure III.3. Convolution layers generally perform several kernel convolution on the input image, they have for this reason a number f of kernels. For example the convolution layer of the CNN presented in Figure III.2 has $f = 5$ different kernels. The result of a convolution layer is a set of f matrices corresponding to the results of each kernel convolution. The computations of kernel convolutions inside a convolution layer are independant and can thus be parallelized.

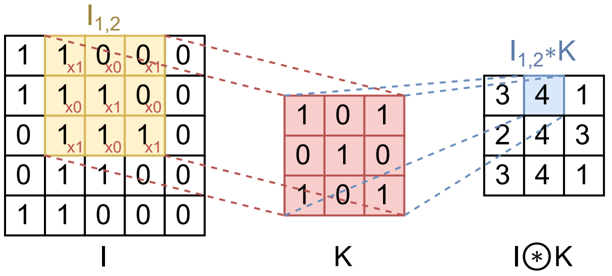
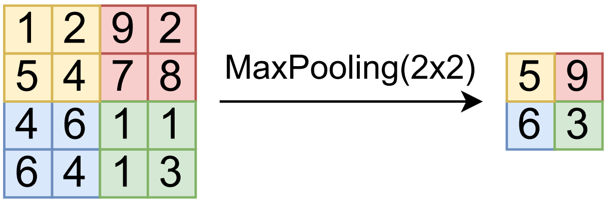


Figure III.3 – Illustration of the operations performed by a kernel convolution with a kernel denoted K on an input image denoted I

Pooling layers down sample the resulting image from a convolution in order to summarize the features extracted from the image by the convolution to ease the remaining processing. Two types of pooling layers are classically used: maximum and average pooling. In the scope of this study we only considered maximum pooling (also referred to as max pooling), but we argue that the modeling flow presented in the remainder of the manuscript can also be re-used with average pooling without loss of accuracy. Max pooling layers split the input image into a set of smaller images, from which it extracts the maximum value of the pixels. For example in a max pooling layer 2×2 as in the CNN from Figure III.2, the input image is splitted into smaller matrices of size 2×2. Equation III.5 provides the output of one max pooling 2×2 operation. Figure III.4 gives an illustration of the processing done by a max pooling 2×2 layer on an input image. Pooling down samples the input image.

$$p_{n,m}\left(\begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}\right) = max(x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2}) \tag{III.5}$$



1	2	9	2
5	4	7	8
4	6	1	1
6	4	1	3

5	9
6	3

Figure III.4 – Illustration of the processing performed by a max pooling 2×2 layer on an input image.

III.2 Description of NNs in Synchronous Data Flow (SDF)

III.2.1 SDF Model of Computation (MoC)

To ease the analysis and optimization process of NN deployments on embedded platforms, dataflow-oriented Models of Computation (MoC) are commonly used. For instance in ZigZag [84], the authors model the dataflow of NNs in order to optimize the data exchanges with the memory to implement efficient accelerator architectures. The approach led by the authors of Eyeriss [63] aims at proposing a new dataflow representation called row-stationary to optimize the exchanges between the processing elements that compose a systolic array for the inference of NNs on FPGA. In this work we rely on the Synchronous Data Flow (SDF) Model of Computation (MoC) introduced in [52]. SDF is commonly used to describe digital processing applications deployed on multi-core platforms (for example, see [67]).

SDF offers a clear and well-known semantic for the description of applications, which:

- allows to guarantee absence of interlocking in the application, taking into account the defined token production/consumption rates for communications,
- offers the separation of computation phases and communication phases, thus making possible the characterization and analysis of these two aspects independently from one another,
- supports real implementation on multi-processor hardware targets,
- offers the possibility to express an application's parallelism through the partitioning of the application in sets of actors,
- is adapted to pipeline/streaming execution of application.

Describing a system in SDF aims at splitting it into a set of actors. Actors model an element of computation. Their behavior is separated into three phases: read, compute and write. During the compute phase, no interference with any other actor can occur. The execution of an actor is called "firing". The dataflow between actors is modeled as communication channels, which are buffers containing data labeled as tokens exchanged between actors under the First In First Out (FIFO) principle. The use of bounded FIFOs prevents memory overflows. Actors exchange data labeled as token via communication channels, which act as buffers. For every communication channel in the SDF graph, each actor has a defined token production rate and a token consumption rate, which correspond to the amount of tokens the actor respectively generates upon firing or requires to fire. SDF graphs can also incorporate a source element, which models the arrival of input data from an external source, and a sink, which models the output of the system sent externally. An example of SDF graph is provided in Figure III.5.

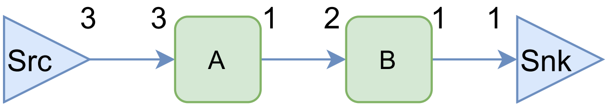


Figure III.5 – A simple SDF graph. Actors are depicted in green whereas communication channels are depicted in blue. The element "Src" is the source of the SDF graph and the element "Snk" is the sink. Black numbers sided next to actors and hovering communication channels are the token rates of actors. The graphical notation of the SDF graph in this figure are reused identically in the rest of the figures of the manuscript.

SDF offers a strict separation of computation and communication (read, write) phases of actors, under the condition that these phases are respected in the latter hardware and software implementation of the SDF graph. A SDF graph can be represented by a topology matrix, which contains the token rates of every actor for each communication channel. In this work, we use SDF to model NNs with separation of the computation and communication phases. We then complement SDF with timing and power models used to address the effect of shared resources on the execution of the application on multi-core platforms. In the next section, we explain how we use SDF to model NNs while offering the possibility to express different degrees of parallelism from the application point of view.

III.2.2 Modeling of NNs in SDF

In this work, we target multi-core platforms, which allow an acceleration of the execution of NNs by exploiting their intrinsic parallelism. SDF allows expressing different degrees of parallelism of the application with respect to the computations and communications performed. In [85], the authors present a way of capturing MLPs in SDF using several levels of granularity:

1. NN grain: where the whole NN is modeled as an actor,
2. Layer grain: where each layer of the NN is modeled as an actor,
3. Neuron grain: where each neuron is modeled as an actor,
4. Operation grain: where each operation (e.g. multiplication, sum, etc.) is modeled as an actor.

In this work, we further extend the work of [85] for intermediate granularity between levels 2. and 3. In opposition to [85], we also allow mixing the level of granularity inside a model: each layer of the NN has its own grain. We also extend this modeling to convolution layers in addition to dense layers. In the coarsest level of granularity a layer is modeled as one single actor. The finest level of granularity considered is the neuron / kernel convolution granularity, in which every neuron / kernel convolution in the layer is described as one actor. For example Figure III.6 provides an illustration of three different levels of granularity that can be used to model a given dense layer composed of 4 neurons. In the coarsest level of granularity denoted ❶, the 4 neurons are modeled as one actor. In this case the computation phase corresponds to the sequential execution of 4 neurons, which therefore does not allow for parallel execution. This will lengthen the execution time of the layer with repercussions on the power consumption. In the finest level of granularity denoted ❷, the 4 neurons are modeled each as one actor. This enables their execution in parallel, with possible execution time and energy enhancement, but it can be noted that this also

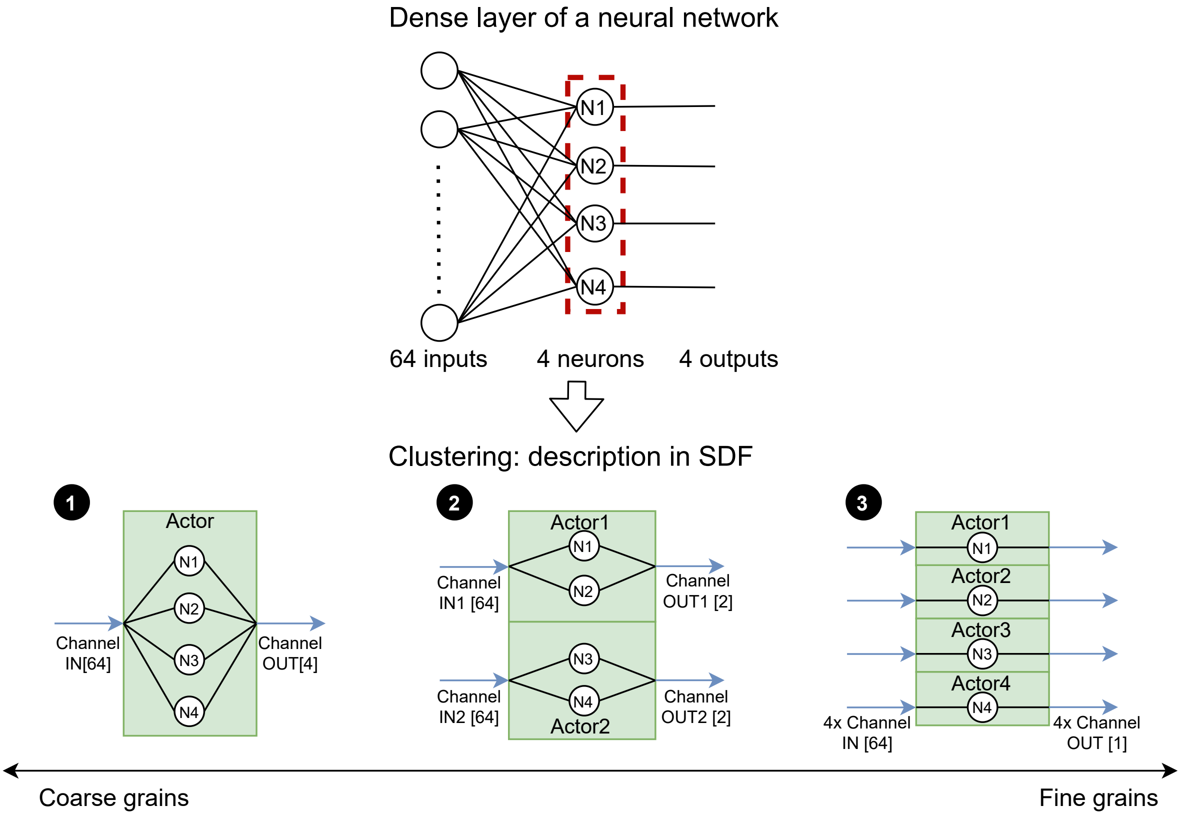


Figure III.6 – Three different *clusterings* of a dense layer composed of 4 neurons. ❶ corresponds to the coarsest granularity, as the whole layer is encapsulated into 1 actor. ❸ corresponds to the finest granularity, as every neuron is encapsulated into 1 individual actor. ❷ corresponds to an intermediate granularity between ❶ and ❸.

generate extra communication channels (4 channels of 64 inputs and 4 channels of 1 output against 1 channel of 64 inputs and 1 channel of 4 outputs for the coarsest grain). The additional channels will generate communication overheads with possible congestion on shared communication medium during concurrent accesses by cores, which will impact both latency and energy. Intermediate levels of granularity between the coarsest and finest grains such as ❸ provide a balanced ratio between parallelism execution of the layer and communication overhead. An evaluation of the level of granularity used to model NN layers is thus necessary to find solutions that optimize latency and power consumption.

To properly define intermediate levels of granularity, we introduce the notion of *clustering* of a layer in SDF, and we denote C the number of actors generated from a layer. For dense layers, the actors issued from the *clustering* contain sets of neurons. For convolution layers, the actors issued from the *clustering* contain sets of filter operations.

In Figure III.6, ❶ corresponds to a *clustering* $C = 1$ as only one actor is generated from the layer, ❷ corresponds to $C = 2$ and ❸ corresponds to $C = 4$. In this work, we only considered the *clustering* of layers into actors containing equitable computation workloads. For a dense layer, all actors issued from the *clustering* of a layer contain roughly the same number of neurons. For example, if the layer contains 4 neurons and $C = 3$, then two actors will contain 1 neuron and the last actor will contain 2 neurons. The same principle is applied to convolution layers with kernel convolutions. The proposed modeling flow could also be applied to uneven *clusterings* (without equitable share of workload among actors issued from a same layer) without modification.

Communication channels of NNs modeled in SDF correspond to the data exchanged between layers. Kernel convolutions and neurons require all the outputs from the previous layer to execute, therefore the token production rate is identical to the token consumption rate. In this thesis, we then refer to the number of tokens exchanged in a communication channel as the token rate, and we only depict the token rate once per communication channel on illustrations of NNs modeled as SDF graphs. The *clustering* of NN layers in SDF effects the number and token rate of communication channels in the SDF graph. The token rate of a communication channel is defined by the number of features (neurons or kernel convolutions) contained in the producing actor, which depends on the *clustering*. The number of communication channels between a layer denoted l and the next layer denoted $l+1$ is $C_l \cdot C_{l+1}$ where C_l is the *clustering* of layer l and C_{l+1} of layer $l+1$. The actors of a layer must all be connected to every actor from the previous layer. This highlights the importance of exploring and evaluating the *clustering* of layers under timing and power constraints, as the number of channels and tokens exchanged increases rapidly with the complexity of the graph, with consequences on the time spent in communication and power consumption.

In this work, we also re-use the introduction of a complementary actor named "decoder" as in [85] when the *clustering* of the last layer of a NN is higher than 1. This actor reads the outputs from the actors issued of the *clustering* of the last layer and assembles them in order to write all into a single output channel directed to the sink of the SDF graph. This actor makes it easier to analyze the graph by clearly identifying when it ends (the tokens from the decoder have been written to the single channel of the sink). It can also be used to normalize the output of the NN, for example by applying the Softmax function, to indicate more clearly which class has been recognized, but in our work the decoder does not apply any additional processing to the NN classification results.

Although this was not considered in this work, it would be possible to set up boundaries regarding the size of clusters using the memory model presented in Appendice C when performing the *clustering*. For example if the maximum memory of a tile is 1024 kB, and it must execute an actor issued from a dense layer with an input of 784 floats [1], then this tile can support a maximum of 320 neurons from this dense layer.

III.3 Model of Architecture (MoA)

III.3.1 Composition of the MoA

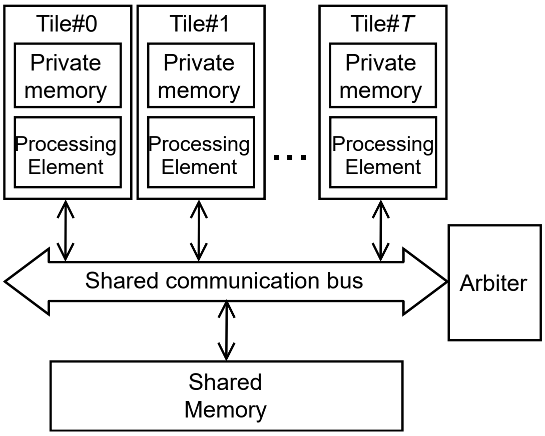


Figure III.7 – Example of platform which subscribes into our MoA. The MoA is composed of a set of tiles containing a single-core processor with private data and instruction memory. A shared memory is available for communications between tiles, which is accessed through a communication bus featuring an arbiter.

The considered MoA is a tile-based model, in which each tile is one single-core processor with its private data and instruction memories. Executing instructions from this private memory causes no interference with other tiles. Tiles are assumed to be identical except for the size of the memory. Data exchanges between different tiles are performed via a shared memory. The accesses to the shared memory are done using a shared communication bus. The shared communication bus integrates an arbiter to manage concurrent accesses to the shared memory by tiles. This MoA allows respecting the separation of computation phases (performed privately inside tiles) and communication phases (which involve accesses to the shared memory) from SDF. This MoA preserves thus the properties of SDF (comm/comp

[1] This corresponds to the MNIST dataset [11] input image size.

separation) and is therefore fully compositional, i.e. adding tiles does not disturb the properties of other tiles. The use of this MoA then allows to propose performance and power models that are also composable with respect to the number of tiles used in the implementation platform. A diagram that positions the different constituents of the MoA is given in Figure III.7.

Table III.1 presents notable real multi-core platforms used both in industry and academia for the processing of NNs and compare them with the platform used in this work. The PETA-MC platform [68, 67] (last entry of the table) is the implementation platform used in previous work of our project, which respects the presented MoA. For most listed platforms, cores are equipped with a private memory of limited size, and a shared memory of bigger size, accessible through a simple communication medium such as a shared interconnect. Real platforms however also feature sometimes additional larger on-chip shared memories and external DRAM. This is the case for example of the Tilera Tile-GX architecture, which feature a 32 MB internal SRAM and DRAM. The Tilera Tile-GX architecture offers the possibility to switch off these extra memories, which have their own power supply in case they are not needed for the execution of an application. When the extra memories (SRAM, DRAM) are not used, the listed platforms subscribes well into our MoA. From the considered platforms, the NxP MSC8156 DSP is the closest to our MoA as it features 6 tiles with private memory and a shared memory accessible through a shared interconnect. Other platforms such as the Kalray MPPA DPU, the Intel SCC and the Tilera Tile-GX features more complex communication medium between cores such as mesh networks. Apart from the communication medium, these platforms are built to be modular in regards to the number of tiles (e.g. the Tilera Tile-Gx architecture features platforms containing 16 up to 100 cores). Other recent platforms integrate hardware accelerators for fast inference of NNs. This is the case of the Coral dev board featuring a NXP i.MX 8M SoC (Quad-core Cortex-A53 plus Cortex-M4F), the NVIDIA Jetson Nano (4 ARM A57 cores) and the Sipeed MAIX Go Suit (2 RISC-V cores). The multi-core SoC used in these platforms uses cores with their own private memory, and internal shared memory accessible through a shared interconnect, as well as an access to DRAM. Compared to the other listed platforms, the Intel Movidius Myriad2 has a singular architecture, it features 2 LEON cores used to run a Real Time Operating System (RTOS) and manage peripherals in real time, and 12 Vector Processing Units (VPUs), which are efficient for processing large chunks of data issued from images and which are therefore often used as NN accelerators (see for example [26, 24, 27]). It also features more than 20 hardware accelerators meant for

Table III.1 – Main features of nine different multi-core platforms. When communicated by the chip provider, the core type, number of cores, core frequencies, memory sizes, communication medium and possible HW accelerator are provided in this table.

Product name	**Core type**	**Number of cores**	**Core frequencies**	**Memory sizes**	**Communication medium**	**Accelerator**
NxP MSC8156 DSP	NxP SC3850	6	1 GHz	64kB PM, 512kB SM, 1056kB add. SM, ext. DRAM	Shared interconnects	/
Kalray MPPA DPU	VLIW 64-bits core	5 clusters containing 16 cores (80)	600 MHz up to 1.2 GHz	32kB PM, 4 MB SM (per cluster), ext. DRAM	Not communicated	Each core features a co-processor for acceleration
Intel SCC	Intel P54C Pentium 32-bits	24 "tiles" containing 2 cores (48)	400 MHz up to 1.2 GHz	32kB core PM, 256kB tile SM, ext. DRAM	Mesh network	/
Tilera Tile-Gx	VLIW 64-bits core	16 up to 100 tiles	1.0 up to 1.5 GHz	64kB L1 PM, 256 kB L2 PM, 32 MB on-chip SRAM, ext. DRAM	Mesh network	/
Coral dev board (NXP i.MX 8M SoC)	Cortex A53 64-bits + Cortex M4F 32-bits	4 + 1	1.5 GHz	64kB PM, 1MB SM, ext. DRAM	Shared interconnect	Machine Learning Accelerator Google Edge TPU co-processor
NVIDIA Jetson Nano	ARM A57 64-bits	4	1.43 GHz	80kB PM, 2MB SM, ext. DRAM	Shared interconnect	128-core NVIDIA Maxwell GPU
Sipeed MAIX GO Suit	RISC-V	2	400-500 MHz	8 MB SM (on-chip SRAM), PM not communicated	Not communicated	KPU (Neural Network Processor)
Intel Movidius Myriad2	LEON 32-bits (RISC) + VPUs	2 + 12	600 MHz	LEON → PM cache (1×40 kB, 1×192 kB), VPUs → 2MB SM, 512MB DDR	Main communication bus → LEON Shared interconnect → VPUs	20+ programmable HW accelerators
PETA-MC platform	MicroBlaze 32-bits (RISC)	7	100 MHz	32-256kB PM 8kB SM	Shared interconnect	/

a fast execution of various DSP computation-intensive tasks. The Movidius Myriad2 is the only platform on the list that does not fit in our MoA, in addition to the use of external DRAM or the use of mesh networks as communication medium. Our MoA doesn't support the use of the external memory (such as DDR-RAM). In fact, using the external memory would break the hypothesis of both MoA and SDF that tiles cannot be interrupted when performing computations: tiles may require additional instruction or data triggering an access to the external memory, which may then lead to competitions between the tiles during computation phases. Possible solutions to extend our MoA and MoC to support the use of external memory are discussed in the perspectives, in conclusion of this manuscript. In the scope of this work, we do not consider more complex memory hierarchies involving external memory. We show that for the execution of simple NNs, our MoA is suited as it avoids important latency and energy overheads due to external memory usage. Our MoA also offers a structure for which the modeling effort is reasonable.

III.3.2 Power management within the MoA

In the scope of this study, we considered two communication modes. The first one is the one used in previous work [68, 67], and the second one introduced in this work, which allows an optimization of the power consumption within the platform. During the execution of NNs on a multi-core platform, when tiles require unavailable tokens to execute, they enter a waiting state until the tokens are available. When tiles are waiting, they are not contributing to the execution of the NN and can therefore be switched to a lower power consumption mode. Several techniques can be used in order to manage and reduce the power consumption of multi-core systems. Notable ones are:

- **Dynamic Voltage & Frequency Scaling (DVFS)** as used for example in [71]. It introduces different execution modes for cores with dedicated voltage and frequency settings.
- **Power gating** as used for example in [86, 25]. This technique aims at switching off the power supply from a device when it is not used. Switching off the power supply of a processing element deletes its execution context, which leads to important overheads when restarting the system afterwards.
- **Clock gating**, as presented in [33]. Clock gating reduces the power consumption of tiles at runtime by temporarily disabling their input clock signal. It does not reduce the static power consumption of circuits as they are still supplied with power. It however allows removing the dynamic part of power consumption as all activities

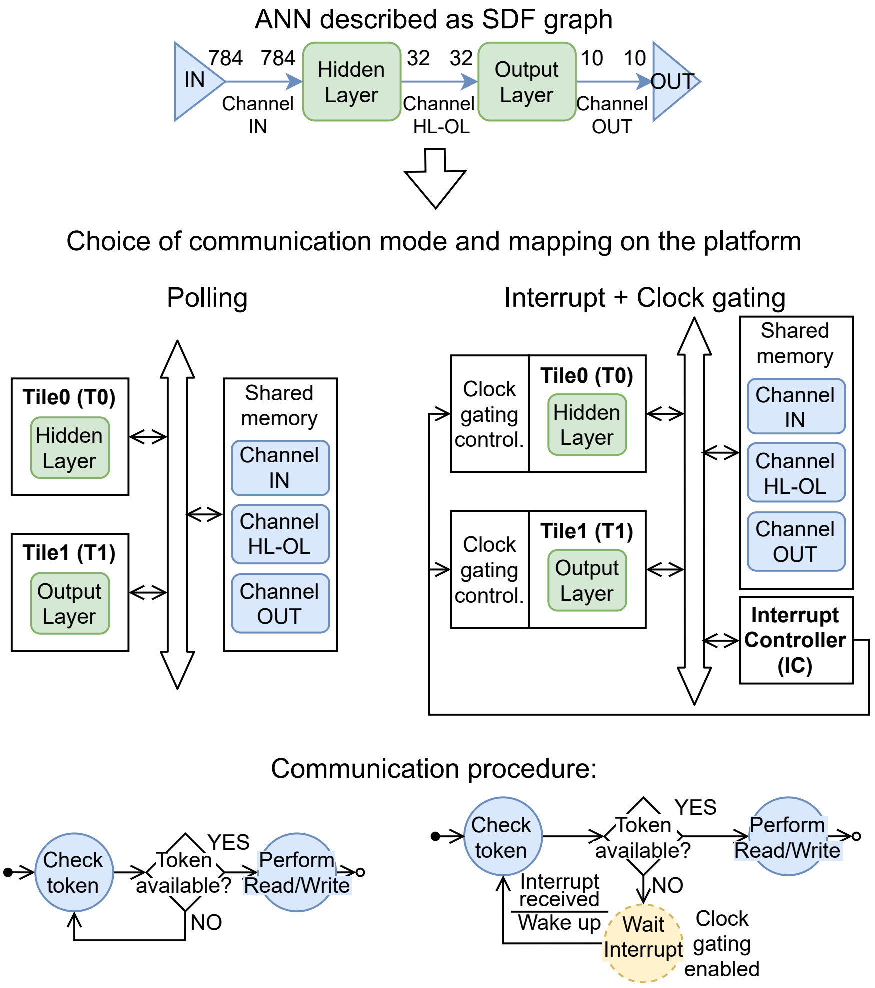


Figure III.8 – The considered versions of the platform. Version ❶ features polling-based communications without the use of clock gating. Version ❷ features interrupt-based communications with the use of clock gating. Automates describing the behavior of tiles when checking the availability of tokens respectively provided in ❸ and ❹ and examples of activity diagram in ❺ and ❻.

are suspended when the clock is disabled. It also has the advantage of offering minimal latency overheads due to the ease of enabling and disabling the clock signal and the low delay needed by the tile to resume its execution.

In this work, we chose to implement clock gating. In order to evaluate the impact of clock gating on NN execution on tile-based multi-core platforms, we considered two platforms with different communication modes, as depicted in Figure III.8 ❶ and ❷:

1. One mode with an active wait implemented as polling-based communications without the use of clock gating. When a tile needs unavailable data to execute, it

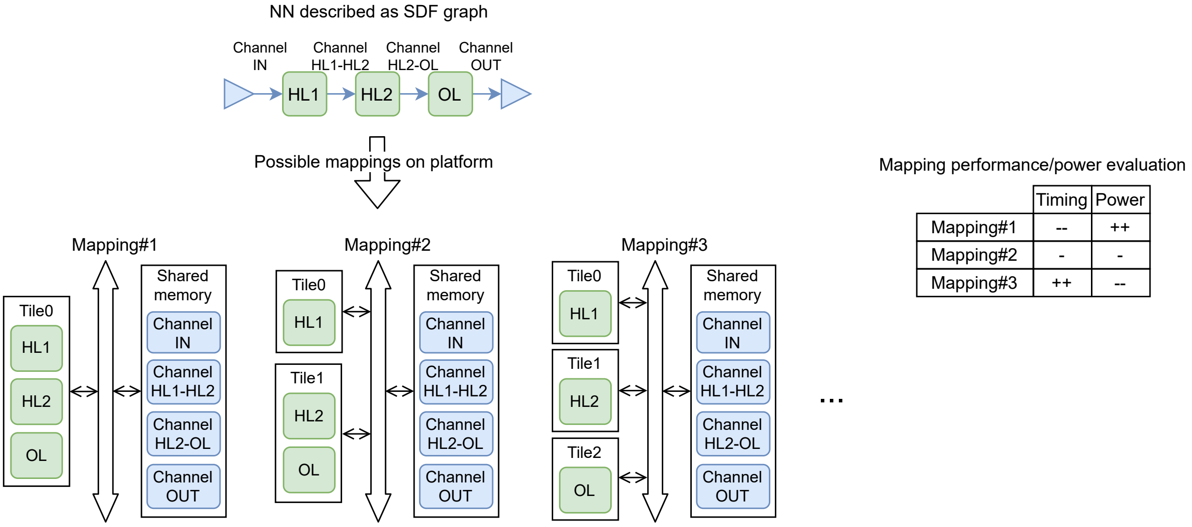
polls the shared memory until the data is available, as depicted in Figure III.8 ❸. This is the communication mode used in previous work [67] [68],

2. One mode with a passive wait implemented as interrupt-based communications with the use of clock gating. When a tile needs unavailable data to execute, it enters the low power mode (clock gating) until its clock is re-enabled by an interrupt signal, which indicates that the data is now available, as shown on the right in Figure III.8 ❹,

The interrupt-based platform introduces an interrupt controller to manage the interrupt signal, and clock gating controllers to manage the activation and deactivation of clock gating, as shown in Figure III.8 ❷. The interrupt signal is enabled by the interrupt controller peripheral when requested by a tile. When finishing a read/write transaction, tiles enables the interrupt signal through the use of the interrupt controller. When the interrupt is enabled, clock gated tiles exit the low power mode and resume their execution. The additional components (interrupt controller and clock gating controllers) cause overheads in latency and increase the power consumption of the system, as illustrated in the execution diagram in Figure III.8 ❻. However they enable the usage of low power mode instead of polling on the shared memory (see Figure III.8 ❺ and ❻), which can lead to power consumption enhancements. The trade-off in latency and power consumption between the two communication modes must therefore be evaluated on a case-by-case basis for each considered NN deployments. The next section explains how we map NNs modeled as SDF graphs on multi-core platforms that respects the hypothesis of our MoA.

III.4 Mapping of NNs modeled in SDF on platforms respecting the MoA

The actors from the clustering are mapped on the tiles available on the platform. The communication channels between actors are mapped on the shared memory. The tiles will read (*ReadTokens()* statement) and write (*WriteTokens()* statement) the data necessary for the execution of the actors in the communication channels. During the execution of the computation phase of actors, cores cannot be interrupted. The application is self-scheduled: the scheduling is established based on the dependency between actors. For a given SDF graph, several mappings of the application are possible. Examples of mapping for a given NN modeled in SDF is shown in Figure III.9. In this example, the SDF graph contains 3 actors *HL1*, *HL2* and *OL*. In the first mapping all actors are mapped onto one core, which does not exploit possible parallel execution. In the second mapping, the actor *HL1* is mapped on tile 0 while the other actors are mapped on tile 1. This enables a streaming execution of the graph by allowing executing *HL1* in parallel to the two other actors. In the third mapping, each actor is mapped on different tiles, which also enables a streaming execution of the graph, but this time *HL2* and *OL* are also executed in parallel. Each of these mappings have different latency and energy consumption, as presented in the table on the right on Figure III.9. In general using more cores enable a parallel execution of the application but it also generates more communications, with impact on the application's latency. Similarly, using more cores will increase the power consumption but the use of passive wait can significantly reduce it. The performance and power consumption of NN deployments on multi-core platforms is highly dependent on the mapping. It is thus necessary to proceed with a thorough evaluation of possible mappings under latency and energy constraints to find optimized ones. Comparing both levels of granularity and mappings of the application allows finding solutions that jointly optimize timing and power properties as well as the number of tiles used, as shown in Chapters IV, V and VI.



	Timing	Power
Mapping#1	--	++
Mapping#2	-	-
Mapping#3	++	--

Figure III.9 – Three different mappings of a NN described in the SDF MoC on the considered platform.

III.5 Real platform prototype implementation and considered applications

In order to obtain real time and power measurements to compare with the predictions of the modeling flow presented in the next chapters, we implemented a tile-based multi-core platform which respects the MoA, shown in Figure III.10. Our measurement setup is shown in Figure III.11. Two versions of the platform are implemented: one which relies on interrupt-based communications with clock gating and one which relies on polling-based communications without clock gating. The platforms are implemented on a Xilinx ZCU102 board, which features a UltraScale MPSoC+ FPGA [87]. They are implemented on the programmable logic section of the FPGA. The processing core of the tiles is a MicroBlaze. The private memory of tiles and the shared memory are implemented as Block Random Access Memory (BRAM), which is internal to the FPGA SoC. The communication medium to access the shared memory is implemented as a shared interconnect Advanced eXtensible Interface (AXI). In the clock gating version, the interrupt controller and the clock gating controller are IPs provided by Xilinx. The main version of the implemented platforms is composed of 7 tiles, including one with 1MB of private memory (Tile0) and others with 256kB of private memory [2]. Other versions have been implemented too as discussed in Chapter V to evaluate the applicability of our power model to platforms with different number of tiles and private memory sizes.

The targeted FPGA features several power supplies [87]. We probed *VCCINT*, which corresponds to the supply voltage of the programmable logic of the FPGA, and *VCCBRAM*, which corresponds to the supply voltage for the BRAM. The BRAM supply voltage is already well optimized by the provider as discussed in [89], and we observed no variation of power consumption on *VCCBRAM* when performing our tests. In fact, the activity linked to memory usage is observed rather on the *VCCINT* power supply, as the memory controllers are implemented on the programmable logic part of the FPGA. The correctness of this observation is confirmed through the use of the Xilinx Power Estimator (XPE) [90] available in Vivado, which predicts marginal dynamic contribution to the system consumption on the *VCCBRAM* voltage supply. For this reason, we focus only on *VCCINT* power consumption in this study.

The timing measurement infrastructure presented in [91] relies on code instrumentation

[2] One tile contains a bigger private memory than the others to enable the execution of single-core scenarios, which require important amounts of memory.

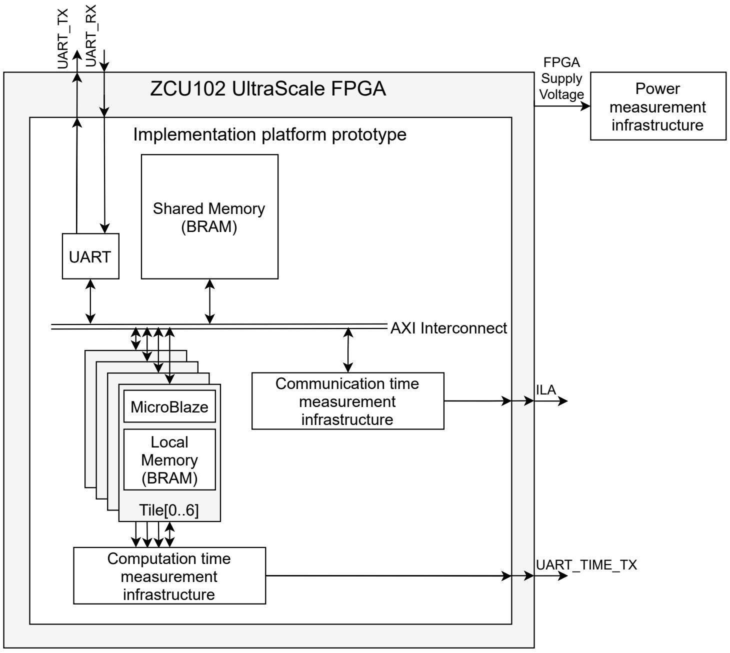

UART_TX
UART_RX
ZCU102 UltraScale FPGA
FPGA Supply Voltage
Power measurement infrastructure
Implementation platform prototype
Shared Memory (BRAM)
UART
AXI Interconnect
Communication time measurement infrastructure
ILA
MicroBlaze
Local Memory (BRAM)
Tile[0..6]
Computation time measurement infrastructure
UART_TIME_TX

Figure III.10 – Block diagram of the prototype implementation platform used in this work.

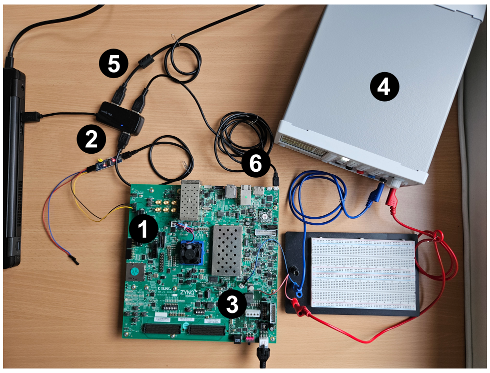


Figure III.11 – Experimental setup used in the scope of this thesis. The board is the ZCU102 UltraScale MPSoC+ [87]. ❶ marks the two pins of PMOD bank (part of the Inputs/Outputs of the FPGA) that we use as UART ports for our timing measurement infrastructure (UART_TIME_TX signal on Figure III.10). ❷ marks the UART-to-USB bridge device that we use to transmit the timing data to our PC ❸ marks the position of the probes for power measurements. Those probes are positioned on the two pins of the VCCINT power supply shunt resistor. ❹ marks the R&S HMC8012 Digital Multimeter [88], which measures the voltage across the VCCINT shunt resistor. ❺ marks the connection between the multimeter and the PC. The multimeter can be operated through Standard Commands for Programmable Instruments (SCPI). ❻ marks the connection of the board to the PC. This connection is used to program and debug the board. The UART_TX and UART_RX signals as shown in Figure III.10 are implemented on this connection.

to issue a start signal at the beginning of the execution of the SDF graph and issue a stop signal at the end, at the cost of the execution of a single instruction (two clock cycles). The overhead on timing when using this infrastructure is thus marginal. Based on the elapsed time between the start and stop signals, the execution time of the SDF graph is measured. In the scope of this work, we focus on streaming SDF applications, and we therefore measure the end-to-end latency of SDF graphs. To do so, we measure the elapsed time between two stop signals. In addition to this, the Integrated Logic Analyzer (ILA) [92] can also be used to measure the duration of transfers on the shared interconnect bus. The power measurements are obtained using the R&S HMC8012 Digital Multimeter [88] at a sampling rate of approximately 100 samples per second. As the execution times measured are in the order of milliseconds or even hundreds of microseconds, we took 10 000 for each considered measurement in order to obtain a representative sample of the system consumption. We then used the average of the measurements.

We considered three MLPs and one CNN to test our timing and power modeling flow. The CNN2, which features the LeNet5 topology [11], is used only in the DSE chapter VI. The four considered NNs vary in complexity with respect to both computations (number of layers, number of neurons and filters) and communications (number of channels, amount of tokens) workloads, which allows for a thorough validation of the modeling flow presented in the remainder of this manuscript. The considered NNs are presented as SDF graphs in Figure III.12, which also contains the number of layers, the number of features by layer (neurons, kernel convolutions) and the number of tokens exchanged between layers. The data-set used to train the NNs and the classification accuracy after training are provided in Table III.2. The *MLP1* has a small amount of computations while the *MLP2* features a more complex structure with two hidden layers and a bigger amount of computations. The different computation and communication workloads presented by these applications allow testing the validity of our flow in this regard. Testing the validity of our flow when applied to different data-sets is necessary as it allows also to test different sizes of inputs

Table III.2 – Number of layers, data-set and classification accuracy of the considered NNs

NN name	Number of layers	Data-set	Accuracy
MLP1	2	MNIST [11]	85%
MLP2	3	MNIST [11]	89%
MLP3	3	GTSRB [93]	20%
CNN1	4	MNIST [11]	77%
CNN2	7	MNIST [11]	N.A.

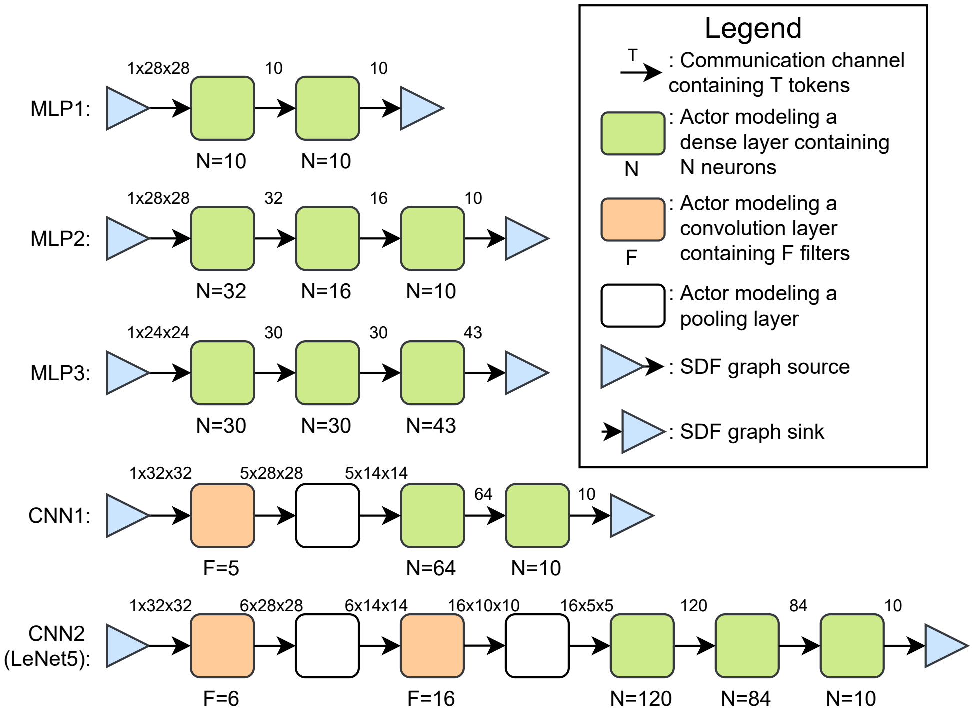


Figure III.12 – The considered NN applications described as SDF graphs with the coarsest level of granularity (layer grain, in which every layer's *clustering* is $C = 1$). We considered 3 MLPs and 1 CNN. The graphs on this figure do not feature a *decoder* actor due to the last layer's *clustering* being $C = 1$.

and outputs. For this reason we also consider a third MLP denoted *MLP3*, which was trained on the German Traffic Sign Recognition Benchmark (*GTSRB*) [93]. To highlight the possibility to use our modeling flow also with CNNs, we consider the *CNN1* application trained on the *MNIST* [11] data-set, developed for MicroBlaze by [94]. As shown in Figure III.12, *MLP1* and *MLP2* have an input channel of $28 \times 28 = 784$ tokens, as the images issued from the *MNIST* [11] handwritten digit recognition data-set are black and white images with a width and length of 28 pixels. The *CNN1* however has an input channel of $1024 = 32 \times 32$ pixels. Zero-padding[3] is in fact applied to the input images of the NN before the processing of the convolution layer to resize the images from the *MNIST* data-set. The *MLP3* features an input channel containing 512 tokens (24×24 grayscale image). It must be noted that the *GTSRB* is a complex data-set featuring images of different resolutions with colors. To enable the use of this data-set on our embedded

[3] In CNNs, zero-padding aims at surrounding the input image with zeroes. This helps preserve features that exist at the edges of the original image and control the size of the output of the convolution layer.

multi-core platform prototype, we apply a processing on the input image sizes to normalize them all to black and white format and 24 × 24 size. We also consider a second CNN, named *CNN2*, and which features the LeNet5 [11] architecture. This CNN is not used for the validation of our flow, but it is used for the demonstration of our DSE flow.

The considered NNs were trained and implemented using lightweight C libraries: the open source Fast Artificial Neural Network Library (FANN) [95] for MLPs, and a similar open source library for CNNs [4]. Lightweight C libraries have limitations regarding the NN's classification accuracy, as they do not offer as much optimizations of NN architecture and training as the mainly used Deep Learning frameworks with Python APIs, e.g. Tensorflow [43, 44], PyTorch [45], Caffe [46] and Keras [47]. However, using lightweight C libraries eases highly the deployment of the NN onto edge devices. While compilers have been proposed to help with the deployment of NNs trained with the mainly used Deep Learning frameworks at the edge, e.g. Apache TVM [48] and the onnx2c project [5], they do not always provide support for all hardware targets, and their use often results in a loss of precision when describing the NN in C language. Some approaches from the related work train NNs with Tensorflow [43, 44] and then implement trained NNs at the edge using custom C libraries [32, 59] [6]. Another major advantage of using C libraries is that they are very lightweight, thus making it easier to satisfy memory constraints of edge platforms. These libraries are valid choices for the implementation of NNs on edge devices. Despite being first online in 2003, LibFANN is still used in recent academic work for the implementation of ML-based applications such as cattle tracking [96], hand gesture recognition [97] and for lane change tracking in Advanced Driver Assistance Systems (ADAS) [98]. Similarly to LibFANN, the company Neuton [99] proposes simple MLP-based ML solutions coded in lightweight C code.

It can be noted in Table III.2 that *MLP3* has very low accuracy. This is due to the high reduction of the input images and simple structure of the network, used for the complex GTSRB data-set [93]. The *CNN1* has lower accuracy than the *MLP1* and the *MLP2* as usually a second convolution layer with another max pooling layer is used after

[4] Available on: `https://github.com/tranleanh/CNN-cpp`. Last accessed: 10.10.2023, commit: aac4107.

[5] onnx2c is a Open Neural Network eXchange (ONNX) to C compiler. It generates C code based on an input ONNX file. ONNX files can be generated by Tensorflow [43, 44], PyTorch [45], Caffe [46] and Keras [47] to save a trained NN model. See more on `https://github.com/kraiskil/onnx2c`. Last accessed: 10.10.2023, commit: dff37f0.

[6] The C library for CNNs used in [32] is available on `https://github.com/Tencent/ncnn`, last accessed: 10.10.2023, commit: d1289fb

the first one to increase the NN's accuracy, as it is done for example in the *LeNet5* CNN topology (*CNN2*) [11]. It can also be noted that the library used to train the network uses elementary training methods, which limits the accuracy. In the scope of this work, the classification accuracy of the network is secondary. Our aim is to propose models to evaluate the non functional properties (latency, power consumption) of NN mapped on multi-core platforms. The optimization of functional properties such as the classification accuracy is outside the scope of this work. All NNs are trained with *float32* precision and they use the ReLU activation function. We tested several clusterings and mappings of the considered NNs. All details regarding the tested clusterings and mappings can be found in appendices of this manuscript.

In the next chapter, we will present the proposed fast yet accurate timing prediction flow for NNs deployed on multi-core platforms under the work hypothesis formulated in this chapter.

Chapter IV

Simulation-based timing properties prediction approach

In this chapter, we present a modeling methodology to enable prediction and analysis of timing properties, such as latency and throughput, for NNs deployed on multi-core platforms that respect our MoA. The main contribution of the proposed methodology is to offer fast yet accurate timing predictions considering the clustering/mapping of the NN, the number of tiles used, the computation/communication workload and the possible use of power management. We propose a hybrid modeling flow, which combines measurements, analytical models and simulation. We explain how we build analytical computation time models for NNs based on the operations executed in layers. The analytical models are used along with a communication time model inside a simulation of the system, which allows taking shared resources into account and predict possible contentions. The computation and communication time models are calibrated through measurement. The validation of our modeling flow for several mappings of different NNs and the comparison with pure analytical models is provided and discussed.

IV.1 Timing modeling and prediction flow overview

The analysis of timing properties for NNs deployed onto multi-core platforms is necessary to optimize solutions and verify that timing constraints are met. Proposing models to achieve this goal is however tedious, as:

1. **Contention on shared resources** (e.g. shared bus and memory) arise when multiple tiles attempt to access them simultaneously. When they occur, contentions lead to important timing overheads that must be modeled. Because contention effects are highly dependent on tile-specific usage patterns and on the clustering/mapping of the application, they are particularly difficult to model.

2. **Computation and communication workload variation**: NNs are applications that highly benefit from several types of parallelization such as inter-layer parallelism (i.e. streaming execution) and intra-layer parallelism (clustering). Proposing models that allow exploring both intra- and inter-layer optimization is necessary to find optimized deployments, but difficult as the computation and communication workloads highly depend on the partitioning of the NN, with important effects on timing. The computation and communication workloads are also dependent on NNs themselves, as workloads differ highly based on the considered application.
3. **Communication procedures:** Multi-core platforms can support different communication procedures, such as polling-based without clock gating and interrupt-based communications with clock gating[1]. The communication procedure has important impact on timing properties of NNs deployments and must be modeled.

To tackle these problems, we propose a modeling methodology to offer prediction of timing properties in consideration of the clustering / mapping of NNs, communication workload and two communication procedures (polling and interrupt-based). Figure IV.1 shows an overview of our timing modeling flow. The novelties of the timing modeling methodology proposed in this chapter compared to the flow developed prior to this project [67] [68] are highlighted on Figure IV.1 by the symbol ★. The contributions to this flow are the following:

1. The proposition of an analytical computation time model to offer a fast prediction of

[1] These two communication procedures are described in details in Chapter III.

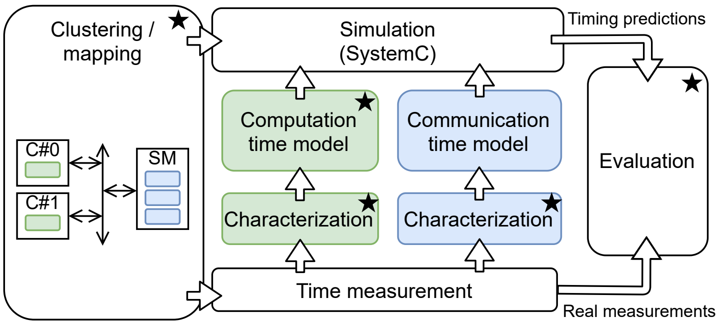


Figure IV.1 – Overview of the timing modeling flow. A mapping of clusterized NN onto the platform is evaluated using an executable model described in SystemC, which uses separate computation and communication time models for delay prediction. These two models are characterized through measurements. The prediction of the models are validated against real measurements. The new contributions to the flow are marked with the ★ symbol.

the delay required to process NN operations on processing elements. We also present our measurement-based characterization approach to appropriately calibrate delay models and enable accurate predictions.

2. The extension of the communication time model proposed in [68] to interrupt-based communications with the use of clock gating.
3. The integration of the new computation time model and the updated communication time model inside the existing simulable SystemC code, which allows modeling shared resources.
4. The validation of the modeling flow on mappings of different NNs to highlight the fast yet accurate predictions regardless of the clustering, mapping and communication workload of NNs onto multi-core platforms.
5. The comparison of our modeling flow's accuracy and evaluation speed compared to a pure analytical model with respect to models proposed in related work, to further comment on the benefits and limitations of our flow.

The main points of our workflow presented in Figure IV.1 are 1. the hybrid modeling approach, 2. the simulation-based prediction and 3. the evaluation of the approach.

Hybrid modeling approach: The input of our modeling flow is a NN described as a SDF graph mapped onto a multi-core platform that respects our MoA. The separation of communication and computation of NNs described in SDF and respected by our MoA allows building separate communication and computation models. We propose analytical computation time models (depicted in green in the middle of Figure IV.1) for NN's layers. The analytical computation time models are presented in Section IV.2. The communication time model (depicted in blue in the middle of Figure IV.1) is issued from [68] and was initially proposed and calibrated for polling-based communications. We explain how this model is constituted in Section IV.3, and how we extended this model to interrupt-based communications. Both computation and communication time models are calibrated through measurements using the timing measurement infrastructure from [67] in order to offer accurate latency predictions.

Simulation-based prediction: The computation and communication time models are used in a simulation model, which captures the structural and behavioral features of the studied architecture. This simulation model is described with the SystemC framework.

The simulation workflow allows considering the shared resources and predicting possible contentions when several cores try accessing the same resource simultaneously. The simulation calls the computation time model for delays regarding computation, and the communication time model for delays regarding communications. The simulation is presented in Section IV.4.

Evaluation of the approach: The final step (on the right of Figure IV.1) aims at validating the predictions of our modeling flow against latency and throughput measurements obtained from the real implementation of the NN on our platform prototype on FPGA. We also propose a pure analytical timing model for NNs mapped onto multi-core platforms, which is presented in Section IV.5.2. We use this model to compare its prediction time and accuracy with the proposed simulation-based modeling flow, to further discuss the advantages and limitations of the flow. The validation results and the comparison of the pure analytical model against the proposed simulation flow are presented in Section IV.5, and discussed in Section IV.6.

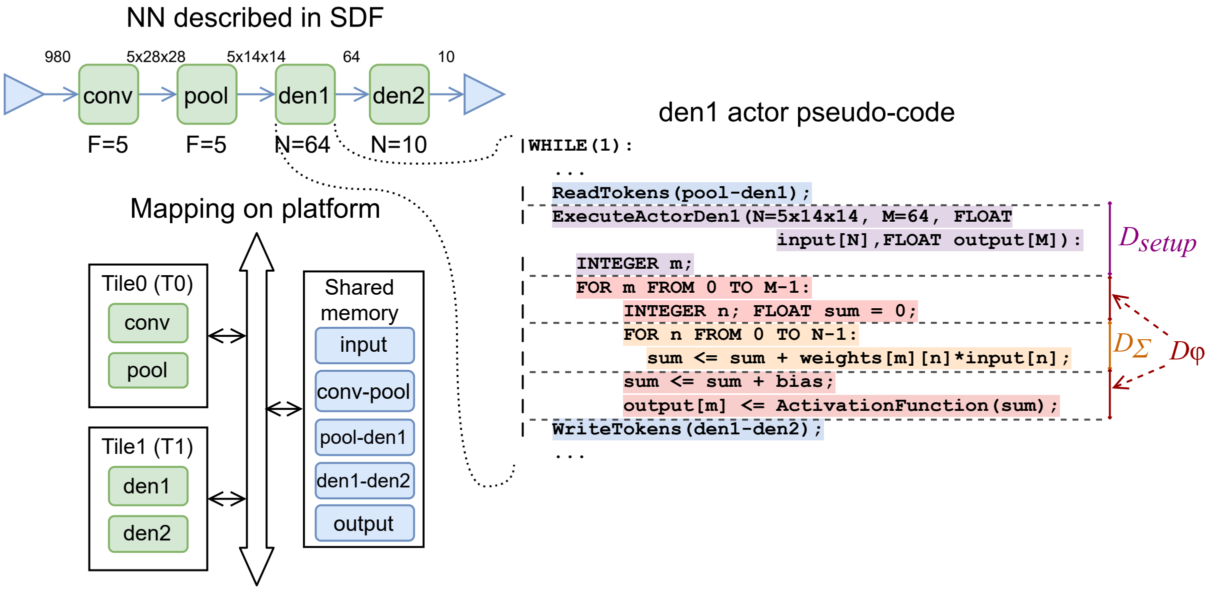
IV.2 Computation time modeling approach

In this section we first present our analytical computation time models, and how they are established based on the operations executed inside NN layers. We then describe the characterization process followed in order to derive and calibrate our models from measurements. Our analytical computation time models are used to predict the time spent in computation by cores when executing NN layer operations. They are depicted in green in Figure IV.1.

IV.2.1 Analytical computation time models

Dense layers: First, we focus on the analytical computation time model proposed for dense (fully-connected) layers[2]. This model can be used to predict the computation delay of any actor containing a set of neurons issued from the clustering of a dense layer. The analytical model is based on the operations executed by each core to compute a dense layer, as illustrated in Figure IV.2. On the right of this figure, a zoom is provided on the pseudo-code of the dense layer actor *den1*. Based on the code, three base delays can be

[2] The operations executed inside dense layers (and also convolutional and pooling layers) are presented in Chapter III.



```
WHILE(1):
  ...
  ReadTokens(pool-den1);
  ExecuteActorDen1(N=5x14x14, M=64, FLOAT
                   input[N],FLOAT output[M]):
    INTEGER m;
    FOR m FROM 0 TO M-1:
        INTEGER n; FLOAT sum = 0;
        FOR n FROM 0 TO N-1:
          sum <= sum + weights[m][n]*input[n];
        sum <= sum + bias;
        output[m] <= ActivationFunction(sum);
  WriteTokens(den1-den2);
  ...
```

Figure IV.2 – Extraction of the analytical computation time model for dense layers from NNs described as SDF graphs. The pseudo-code of the dense layer *den1* is provided. The elementary delays D_Σ, D_φ and D_{setup} can be identified from the code.

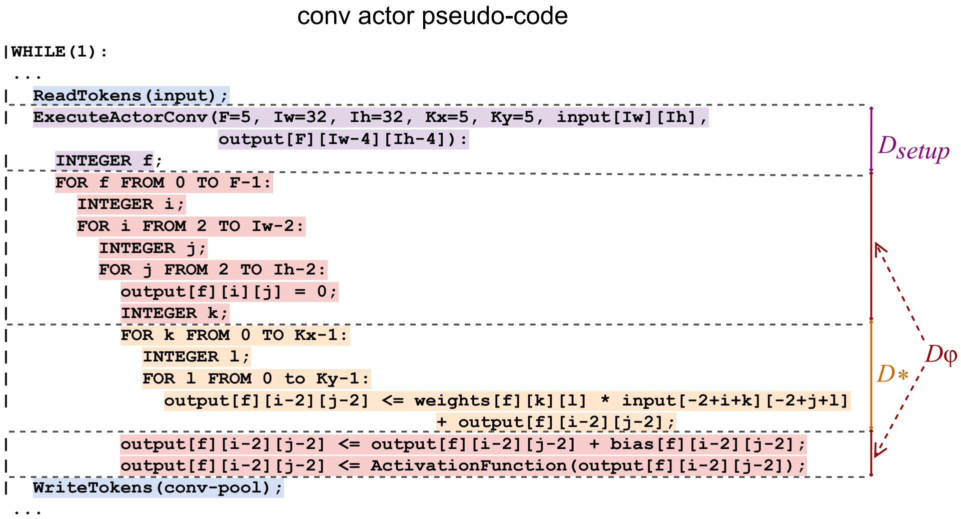
identified: D_Σ, associated to the code highlighted in orange, D_φ, associated to the code in red, and D_{setup}, associated to the code in purple. D_Σ is the delay needed to perform the Multiply-ACcumulate operation (MAC): the multiplication of the input of the neuron by the weight and its addition to the output of the neuron. D_φ corresponds to the delay needed to compute the activation function of the neuron but also to add the bias (see Equation III.1). In addition to these two delays, the initialization of the variables at the beginning of the function *ExecuteDen1* and the call and return procedures of the function causes a delay denoted D_{setup}.

According to the definition of a neuron of a dense layer, its computation depends linearly on the number of inputs it has. In the pseudo-code on Figure IV.2, we can see that the MAC operation (delay D_Σ) is executed N times inside a *for* loop, where N denotes the number of inputs of the dense layer. The addition of the bias and computation of the activation function (D_φ) are only done once per neuron. Because neurons from a same dense layer are independent, the execution time of a cluster of neurons depends linearly on the number of neurons it contains. As seen in the pseudo-code, the computations of the activation function and addition of the bias (delay D_φ) are repeated M times in a *for* loop, where M is the number of neurons inside the actor. Since the MAC operations (D_Σ)

are also encapsulated in this second *for* loop, they are repeated a total of $M \cdot N$ times. Equation (IV.1) presents the formula of the total delay needed to compute a cluster of neurons issued from a dense layer. We suppose that during the computation phase of the cluster of neurons, all the data (neuron's weights, inputs) and instructions are available in the local memory of tiles (assumption in line with our MoA). The model is thus scalable in consideration of the number of neurons contained in actors issued from the clustering of dense layers.

$$D_{dense}(M,N) = M \cdot N \cdot D_{\Sigma} + M \cdot D_{\varphi} + D_{setup} \tag{IV.1}$$

This equation is valid for processing elements computing NNs under the hypothesis of our MoC and MoA and would not be applicable to architectures incompatible with our MoA. It can be observed on Figure IV.2 that the delays D_{Σ} and D_{φ} also extra-delays linked to the implementation in code of the NN. For instance, D_{Σ} and D_{φ} also include delays for the initialization and increment of variables used inside *for* loops. D_{φ} also contains the delay to initialize the variable containing the neuron's result denoted *sum*.



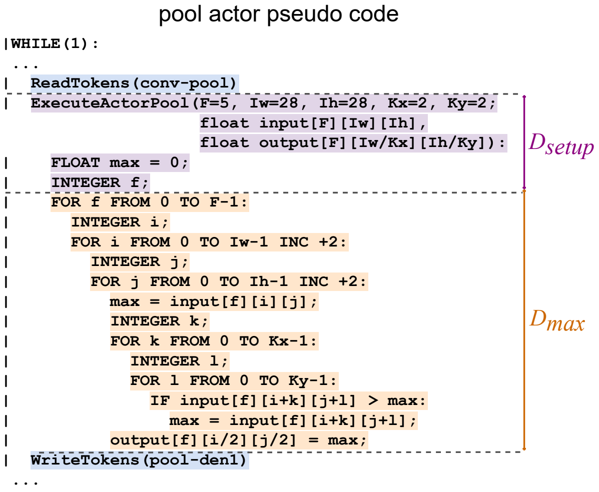
```
|WHILE(1):
 ...
|  ReadTokens(input);
|  ExecuteActorConv(F=5, Iw=32, Ih=32, Kx=5, Ky=5, input[Iw][Ih],
                    output[F][Iw-4][Ih-4]):
|    INTEGER f;
|    FOR f FROM 0 TO F-1:
|      INTEGER i;
|      FOR i FROM 2 TO Iw-2:
|        INTEGER j;
|        FOR j FROM 2 TO Ih-2:
|          output[f][i][j] = 0;
|          INTEGER k;
|          FOR k FROM 0 TO Kx-1:
|            INTEGER l;
|            FOR l FROM 0 to Ky-1:
|              output[f][i-2][j-2] <= weights[f][k][l] * input[-2+i+k][-2+j+l]
                                     + output[f][i-2][j-2];
|          output[f][i-2][j-2] <= output[f][i-2][j-2] + bias[f][i-2][j-2];
|          output[f][i-2][j-2] <= ActivationFunction(output[f][i-2][j-2]);
|  WriteTokens(conv-pool);
 ...
```

Figure IV.3 – Extraction of the analytical computation time model for convolution layers from NNs described as SDF. The pseudo-code of the convolution layer *conv* issued from the SDF graph presented in Figure IV.2 is provided. The elementary delays D_*, D_{φ} and D_{setup} can be identified from the code.

Convolutional layers: Using the same approach, we propose an analytical computation time model for actors issued from the clustering of convolution layers (Equation IV.2).

$$D_{conv}(F, I_w, I_h, K_x, K_y) = F \cdot I_w \cdot I_h \cdot K_x \cdot K_y \cdot D_* + F \cdot I_w \cdot I_h \cdot D_\varphi + D_{setup} \tag{IV.2}$$

D_{conv} depends on the number of convolution filters F, the width of the input image I_w and its height I_h, and the filters' width K_x and height K_y. Based on the code executed to obtain the results of a convolution layer presented in Figure IV.3, we also identify the elementary delays D_*, which is the delay needed to perform the convolution operation, D_φ, which is the delay needed to add the bias and compute the activation function, and D_{setup}, which is the time needed to call the *ExecuteActorConv* function. It can be noted from Figure IV.3 that all delays contain also extra delays linked to the initialization of variables and the increment of variables used in for loops. For example D_φ also contains the delays to increment the variables (*for* loops) in order to perform all convolution operations when performing kernel computations.



```
|WHILE(1):
 ...
|  ReadTokens(conv-pool)
|  ExecuteActorPool(F=5, Iw=28, Ih=28, Kx=2, Ky=2;
                    float input[F][Iw][Ih],
                    float output[F][Iw/Kx][Ih/Ky]):
|    FLOAT max = 0;
|    INTEGER f;
|    FOR f FROM 0 TO F-1:
|      INTEGER i;
|      FOR i FROM 0 TO Iw-1 INC +2:
|        INTEGER j;
|        FOR j FROM 0 TO Ih-1 INC +2:
|          max = input[f][i][j];
|          INTEGER k;
|          FOR k FROM 0 TO Kx-1:
|            INTEGER l;
|            FOR l FROM 0 TO Ky-1:
|              IF input[f][i+k][j+l] > max:
|                max = input[f][i+k][j+l];
|          output[f][i/2][j/2] = max;
|  WriteTokens(pool-den1)
 ...
```

Figure IV.4 – Extraction of the analytical computation time model for max pooling layers from NNs described as SDF graphs. The pseudo-code of the max pooling layer *pool* issued from the SDF graph presented in Figure IV.2 is provided. The elementary delays D_{max} and D_{setup} can be identified from the code.

Pooling layers: The analytical computation time model for actors of max pooling layers is provided in Equation IV.3.

$$D_{pool}(F, I_w, I_h, K_x, K_y) = F \cdot I_w \cdot I_h \cdot K_x \cdot K_y \cdot D_{max} + D_{setup} \tag{IV.3}$$

The delay needed to compute actors issued from the clustering of max pooling layers depends on the number of filters F, the input image's width I_w and height I_h, and the maximum filter's width K_x and height K_y. Figure IV.4 shows that two elementary delays can be identified from the computation of these operations: D_{max}, which corresponds to the delay needed to compute the maximum of $K_x \cdot K_y$ pixels from the input image and parse the input image, and D_{setup}, which corresponds to the delay needed to call the function *ExecuteActorPool*. Once again it can be noted from the provided snippet that extra delays linked to the initialization of variables and increment of variables used in *for* loop are also included in D_{max} and D_{setup}.

IV.2.2 Measurement-based characterization approach for computation time models

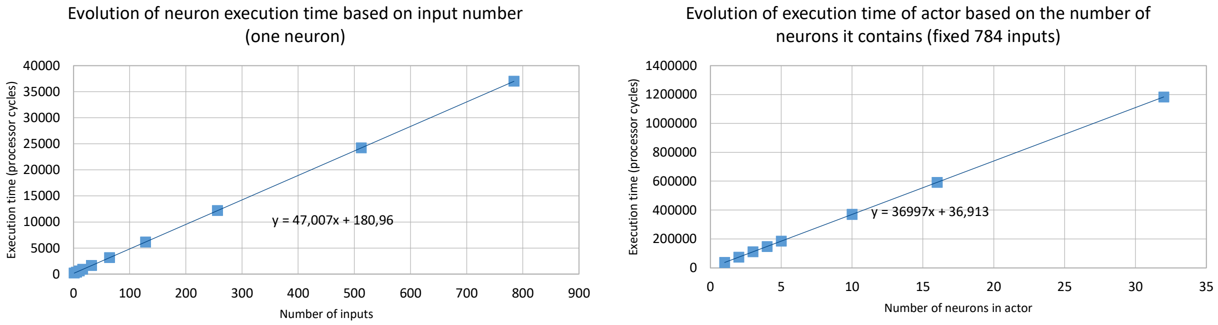


Figure IV.5 – Examples of plots obtained from the calibration of the analytical computation time model for dense layers. The measurements are obtained on a tile consisting of a MicroBlaze core and its private memory. The MicroBlaze code and data are stored entirely in the private memory (in compliance with the MoA). The plot on the left shows the evolution of the execution time of a neuron based on the number of inputs it has. The plot on the right shows the evolution of the execution time of an actor based on the number of neurons it contains (with a fixed number of inputs). Plots in this figure represent a subset of the tested parameters and measured data, on which multi-linear regression was applied.

In our approach elementary delays identified in the proposed models (e.g. D_Σ) are calibrated through measurement. Another possibility to alleviate the calibration effort is to provide an estimation of the elementary delays for a given processor, using information from the chip provider for example. Nevertheless the advantage of characterization through measurements over estimations is to offer higher accuracy, as measurements allow to properly model the effect of the code implementation of the NN (i.e. for instance the initialization of variables used for the computation) as well as the effect of the compiler on timing. Our experiments have shown indeed that base delays vary highly depending on the hardware used (e.g.: use of FPUs or multipliers) and the compiler settings (e.g. *-O0*, *-O1*, etc.).

The calibration is performed by measuring the execution time of elementary NN layer operations on tiles consisting of a MicroBlaze core from the prototype implementation platform while manually varying the parameters of the cluster in the code (e.g. for dense layers N the number of neurons and I the number of inputs). In the scope of our experiments all tiles feature one MicroBlaze core equipped implementing a FPU and a multiplier, and we use the *mb-gcc* compiler with *-O0* option. A multi-linear regression is performed on the measured data to extract the calibrated elementary delays. An example of the evolution of the execution time of actors issued from dense layers measured and regressed in order to obtain the calibrated model based on the tested parameters is provided in Figure IV.5. Once calibrated, the models can be used to predict the computation time of any actor issued from the clustering of NN layers without further re-calibration for a given core and compiler setting. Because tiles are assumed to be identical except for their private memory size, they can be used to predict the computation time of clusters on any tile.

To calibrate the analytical computation time models, we used the measurement infrastructure implemented inside our platform prototype as presented in Chapter III. The elementary delays for the analytical computation time model have been calibrated as follows (all delays provided in processor cycle number):

- Dense layers (MLP): $D_{\Sigma} = 47$, $D_{\varphi} = 146$ and $D_{setup} = 39$,
- Dense layers (CNN): $D_{\Sigma} = 50$, $D_{\varphi,ReLU} = 106$, $D_{\varphi,None} = 53$ and $D_{setup} = 31$,
- Max pooling layers: $D_{max} = 25$ and $D_{setup} = 106$,
- Convolution layers: $D_{*} = 77$, $D_{\varphi} = 631$ and $D_{setup} = 28$.

We observed little to no variability when calibrating these base delays, with the exception of the activation function, whose delay depends on the value (several if statements). We took the average value for all measured delays. The elementary delays for the dense layers are calibrated with different values for MLPs and CNNs. This is due to the different implementation of the dense layers between the library LibFANN we used for the training and inference of MLPs, and the CNN_C library we use for the training and inference of

CNNs. The analytical formula is unchanged, only the elementary delays are calibrated with different values. This highlights the importance of the characterization process through measurements, which allows properly modeling the effect of the implementation of the NN and the compiler settings, which can differ between deep learning frameworks. If we would have use estimations instead of measurements, while making the coarse grained assumption that each statement[3] requires 3 instructions, which take 1 cycle to be processed, the delays for dense layers of MLPs would have been $D_{\Sigma} = 30$, $D_{\varphi} = 61$ and $D_{setup} = 6$, which represents a major underestimation against the values obtained through measurements. This shows that the base delay values are not trivial to predict and that measurement is required to properly set them.

IV.3 Communication time modeling approach

The communication time model is depicted in blue in the middle of Figure IV.1. The communication time model was initially proposed in [68] to offer fast yet accurate timing prediction of dataflow applications on multi-core platforms that respects the MoA. It uses an analytical timing model to predict the delay of tiles when writing/reading tokens in shared memory presented in Section IV.3.1. This timing model is combined with a high level abstraction executable model of channels introduced in Section IV.3.2. This leads to an activity-sensitive model, which makes possible to predict the usage of communication resources even in the case of contention due to concurrent accesses. The characterization process of the model through measurements is explained in Section IV.3.3.

IV.3.1 Analytical timing model for token production/reading in shared memory

In the considered MoA, the communications of data between tiles are implemented using the shared memory, which is accessed through the shared bus. An arbiter is implemented on this bus to ensure that only one tile is accessing the shared memory at a time. In this section, we present the analytical timing model for token production/consumption (write and read access) in shared memory. A write or read access to the shared memory is composed of three main phases: check token availability, buffer access (read/write) and token status update. Equation IV.4 presents the analytical model issued from the

[3] i.e. variable initialization, assignment, operations, memory accesses, etc.

communication time model presented in [68], which provides the delay denoted D_{RW} needed to read/write a total of n_T tokens inside a channel on shared memory. It is important to note that this delay does not model contentions for shared resources.

$$D_{RW}(n_T) = \underbrace{t_{init,RW} + t_p}_{\text{Check token availability}} \underbrace{+t_{pre,RW} + t_{RW} \cdot n_T + t_{RWl} \cdot (n_T - 1)}_{\text{Buffer access}} \underbrace{+t_{post,RW} + t_w}_{\text{Token status update}} \qquad \text{(IV.4)}$$

In this equation, n_T is the total number of tokens to read/write. The buffer access, which aims at reading/writing the n_T tokens, is preceded by an operation to check the buffer's availability by reading a single value, and followed by the token status update, which corresponds to the writing of a single value. $t_{init,RW}$ is the delay to initiate the read/write transaction. t_p is the delay to check the availability of a token. $t_{pre,RW}$ is the time needed to prepare the read/write transaction. t_{RW} is the delay of performing one token read/write transaction. t_{RWl} is a delay needed to wait to proceed with the next read/write operation. $t_{post,RW}$ is the delay needed to terminate the memory access. t_w is the delay needed to write the new token availability status after the read/write transaction.

This analytical model allows modeling communications on the platform with a coarse granularity, as it only models the time to perform a read and write transaction. In reality when executing NNs on multi-core platforms, contentions occur due to concurrent accesses of tiles to shared memory, which generate important overheads in timing. The analytical model in Equation IV.4 must therefore be extended to also model the shared communication medium, the shared memory and contentions.

IV.3.2 Message level communication time model

In [68], the authors proposed to integrate the analytical model from Section IV.3.1 in an executable model, which takes into account all communication phases of tiles (read, write, wait), as well as the behavior of communication channels, the shared bus and the shared memory. The proposed model is a message level communication model, which was compared to a cycle-accurate communication time model in [100] and showed significant prediction speedup without loss of accuracy. The principle behind this model is depicted in Figure IV.6. An execution diagram is provided, representing how many simulation synchronization events are necessary in order to predict the time spent by a tile to perform a write operation in shared memory with a waiting period. With the cycle accurate model,

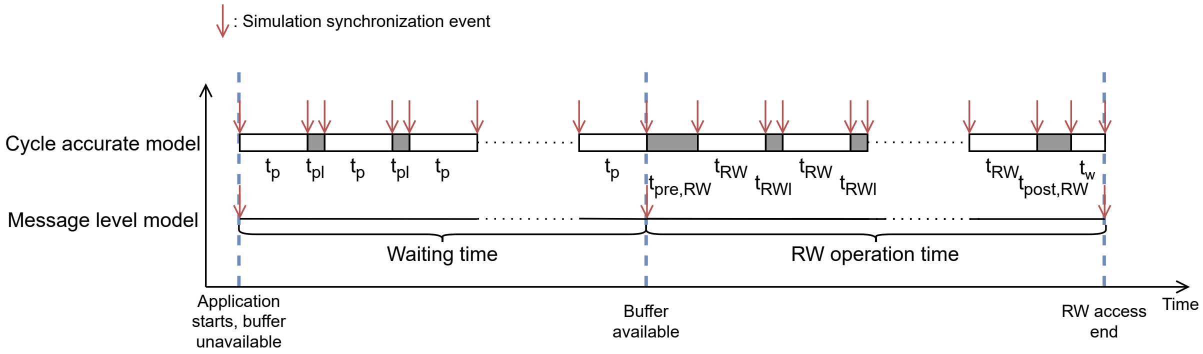


Figure IV.6 – Illustration of the message level communication time model from [68] compared to a transaction level model. The diagram shows the calls to the communication time model that need to be performed in order to predict the time spent by a tile to undertake a write operation of k tokens on the shared memory with a waiting period.

simulation synchronization events are needed after each delay from Equation IV.4. With the message level model, simulation events are only required upon the change of buffer availability. The number of simulation events with the message level model is highly reduced, which allows improving the simulation speed.

This communication time model was tested in [68] with the same communication bus as we use in our experiment platform. It was tested for polling-based communications on platforms that respects our MoA. In this work, this model is used to predict the communication durations between clusters issued from the clustering of NN layers while considering the influence of platform shared resources. To support interrupt-based communications with the use of clock gating, we performed a new characterization process, which highlighted that the model's structure can be re-used identically for interrupt-based communications with different values of elementary delays, obtained through measurements.

IV.3.3 Measurement-based characterization approach for the communication time model

The characterization of the model for both communication procedures is done using the Integrated Logic Analyzers (ILA) implemented inside the Xilinx Vivado design suite. The ILA probes the signals linked to the usage of the shared memory. We generated artificial communication workloads on the shared memory and used the delays obtained from the data probed by the ILAs to calibrate the elementary communication delays used in the model. Table IV.1 provides the values of the elementary delays in processor cycle count obtained through measurements based on the communication procedure. Elementary delay notations are maintained between the two communication procedures. It can be noted that most delays are calibrated with the same values between the two communication models. The differences lie in the delays related to polling, which are all set to 0 in the interrupt-based platform since polling is replaced by waiting for the interrupt signal to be enabled. The delay for the initialization phase of the read/write transactions is much longer on the interrupt-based platform, since the core must enable its sensitivity to interrupt sequences and set the interrupt controller, which is not the case with the polling-based communications.

Table IV.1 – Calibrated elementary delays for the communication time model for polling and interrupt-based communications. All delays are in processor cycle number. The delays that differ between the two communication procedures are highlighted in bold.

Communication procedure	t_r	t_p	t_w	t_{rl}	t_{wl}	t_{pl}	$t_{r_{loop}}$	$t_{w_{loop}}$	$t_{p_{loop}}$	t_{pr_r}	t_{po_r}	t_{pr_w}	t_{po_w}	t_{init_r}	t_{init_w}
Polling	8	**8**	5	14	13	**7**	22	18	**15**	15	11	15	9	**15**	**16**
Interrupt	8	**0**	5	14	13	**0**	22	18	**0**	15	11	15	9	**348**	**349**

IV.4 Simulation model description in SystemC

Our performance models were created with the SystemC framework with the same organization as in [67]. In order to predict the performance of NN mapped onto multi-core platforms, both the analytical computation time model presented in Section IV.2 and the message level communication time model presented in Section IV.3 are used inside simulations of the platform executing the NN described in SystemC. The resulting executable model allows to model all the tiles composing the system as well as the shared resources, and predicting the impact of possible contentions on the latency of the application. Figure IV.7 provides an illustration of the use of the SystemC models. The computation and communication time models are integrated in a behavioral description of each tile, which describes the sequence of the mapped computation and communication statements, as shown in Figure IV.7 inside the tiles on the right. When an actor is being executed in simulation, the analytical computation time model is called to compute the corresponding wait delay. During communications through channels, the communication time model is called to compute the delays of communications on the platform even in the case of contentions at shared resources. The simulation predicts the state of tiles (wait,

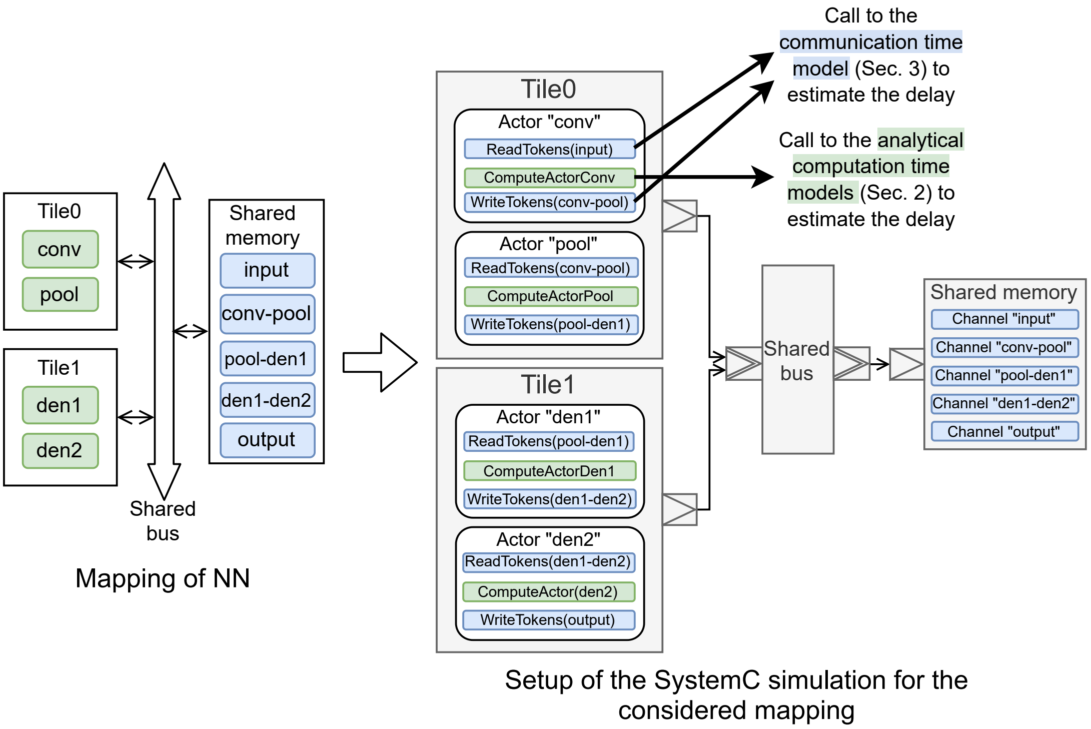


Figure IV.7 – Illustration of the use of the SystemC model to simulate the execution of a NN mapped onto a multi-core platform. The simulation calls the computation or communication time models based on the delay that need to be predicted.

read, write or compute) and shared resources during the execution of the NN on the platform.

Once the simulation terminated, the estimated execution traces are output, which contain the information about tiles and shared resource states as well as the actor being executed by tiles at any given time of the simulated execution. The latency and throughput of the NN mapping can be extracted from the execution traces. The number of iterations simulated by the SystemC model is specified by the user. Simulating several iterations is necessary to evaluate the effect on timing of The number of iterations must be high enough to properly evaluate the effect of streaming execution, i.e. the execution of several iterations of the NN on the platform simultaneously. It must also be low enough to save evaluation time. In our experiments, for each considered mapping, we empirically set the total number of iterations to 100. This number offers a good compromise between fast evaluation time and good prediction accuracy. To evaluate our approach, several mappings are simulated and the obtained results are compared with measurements.

IV.5 Experiment results

IV.5.1 Tested scenarios

We tested and stressed our timing modeling flow in regards to the following aspects:

1. Consideration of applications of different complexity. We considered several NNs, with different computation and communication workloads. We used SDF graphs with a low amount of actors and communication channels, respectively 2 and 3 as lowest (MLP1 C1 as shown in Table A.1 in Appendices), as well as complex SDF graph featuring up to 22 actors and 113 communication channels (MLP2 C7 as shown in Table A.3 in Appendices).
2. Consideration of single-core mappings as well as multi-core mappings containing up to 7 tiles. The multi-core mappings leverage both NN in-layer parallelism (clusters from a same layer mapped on different tiles to parallelize their computation) and intra-layer parallelism (mapping of layers on different tiles to parallelize the computation of layers and offer a streaming execution of the NN).
3. Consideration of mappings with high and low communication rates. The communication rate is defined as the percentage of time spent by all tiles on average in communication. The lowest observed communication rate is 2 % on one mapping and the highest is 70 % on another mapping.
4. Every mapping is tested both with and without power management, i.e. both with polling-based communications without the use of clock gating and with interrupt-based communications, in which cores are clock gated when waiting for the availability of data.

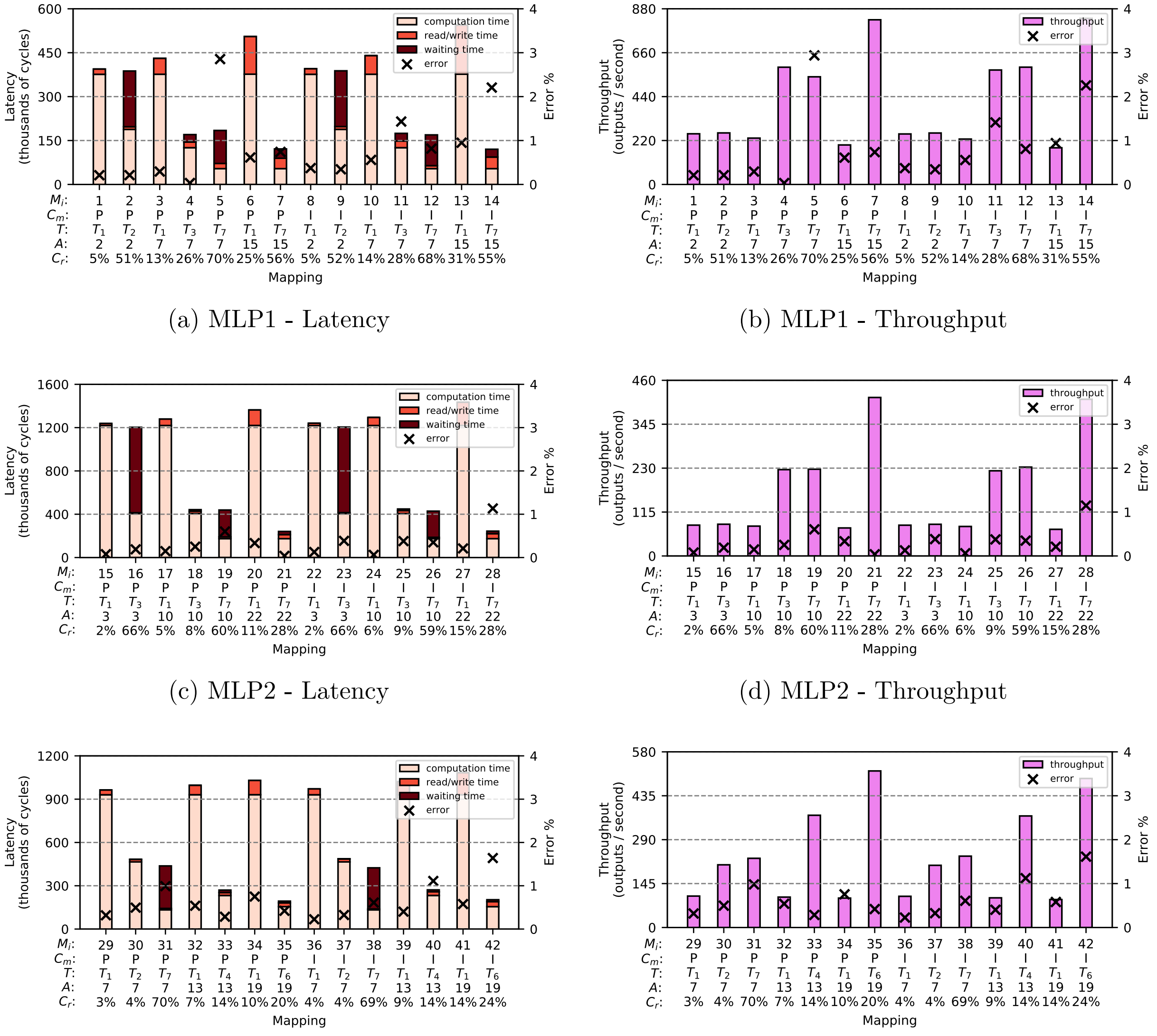
We tested 27 different mappings of the 4 considered NNs (CNN1 and all MLPs) with both communication procedures (which amounts to 54 mappings in total). All tested mappings are presented in details in Appendices A. We predict the end-to-end latency and throughput using the SystemC model and a pure analytical timing modeling flow presented in the next section. The predictions are compared with real end-to-end latency and throughput measured on the platform prototype. Two FPGA boards have been successively considered for the implementation of the platform prototype: the ZCU102 board, which features a UltraScale FPGA, and the ZC702 board, which features a Zynq7000 FPGA. The ZC702 was replaced by the ZCU102, which offered more resources. The difference in latency and throughput per mapping measured on the two boards is marginal.

IV.5.2 Pure analytical model for comparison against the simulation

In order to evaluate the contribution of simulation to model shared resource contentions and to compare our flow with respect to purely analytical modeling flows from related work, we introduce a pure analytical timing model for NNs on multi-core platforms. The pure analytical timing model evaluates the latency and throughput of cores based on the mapping, using the computation time model presented in Section IV.2 and the analytical model used inside the communication time model as presented in Section IV.3.1. The pure analytical timing model neglects shared resource contentions and therefore corresponds to a best case or lower bound model. The accuracy and evaluation time using both modeling flows are compared in Section IV.6.

IV.5.3 Validation results

Table IV.2 shows the average and maximum prediction error on end to end latency and throughput for single-core and multi-core mappings for all applications, using the simulation model and the pure analytical model. Prediction errors are computed using the formula $\frac{X_\mathrm{P}-X_\mathrm{M}}{X_\mathrm{M}}$ where X_P is the prediction and X_M is the measurement. The errors are provided in absolute value and percentage. In this table, both the mappings with polling-based and interrupt-based communications are provided. The detailed predictions of the simulation for both end to end latency and throughput with the error for every tested mapping are shown in Figure IV.8 and IV.9. For example, on Figure IV.8 (c) it can be observed that the lowest predicted latency for the MLP2 is reached for the mapping with index 21. The predicted latency's value can be read on the left Y axis, it is approximately equal to 241 thousands of cycles. The prediction error can be read on the right Y axis, it is approximately equal to 0.04 % for this mapping. The same process on Figure IV.8 (d) allows reading the predicted throughput's value, which is approximately equal to 415 inferences per second, and the prediction error on throughput, which is also equal to 0.04 %.



(a) MLP1 - Latency

(b) MLP1 - Throughput

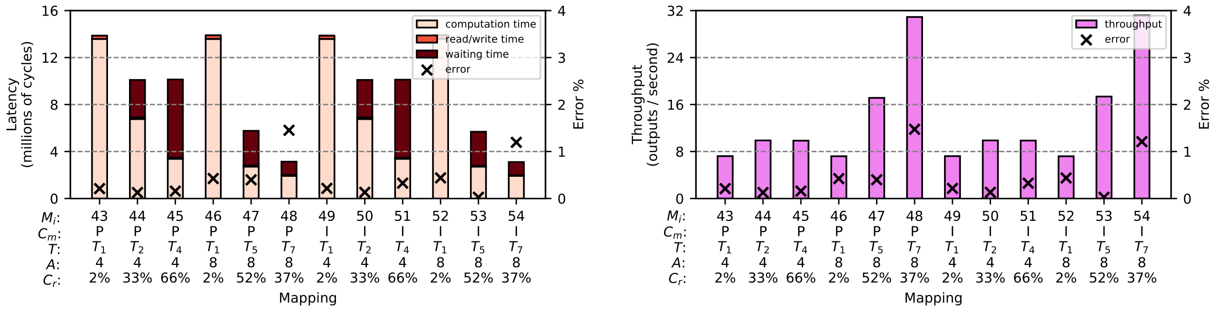
(c) MLP2 - Latency

(d) MLP2 - Throughput

(e) MLP3 - Latency

(f) MLP3 - Throughput

Figure IV.8 – Predicted end to end latency and throughput by the simulable model for the considered MLP mappings. The prediction error against measurements in absolute and percentage is also provided for every mapping. On the plot of the latency, the % of time spent on average by cores during the execution of the application in computation, read and write and waiting phases are depicted. In X-axis, information about the tested mapping is provided: the top number M_i is the mapping index, which can be used to find more information about the mapping in appendix to this manuscript. The number C_m below M_i is the communication mode - P stands for polling, while I stands for interrupt. The number of tiles used is indicated by the indice of T. Then respectively the number of actors A in the SDF graph and the average time spent in communication by tiles C_r (combination of read/write time and wait time) are provided.



(a) CNN - Latency

(b) CNN - Throughput

Figure IV.9 – Predicted end to end latency and throughput by the simulable model for the considered CNN mappings. The absolute prediction error against measurements is also provided for each mapping. More information about the legend of the plots can be found in the caption of Figure IV.8.

Table IV.2 – Observed average and maximum error against measurements on tested mappings regarding the end to end latency in processor cycles ($\mathcal{L}$) and the throughput in outputs/s (Φ). The column titled "# tested mappings" provides the total number of different mappings tested for each application. All details about the tested mappings can be found in appendix. In this table, the mappings using polling-based and interrupt-based communications are combined. The evaluation time using the simulation flow is ≈ 20 s when including compilation time. Without compilation time, it is in the order of tenth of seconds. The evaluation time using pure analytical models is in the order of ms.

Application	Mapping type	# tested mappings	Metric	Pure analytical model - Error %		Simulation-based flow - Error %	
				avg	max	avg	max
MLP1	Single-core	6	$\mathcal{L}$	1.07	4.94	0.50	0.95
			Φ	1.24	5.19	0.50	0.94
	Multi-core	8	$\mathcal{L}$	**7.63**	**22.13**	1.08	**2.85**
			Φ	**9.43**	**28.33**	1.09	**2.94**
MLP2	Single-core	6	$\mathcal{L}$	1.08	4.55	0.16	0.33
			Φ	0.96	3.49	0.16	0.33
	Multi-core	8	$\mathcal{L}$	3.91	**12.97**	0.42	1.13
			Φ	4.29	**14.86**	0.42	1.15
MLP3	Single-core	6	$\mathcal{L}$	1.42	4.41	0.47	0.75
			Φ	1.28	4.60	0.47	0.76
	Multi-core	8	$\mathcal{L}$	3.63	**11.52**	0.73	1.64
			Φ	3.86	**12.97**	0.73	1.61
CNN	Single-core	4	$\mathcal{L}$	1.32	2.49	0.32	0.44
			Φ	1.29	2.43	0.32	0.44
	Multi-core	8	$\mathcal{L}$	0.87	3.13	0.59	1.45
			Φ	0.89	3.23	0.60	1.47

IV.6 Discussions

Results overview: The proposed simulation-based flow allows predicting both the latency and throughput of the 54 mappings with more than 97 % accuracy. The worst observed error of the simulable model is 2.85 % on latency and 2.94 % on throughput. The model is also fast with an evaluation time of approximately 20 s per mapping.

This is rendered possible by the hypothesis of strict separation of computations and communications using SDF and the MoA, which allows building separate computation time models for NN layers and communication time model. The calibration through measurements of these models for a given core, compiler setting and communication medium allows high prediction accuracy. The analytical computation time model is validated by the high accuracy - 99.63 % on average for both latency and throughput - obtained on single-core scenarios, which have a limited amount of communications.

The use of simulation and the message level communication time model offer an accurate modeling of communications on the platform and contentions when multiple cores access the shared memory simultaneously. This allows offering high accuracy on multi-core mappings (99.29 % on average for both latency and throughput). In the following of this section, we will discuss the observed validation results in regards to the identified elements that render the evaluation of NNs on multi-core platforms difficult.

Evaluation speed: The evaluation speed of a NN mapping using the SystemC model is 20 s including compilation time. This high evaluation speed is rendered possible by the use of analytical models to predict computation and communication time inside our system level simulation. The evaluation of a mapping using the proposed modeling flow is 2 times faster than our automatized timing measurement infrastructure, which takes 40 s including also compilation and FPGA programming time, but excluding NN training, FPGA design synthesis, BSP generation and source code development. When also considering the NN training and SW/HW development of the application, the modeling flow offers a significant speed-up. Our SystemC model allows engineer to perform a fast and reliable evaluation of candidate NN deployments under latency and throughput constraints, while alleviating important effort spent on the development of NN applications on the targeted implementation platform.

Scalability regarding NNs: The highest prediction error is observed for the MLP1 with 0.83 % latency prediction error on average on all tested mappings. The maximum error on latency for the MLP1 is 2.85 %, which is also the highest observed error for all applications. We observe 0.31 % error on average for the MLP2, 0.62 % for the MLP3 and 0.43 % for the CNN. Overall the prediction error is slightly higher on the MLP1 as it is the application with the highest communication workload percentage compared to the computation workload. The error is still very low, which allows validating the scalability of the simulation-based flow in consideration of the application. Similar observations are done regarding throughput.

Scalability regarding communication procedure: The average prediction error is 0.49 % on the 22 mappings using polling-based communications without clock gating and 0.61 % on the 22 mappings using interrupt-based communications with clock gating. The maximum error is 2.85 % for polling-based communications, and 2.21 % for interrupt-based communications. The accuracy of the model appears to be stable from one communication procedure to the other. This validates the scalability of the proposed timing modeling flow in regards to the communication procedure.

Scalability regarding the number of cores used: The evolution of the prediction error of the proposed modeling flow based on the number of cores in the mapping can be observed in Table IV.3. While the error is stable for mappings relying on 1 up to 5 cores, it raises up to 1.03 % and 1.08 % on average for mappings using respectively 6 and 7 tiles. The prediction accuracy of the modeling flow remains acceptable and does not seem to correlate directly with the number of cores used. It is important to note though that the number of mappings with 5 and 6 cores is low, which can add a bias to these results. This allows anyhow validating the scalability of the modeling flow in regards to the number of cores used in the mapping.

Table IV.3 – Summary of the average and maximum prediction error on latency (in absolute value) of the simulation-based power modeling flow based on (a): the number of cores used in the mapping (b): the communication rate. Similar results are observed for throughput.

(a) Core number

Core number	1	2	3	4	5	6	7
Average error	0.37	0.27	0.44	0.46	0.21	1.03	1.08
Max error	0.95	0.49	1.43	1.11	0.40	1.64	2.85
Total mapping number	22	6	6	4	2	2	12

(b) Communication rate

Comm. rate	0-9%	10-19%	20-29%	30-39%	40-59%	+60%
Average error	0.28	0.50	0.76	0.77	0.65	0.72
Max error	0.54	1.11	1.63	1.45	2.21	2.85
Total mapping number	17	9	7	5	6	10

Scalability regarding communication rates: Table IV.3 provides the prediction error of the modeling flow based on the average percentage of time spent by tiles in communication for the tested mapping. The average prediction error increases with the amount of communications for communication rates ranging up to 29 %. It then remains stable around 0.7 % regardless of the communication rate. However the observed maximum error keeps raising with the communication rate up to 2.85 % for communication rates higher than 60 %. It can be noted that such communication rates are however high and were used in our test cases to stress and thoroughly validate the modeling flow. They are not usually found for real relevant mappings, as it results from a poor usage of resources since in such settings tiles spend more time waiting for resource than contributing by doing computations. The error remains acceptable for such high communication rates, which allows validating the scalability of the modeling flow in this regard.

Comparison against the pure analytical model: The time taken to evaluate a mapping using the purely analytical model is of the order of a millisecond, i.e. 10^4 an order of magnitude faster than the SystemC model. The evaluation speed of a mapping using the pure analytical timing model is approximately 1 ms, which is faster than the SystemC model. However, the pure analytical timing model always underestimates the effect of shared resources on the latency and throughput. For example, the highest observed error of the pure analytical model to 28.33 % on throughput and −22.13 % on latency on the mapping with index 14 of MLP1 with clustering $C = 7$, 7 tiles used and interrupt-based communications. The fact that the prediction error raises with the communication rate due to the important use of shared resources make this model unreliable for the evaluation of timing properties of NNs deployed on multi-core platforms. These results highlight the need to use the proposed simulation-based flow over pure analytical modeling flows for such platforms. Due to its fast evaluation time, the pure analytical modeling flow can however be used in conjunction with the simulation flow to offer a fast pruning of the design space before the evaluation of relevant mappings by the SystemC model, which offers slower but more reliable predictions.

Conclusion: The results allow validating the use of the proposed simulation flow for fast-yet-accurate evaluation of NN mappings onto multi-core platforms. The flow offers indeed an accuracy of more than 97 % on timing for an evaluation time of 20 s per mapping. Its applicability in consideration to NN applications, clustering, number of tiles, communication rate and communication procedure are successfully validated. In comparison, pure analytical modeling flows show limitations as they cannot efficiently predict the effect of shared resources on timing properties.

Due to its high accuracy, evaluation speed and scalability, the proposed modeling flow can be used to efficiently explore the design space of NN deployments onto multi-core platforms. It allows finding NN deployments that optimize timing properties on a user defined multi-core platform, as well as optimizing multi-core platform architectures for NN deployments under timing constraints.

IV.7 Conclusion

In this chapter, we presented the proposed modeling flow for the timing of NNs deployed on multi-core platforms. The presented contributions are

1. the analytical computation time model, which allows predicting the execution time of any actor issued from the clustering of a NN in SDF,
2. the extension of the communication time model used in previous work to support interrupt-based communications,
3. the integration of the computation time model and communication time model inside a SystemC simulation to model shared resources.

The modeling flow is validated against latency and throughput measurements from a real implementation of 54 mappings of 4 different NNs, with various clustering complexity, low and high communication rates, and with both polling-based and interrupt-based communications. It allows predicting the latency and throughput of the tested mappings with more than 97 % accuracy. In addition to its high accuracy, the modeling flow offers fast predictions with an average prediction time of approximately 20 s per mapping. We compared our modeling flow to a pure analytical model with respect to propositions from related work, which showed limitations to predict efficiently the latency and throughput when applied to multi-core mappings using shared resources.

This is rendered possible by the separation of computation and communication respected by our MoC and MoA, which allows building separate computation and communication time models. Both models are calibrated through measurements, which allows modeling the effect of the HW/SW implementation and compiler settings and offer high accuracy. They are used inside a simulation captured in SystemC of the platform infering the mapping, which allows modeling the shared resources. The high level of abstraction of all considered models allow offering high evaluation speed.

The computation time models were calibrated for MicroBlaze cores using FPUs and multipliers with compiler setting "-O0". They must be recalibrated for use with other core types and compiler settings. This can represent an important effort, but it is necessary to guarantee high accuracy. The communication time model was calibrated for a shared interconnect AXI bus with a First Come First Served policy. It can be re-used for the same communication medium without re-performing calibrations. However, if another communication medium or another arbiter policy is used, it must also be re-calibrated. In the scope of this work, we tested MLPs and CNNs, but the models could be extended to other classes of NNs by proposing analytical computation time models using the same approach. In the next chapter, we present how this timing modeling flow is extended to allow power and energy predictions.

Chapter V

Power and energy modeling and analysis flow

In this chapter, we present the power and energy prediction flow for NNs deployed on multi-core platforms that respects our MoA. The proposed flow enhances our simulation-based flow for timing prediction presented in Chapter IV. It is used to predict power and energy consumption in regards to the phases executed by tiles and possible resource contention. One benefit of our modeling approach is the combination of simulation and measurement-based characterization, which allows offering scalability in regards to the clustering, mapping, communication workload and power management. We present an additional calibration step to extend the power model's applicability to platforms with different number of tiles and private memory sizes, at the cost of a more intensive characterization effort. We evaluate our power and energy modeling flow on 27 mappings of 4 NNs with and without power management, and 7 different platform versions.

V.1 Power modeling and analysis flow overview

Embedded platforms come with strong energy constraints, while NNs are power intensive applications due to computations. To deploy NNs onto such platforms, it is therefore essential to optimize deployments under power and energy constraints. As discussed in Chapter II, most approaches for the evaluation of NNs on embedded platforms show limitations when applied to tile-based multi-core platforms.

The proposed power model is separated into two main terms: static and dynamic power consumption. The dynamic power consumption model captures the contribution of tiles to power in the different phases of NN inference, such as computation of layers, and communications on shared memory. It also takes shared resource contentions into account. The static power consumption model allows capturing the power consumption of the platform when not clocked, including the power consumption of tiles when clock gated (if using power management). Both models use the simulation presented in Chapter IV to obtain information regarding the use of shared resources and the phase in which tiles are during the execution. The proposed power modeling flow is based on quantities calibrated through measurements. We also present an additional calibration approach to extend the static power model to platforms with varying number of tiles and private memory sizes to enable the use of the model to size multi-core platforms in a DSE setup. The proposed power and energy modeling flow can achieve two objectives:

1. Evaluate power and energy for NNs deployments on a user-defined multi-core platform and identify optimized mappings,
2. Jointly evaluate and optimize multi-core platform architectures and NN deployments under power and energy constraints.

In Figure V.1 the proposed power and energy model is represented in yellow in the middle. It is presented in Section V.2. As shown in the legend, the methodology to obtain the power model is split into two main phases: the first phase (❶ - ❹) colored in red corresponds to its calibration, and the second phase (❺ - ❽) colored in teal corresponds to its validation and application. To perform the calibration through measurements, we focus on the prototype implementation platforms presented in Chapter III. As explained in Section III.3.2, two main platforms are considered: one with power management which implements interrupt-based communications and clock gating, and one without, which implements polling-based communications without clock gating.

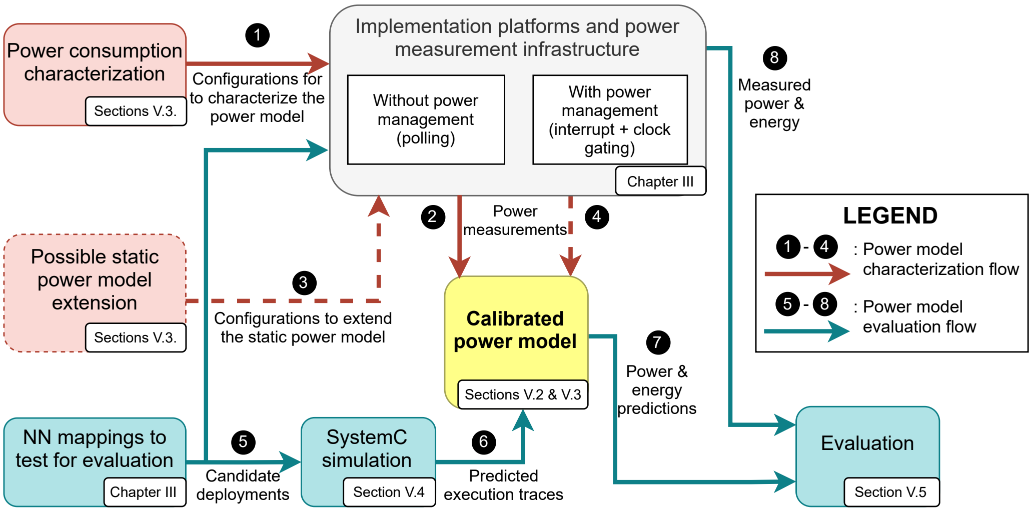

Power consumption characterization
Sections V.3.
❶
Configurations for to characterize the power model
Implementation platforms and power measurement infrastructure
Without power management (polling)
With power management (interrupt + clock gating)
Chapter III
❽
Measured power & energy
❷
Power measurements
❹
Possible static power model extension
Sections V.3.
❸
Configurations to extend the static power model
Calibrated power model
Sections V.2 & V.3
❼
Power & energy predictions
LEGEND
❶ - ❹ : Power model characterization flow
❺ - ❽ : Power model evaluation flow
NN mappings to test for evaluation
Chapter III
❺
Candidate deployments
SystemC simulation
Section V.4
❻
Predicted execution traces
Evaluation
Section V.5

Figure V.1 – Overview of the methodology to obtain a power model for power and energy prediction of NNs on multi-core platforms.

In Section V.3, we explain the calibration methodology of the power model, and we present the calibration's results on our platform. ❶ and ❷ correspond to the characterization of static and dynamic power consumption of tiles. A multi-linear regression is applied on the measured power profiles in order to obtain the calibrated power model. Optionally, ❸ and ❹ allow extending the calibrated power model to offer scalability in regards to the number of tiles in the platform and private memory sizes. The resulting power model allows estimating the power consumption of multi-core platforms executing NNs, while being scalable in consideration to the platform's dimensions.

Once the power model obtained, we proceed with its use and evaluation. The SystemC model generates execution traces ❻ for candidate mappings ❺, which are used by the power model to predict the power consumption and energy of the system inferring these mappings ❼. The use of the SystemC simulation with the power model is described in Section V.4. The power and energy predictions are finally compared with measured power and energy from a real implementation of the tested mappings ❽ to evaluate the accuracy of the model. The validation methodology and results are given and discussed in Section V.5. We validate our modeling flow with a total of 27 different mappings of the 4 considered NNs tested with and without power management [1]. These different mappings allow verifying the scalability of the power model to the criterias outlined above. We also tested its applicability to a total of 7 different prototype platforms implemented on FPGA, including single and multi-core platforms with varying size of private memory.

V.2 Power model proposal

Coarsed-grained model: To obtain the power consumption of the system $P_{\text{total}}(t)$, we must consider both static and dynamic power consumption. We consider that the static power consumption P_{static} is the power consumption of the system when supplied with power but not clocked. When using power management, additional components are implemented to support interrupt-based communications and clock gating. We thus introduce different models for the static power consumption of the platform: with power management $P_{\blacktriangle,\,\text{static}}$ and without $P_{\vartriangle,\,\text{static}}$.

Regarding dynamic power consumption, when executing NNs, tiles execute computation and communication activities, which are strictly separated due to SDF and our MoA. The different tile phases during the execution of NN mappings are illustrated on the execution

[1] In total, this amounts to $2 \times 27 = 54$ mappings.

diagram on the right of Figure V.2 when not using power management: the phase in green corresponds to computations, and phases in blue (read, write, wait) correspond to communications. We describe the dynamic power consumption of the platform executing NNs using the following terms:

— $P_{\text{comp}}(t)$, the total power consumption of tiles in computation (comp.) phase at time t.

— $P_{\blacktriangle,\,\text{comm}}(t)$, the total power consumption in communication (comm.) phase at time t when using power management, and $P_{\triangle,\,\text{comm}}(t)$ when not using power management.

To summarize, the total power consumption of the system at time t is provided in Equation V.1 and Equation V.2 respectively when using ($\blacktriangle$) / not using ($\triangle$) power management.

$$P_{\blacktriangle}(t) = P_{\blacktriangle,\,\text{static}} + P_{\text{comp}}(t) + P_{\blacktriangle,\,\text{comm}}(t) \tag{V.1}$$

$$P_{\triangle}(t) = P_{\triangle,\,\text{static}} + P_{\text{comp}}(t) + P_{\triangle,\,\text{comm}}(t) \tag{V.2}$$

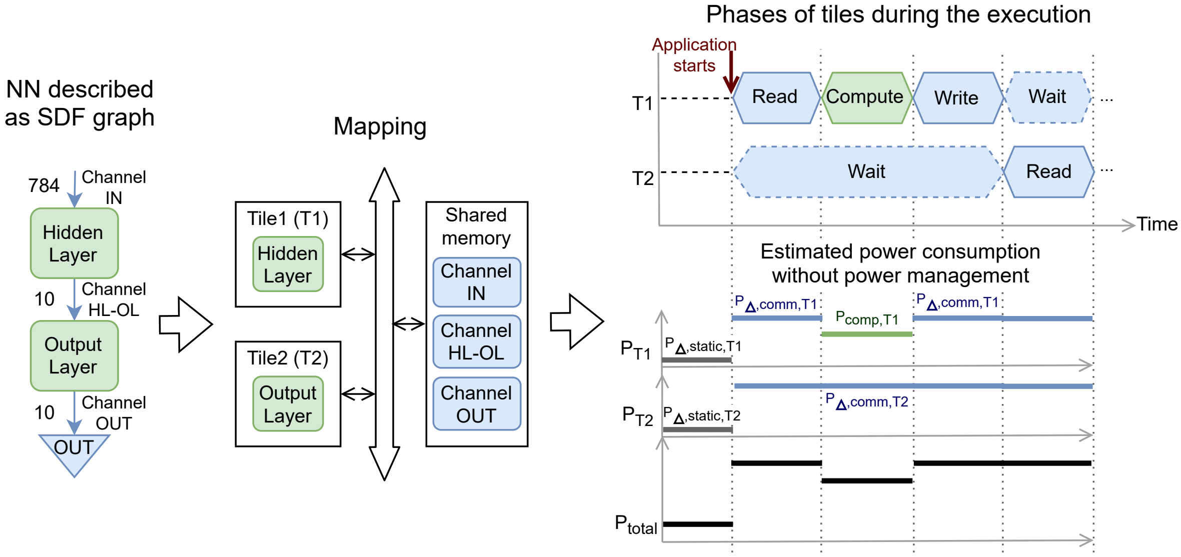


Figure V.2 – Estimation of power consumption in regards to the phase of tiles during the execution of NN mappings. Possible phases are: computation, read/write on shared memory and waiting for buffer availability. The estimation of power consumption on the bottom right of the figure is provided for a platform without power management (△).

Refinement of $P_{\mathbf{comp}}(t)$: When tiles are in computation phase, they are performing the computations of the neural network (e.g. processing neurons from dense layers). In this phase, tiles are independent from one another as all necessary data is contained in the private memory of the tile. The proposed model for $P_{\text{comp}}(t)$ is thus linear in consideration of tiles in computation phase. The model for $P_{\text{comp}}(t)$ is provided in Equation V.3.

$$\begin{aligned} P_{\text{comp}}(t) &= \sum_{i=1}^{T} P_{\text{comp},i}\ \alpha_{\text{comp},i}(t) \\ \text{with} \quad \alpha_{\text{comp},i}(t) &= \begin{cases} 1 & \text{if tile } i \text{ is in computation phase at time } t \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{V.3}$$

In this equation, T denotes the number of tiles in the platform. $P_{comp,i}$ is the power consumption of tile i in computation phase, with with $1 \leq i \leq T$. $\alpha_{comp,i}(t)$ is a factor indicating if tile i is in computation phase at time t: it is equal to 1 if tile i is in computation phase at time t, else it is equal to 0. The $\alpha_{\text{comp},i}(t)$ are obtained from the execution traces of the simulation.

Refinement of $P_{\mathbf{comm}}(t)$: When tiles are in the communication activity, they either: 1. perform a read or write in the shared memory, 2. wait for the availability of data. As presented in Chapter III, with power management, tiles enter clock gated mode when waiting for data, and without, tiles perform active waiting by polling the shared memory. When clock gated, tiles are by definition not supplied with the clock. They contribute theoretically only to the static power consumption of the system $P_{\blacktriangle,\,\text{static}}$, and are thus not modeled in $P_{\blacktriangle,\,\text{comm}}(t)$. Read, write and polling operations are interleaved and correspond to the total power consumption of tiles accessing the shared memory at time t. The shared memory is a unique resource. When several tiles try to access the shared memory simultaneously, the bus arbiter gives the access to only one tile and the others are paused until the shared memory access is free. The shared memory can thus be accessed by only one tile at any given time t. To simplify our model, we assume that tiles in read, write and polling phases share the same dynamic contribution to power denoted P_{sm}, which corresponds to the power consumption of tiles accessing the shared memory. In light of this, the power consumption $P_{\text{comm}}(t)$ is either equal to P_{sm} when at least one tile is accessing the shared memory, or 0 otherwise. The proposed models for $P_{\text{comm}}(t)$ with and without the use of power management ($\blacktriangle/\triangle$) are given in Equations V.5 and V.4.

$$P_{\blacktriangle,\,\mathrm{comm}}(t) = P_{\blacktriangle,\,\mathrm{rw}}(t) = P_{\mathrm{sm}} \left(1 - \prod_{i=1}^{T} \left(1 - \alpha_{\mathrm{rw},\,i}\,(t)\right)\right) \tag{V.4}$$

$$= \begin{cases} P_{\mathrm{sm}} & \text{if at least one tile is reading} \\ & \text{or writing on shared memory at time } t \\ 0 & \text{otherwise} \end{cases}$$

$$P_{\triangle,\,\mathrm{comm}}(t) = P_{\triangle,\,\mathrm{rwp}}(t) = P_{\mathrm{sm}} \left(1 - \prod_{i=1}^{T} \left(1 - \alpha_{\mathrm{rwp},\,i}\,(t)\right)\right) \tag{V.5}$$

$$= \begin{cases} P_{\mathrm{sm}} & \text{if at least one tile is reading, writing} \\ & \text{or polling on shared memory at time } t \\ 0 & \text{otherwise} \end{cases}$$

In this equation, T denotes the number of tiles in the platform. P_{sm} is the power consumption of one tile accessing the shared memory. $\alpha_{\mathrm{rwp},\,i}\,(t)$ is a factor indicating if tile i is accessing the shared memory at time t when not using power management: it is equal to 1 when it is the case (tile i is in polling, read or write phase), else it is equal to 0. $\alpha_{\mathrm{rw},\,i}\,(t)$ is the factor defined similarly when using power management, which thus excludes the waiting phases: it is equal to 1 when tile i is in read or write phase, else it is equal to 0. The α terms are obtained from the execution traces of the simulation, as presented in Section V.4.

Possible refinement of $P_{\mathbf{static}}$: The static power consumption is highly dependent on the number of tiles implemented in the system and the private memory sizes. In order to offer scalability in regards to these aspects, we propose a refined model for P_{static}. This refined model must be considered to jointly size a multi-core platform and optimize NN deployment under energy constraints. In Equation V.6 we propose the model for $P_{\blacktriangle,\,\mathrm{static}}$ for platforms with power management, and in Equation V.7 the model $P_{\triangle,\,\mathrm{static}}$ for ones without power management.

$$P_{\blacktriangle,\,\text{static}}(T,\mathcal{M}) = \sum_{i=1}^{T} P_{\text{tile, mem}}\,\mathcal{M}_i + T P_{\text{tile, core}} + P_{\text{circuit}} + P_{\blacktriangle,\,\text{components}} \tag{V.6}$$

$$P_{\triangle,\,\text{static}}(T,\mathcal{M}) = \sum_{i=1}^{T} P_{\text{tile, mem}}\,\mathcal{M}_i + T P_{\text{tile, core}} + P_{\text{circuit}} \tag{V.7}$$

In these equations, T is the number of tiles in the considered platform. The term $\mathcal{M}_i$ corresponds to the private memory size[2] of tile i. The private memory of a tile is used to store its instructions and data. The term $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2 \dots \mathcal{M}_T\}$ is a vector of T elements that corresponds to the private memory size of each tile. $P_{\text{tile, mem}}$ is the base contribution to static power consumption of a unit of private memory. $P_{\text{tile, core}}$ is the contribution of the processor core of tiles to the static power consumption, which is independent of the private memory size. P_{circuit} is the contribution of the remainder of the circuit to the static power consumption, which corresponds to the shared memory and shared interconnect. Finally, $P_{\blacktriangle,\,\text{components}}$ is the contribution to the static power consumption of the additional components used to support power management, i.e. the interrupt and clock gating controllers.

Resulting model of system's power consumption: When taking into account the aforementioned refinements, we obtain the system's power consumption model as shown in Equations V.8 and V.9.

$$\begin{aligned}
P_{\text{total},\triangle}(t) &= P_{\triangle,\,\text{static}}(T,\mathcal{M}) + P_{\text{comp}}(t) + P_{\triangle,\,\text{comm}}(t) \\
&= \sum_{i=1}^{T} P_{\text{tile, mem}}\,\mathcal{M}_i + T P_{\text{tile, core}} + P_{\text{circuit}} \\
&\quad + \sum_{i=1}^{T} P_{\text{comp},i}\;\alpha_{\text{comp},i}(t) \;+ P_{\text{sm}}\left(1 - \prod_{i=1}^{T}\left(1 - \alpha_{\text{rwp},\,i}\,(t)\right)\right)
\end{aligned} \tag{V.8}$$

$$\begin{aligned}
P_{\text{total},\blacktriangle}(t) &= P_{\blacktriangle,\,\text{static}}(T,\mathcal{M}) + P_{\text{comp}}(t) + P_{\blacktriangle,\,\text{comm}}(t) \\
&= \sum_{i=1}^{T} P_{\text{tile, mem}}\,\mathcal{M}_i + T P_{\text{tile, core}} + P_{\text{circuit}} + P_{\blacktriangle,\,\text{components}} \\
&\quad + \sum_{i=1}^{T} P_{\text{comp},i}\;\alpha_{\text{comp},i}(t) \;+ P_{\text{sm}}\left(1 - \prod_{i=1}^{T}\left(1 - \alpha_{\text{rw},\,i}\,(t)\right)\right)
\end{aligned} \tag{V.9}$$

[2] In our work, this value is in kB.

Adaptations due to our implementation technology: In our work, due to the technology selected to implement our platform prototypes, we adapted the model to improve accuracy. Our platform prototypes are implemented on the ZCU102 board, which features a UltraScale+ MPSoC [87]. When implementing our platform as presented in Chapter III, additional components are implemented on the programmable logic section of the FPGA to support the use of several peripherals by the arm cores present in the SoC, even though they are not used in the scope of our work. We also implement a MicroBlaze debugger IP provided by AMD for each tile, and one shared UART peripheral, which can be accessed through the bus. These additional components are used in order to support debug of the software executed by tiles. When performing the power measurements, these components are not used and thus they do not contribute to the dynamic power consumption. However, they contribute to the static power consumption, in particular P_{circuit}. On the FPGA chip, there are also LUTs that are not used for the system's implementation, but are still powered. They thus have a contribution to the static power consumption too. Their contribution in P_{circuit} is captured when performing the model's calibration.

When using power management, the contribution of clock gated tiles on the system's power consumption is also influenced by our implementation technology. In fact, we observe that, when using clock gating on our platform prototype, the clock is not delivered to both the processing core and private memory block. However, when performing the measurement of static power consumption, tiles are stopped through the use of the debugger IPs, which actually blocks the delivery of the clock to the core but still delivers it to the private memory block. In this case, the private memory block of the tile bears an additional contribution to power, which is captured in the static power consumption calibrated value but is not observed when using clock gating. Due to this difference, we introduce a complementary static power consumption model for tiles in clock gating phase. The total contribution of clock gated tiles to the system's power consumption is modeled as $P_{\blacktriangle,\,\text{cg}}(t)$ as shown in Equation V.10. This contribution is negative due to the difference between the observed static power consumption of the platform when tiles are stopped and when they are clock gated.

$$P_{\blacktriangle,\,\text{cg}}(t) = -\sum_{i=1}^{T} P_{\text{cg},i}\ \alpha_{\text{cg},i}(t) \tag{V.10}$$

In this equation, T denotes the number of tiles in the platform. $P_{\text{cg},i}$ is the power consumption of tile i when clock gated: it corresponds in fact to the difference between the power consumption of tile i when only the core has the clock removed and when both core and private memory block have the clock removed. $\alpha_{cg,i}(t)$ is a factor indicating if tile i is in clock gated phase at time t. The model for the consumption of tiles in communication phase with power management $P_{\blacktriangle,\,\mathrm{comm}}(t)$ provided in Equation V.4 thus becomes (Equation V.11):

$$P_{\blacktriangle,\,\mathrm{comm}}(t) = P_{\blacktriangle,\,\mathrm{rw}}(t) \; + P_{\blacktriangle,\,\mathrm{cg}}(t) = P_{\mathrm{sm}} \left(1 - \prod_{i=1}^{T} \left(1 - \alpha_{\mathrm{rw},\,i}\,(t)\right)\right) - \sum_{i=1}^{T} P_{\mathrm{cg},i}\; \alpha_{\mathrm{cg},i}(t) \quad \text{(V.11)}$$

And the model for $P_{\mathrm{total},\blacktriangle}(t)$ given in Equation V.9 is refined in Equation V.12. The term $P_{\mathrm{cg},i}$ is a base power consumption term.

$$\begin{aligned} P_{\mathrm{total},\blacktriangle}(t) &= P_{\blacktriangle,\,\mathrm{static}}(T,\mathcal{M}) + P_{\mathrm{comp}}(t) + P_{\blacktriangle,\,\mathrm{comm}}(t) \qquad \text{(V.12)}\\ &= \sum_{i=1}^{T} P_{\mathrm{tile},\delta}\,\mathcal{M}_i + T P_{\mathrm{tile,\ fix}} + P_{\mathrm{circuit}} + P_{\blacktriangle,\,\mathrm{components}} \\ &\quad + \sum_{i=1}^{T} P_{\mathrm{comp},i}\;\alpha_{\mathrm{comp},i}(t) \; + P_{\mathrm{sm}} \left(1 - \prod_{i=1}^{T} \left(1 - \alpha_{\mathrm{rw},\,i}\,(t)\right)\right) - \sum_{i=1}^{T} P_{\mathrm{cg},i}\;\alpha_{\mathrm{cg},i}(t) \end{aligned}$$

In the next section, we explain how the power model is calibrated through measurements.

V.3 Power model calibration

V.3.1 Calibration methodology

V.3.1.1 Main calibration approach through measurements

The base power consumption terms P_{sm}, $P_{\mathrm{comp},i}$, $P_{\mathrm{cg},i}$ used in the power model in Equations V.8 and V.12 are calibrated through measurements (Steps ❶ and ❷ in Figure V.1). Calibrating a power model through measurements allows offering a more accurate representation of the system's power consumption than using estimates. It allows finely capturing the effect of the compiler and of the hardware architecture.

In this step the dynamic power consumption of tiles in the different phases are measured, and the measured data is regressed in order to obtain the calibrated power model. The regression allows setting the terms P_{sm}, $P_{\mathrm{comp},i}$, $P_{\mathrm{cg},i}$ based on the data measured for the platform set in numerous different configurations. In this work, we use multi-linear regression. Since $P_{\blacktriangle,\,\mathrm{rw}}(t)$ and $P_{\vartriangle,\,\mathrm{rwp}}(t)$ are not linear, they cannot be modeled using this regression method. For these models, in addition to the multi-linear regression process we thus calibrate P_{sm} directly with measured data and verify if it suits the gathered power profiles. The regression also allows calibrating the value of $P_{\vartriangle,\,\mathrm{static}}(T,\mathcal{M})$ and $P_{\blacktriangle,\,\mathrm{static}}(T,\mathcal{M})$, with the number of tiles T and private memory sizes $\mathcal{M}$ corresponding to those of the prototype platform used.

The multi-linear regression aims at solving a system of equations denoted $(\mathcal{S})$, which is expressed in matrix form in Equations V.13 and V.14. In our case we perform the regression two times: one time for the platform with power management $(\mathcal{S}_{\blacktriangle})$ containing $K_{\blacktriangle}$ equations and one time for the platform without power management $(\mathcal{S}_{\vartriangle})$ containing $K_{\vartriangle}$ equations.

$$(\mathcal{S}_{\vartriangle}): \quad \mathbf{A}_{\vartriangle} \cdot \mathbf{P}_{\vartriangle,\,\mathrm{base}} = \mathbf{P}_{\vartriangle} \tag{V.13}$$

$$(S_{\blacktriangle}): \quad \mathbf{A}_{\blacktriangle} \cdot \mathbf{P}_{\blacktriangle,\,\mathrm{base}} = \mathbf{P}_{\blacktriangle} \tag{V.14}$$

In these equations, the unknowns to determine are represented by the vectors $\mathbf{P}_{\vartriangle,\,\mathrm{base}}$ and $\mathbf{P}_{\blacktriangle,\,\mathrm{base}}$. They correspond to the base power consumptions used inside our model (e.g.: P_{comp}). The vectors $\mathbf{P}_{\vartriangle}$ and $\mathbf{P}_{\blacktriangle}$ correspond to the measured values of the system's power consumption in the configurations of the platform specified inside the tile configuration matrices $\mathbf{A}_{\blacktriangle}$ and $\mathbf{A}_{\vartriangle}$.

Base power consumptions to determine $\mathbf{P}_{\mathbf{base}}$: The unknowns $\mathbf{P}_{\vartriangle,\,\mathrm{base}}$ and $\mathbf{P}_{\blacktriangle,\,\mathrm{base}}$ correspond to the base power consumption terms. Those terms are P_{comp} and P_{sm}, which are common to both platforms, $P_{\vartriangle,\,\mathrm{static}}$ for the platform without power management, and $P_{\blacktriangle,\,\mathrm{static}}$ and P_{cg} for the platform with power management. The definition of $\mathbf{P}_{\vartriangle,\,\mathrm{base}}$ and $\mathbf{P}_{\blacktriangle,\,\mathrm{base}}$ are given in Equations V.15 and V.16.

$$\mathbf{P}_{\triangle,\text{ base}} = \begin{pmatrix} P_{\triangle,\text{ static}} \\ P_{\text{comp},1} \\ P_{\text{comp},2} \\ \vdots \\ P_{\text{comp},T} \\ P_{\text{sm},1} \\ P_{\text{sm},2} \\ \vdots \\ P_{\text{sm},T} \end{pmatrix} \tag{V.15}$$

$$\mathbf{P}_{\blacktriangle,\text{ base}} = \begin{pmatrix} P_{\blacktriangle,\text{ static}} \\ P_{\text{comp},1} \\ P_{\text{comp},2} \\ \vdots \\ P_{\text{comp},T} \\ P_{\text{sm},1} \\ P_{\text{sm},2} \\ \vdots \\ P_{\text{sm},T} \\ P_{\text{cg},1} \\ P_{\text{cg},2} \\ \vdots \\ P_{\text{cg},T} \end{pmatrix} \tag{V.16}$$

Measured values of the system's power consumption $\mathbf{P_{total}}$: To solve $(\mathcal{S})$, measurements of the system's power consumption are required. Each power measurement is performed on the platform set in configurations specified in the matrix $\mathbf{A}$ are required. $K_\triangle$ and $K_\blacktriangle$ denotes the total number of equations in $(\mathcal{S})$. The vectors $\mathbf{P}_{\text{total},\triangle}$ and $\mathbf{P}_{\text{total},\blacktriangle}$ provided in Equations V.17 and V.18 contains the measured power consumption of the system in the tested configurations.

$$\mathbf{P}_{\text{total},\triangle} = \begin{pmatrix} P_{\text{total},\triangle,k=1} \\ P_{\text{total},\triangle,k=2} \\ \vdots \\ P_{\text{total},\triangle,k=K_\triangle} \end{pmatrix} \tag{V.17}$$

$$\mathbf{P}_{\text{total},\blacktriangle} = \begin{pmatrix} P_{\text{total},\blacktriangle,k=1} \\ P_{\text{total},\blacktriangle,k=2} \\ \vdots \\ P_{\text{total},\blacktriangle,k=K_\blacktriangle} \end{pmatrix} \tag{V.18}$$

Considered configurations and example: Performing the multi-linear regression to get the calibrated values of all P_{base} terms amounts to solving the system of equations $(\mathcal{S})$. Each configuration is specified as a line in the matrix $\mathbf{A}$ and corresponds to an equation in the system $(\mathcal{S})$. At least one configuration is thus required for every unknown variable P_{base}. In addition to the minimum number of configurations, we also consider additional ones to further stress the multi-linear regression, which allows reinforcing the quality of the calibration. This is necessary as power measurements bear measurement uncertainty, which effect can be mitigated by adding complementary configurations. This also allows verifying the proposed analytical model for power consumption, by testing its applicability to the additional cases.

To help the reader understand the tested configurations, we provide in Equation V.19 an example of matrix $\mathbf{A}_{\triangle}$ in the case of a platform without power management and composed of 3 tiles. At the top of the matrix are the α terms corresponding to the matrix index in the column. These α terms define the setting of tiles in the test configuration. For example, on the second line of the matrix in Equation V.19, the second term, corresponding to $\alpha_{\text{comp},1}$ is set to 1, which means that tile 1 is set in computation phase. All other α terms are set to 0, which means that only tile 1 is executing. The first term is always equal to 1, it corresponds to the static power consumption. To the right of the matrix, markers indicate the different configuration types tested.

Equation V.20 gives the expression of the system of equations to be solved $(\mathcal{S}_{\triangle})$ in the case of the considered example with the platform containing 3 tiles and implementing no power management. It can be noted that the configurations marked by I and II are sufficient to solve the system $(\mathcal{S}_{\triangle})$, but additional configurations III and IV are also considered to improve the quality of the calibration.

Once the measurements obtained in the different configurations, the data is regressed. In the following section, we explain how to extend the power model calibration to extend its scalability in regards to platform of different dimensions with reduced calibration effort.

The following configurations are considered:

I All tiles are disabled in order to characterize $P_{\triangle,\,\text{static}}$ / $P_{\blacktriangle,\,\text{static}}$. For the configuration marked by I on Equation V.19, we can see that only the static component is tested.

II One tile is set to execute the phase at test when others are deactivated to characterize the contribution to power consumption of each individual tile. For configurations marked by II on Equation V.19, we can see that only one tile is activated at a time.

III Tiles are progressively tasked to execute the phase at test simultaneously to characterize the evolution of power consumption when several tiles are enabled.

IV Complementary configurations are also considered to further constraint and improve the quality of the characterization. Testing all possible configurations on platforms with numerous tiles would require an important characterization effort. Instead we generated randomly a tenth of additional configurations in which tiles are arbitrarily either set in activity or disabled. In our example on Equation V.19, all possible complementary configurations were considered. This is because, in the case of this example, there are only 4 additional configurations in total.

$$\mathbf{A}_{\triangle} = \begin{array}{ccccccc|l}
\underline{1} & \underline{\alpha_{\text{comp},1}} & \underline{\alpha_{\text{comp},2}} & \underline{\alpha_{\text{comp},3}} & \underline{\alpha_{\text{rwp},1}} & \underline{\alpha_{\text{rwp},2}} & \underline{\alpha_{\text{rwp},3}} & \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & \text{I} \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & \text{II} \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & \text{III} \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & \text{IV} \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & \\
1 & 0 & 0 & 0 & 0 & 1 & 1 &
\end{array} \tag{V.19}$$

$$(\mathcal{S}_{\triangle}): \quad \mathbf{A}_{\triangle} \cdot \mathbf{P}_{\triangle,\,\text{base}} \;=\; \mathbf{P}_{\triangle,\text{total}} \tag{V.20}$$

$$\Leftrightarrow \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} P_{\triangle,\,\text{static}} \\ P_{\text{comp},1} \\ P_{\text{comp},2} \\ P_{\text{comp},3} \\ P_{\text{sm},1} \\ P_{\text{sm},2} \\ P_{\text{sm},3} \end{pmatrix} = \begin{pmatrix} P_{\triangle,\text{total},1} \\ P_{\triangle,\text{total},2} \\ P_{\triangle,\text{total},3} \\ P_{\triangle,\text{total},4} \\ P_{\triangle,\text{total},5} \\ P_{\triangle,\text{total},6} \\ P_{\triangle,\text{total},7} \\ P_{\triangle,\text{total},8} \\ P_{\triangle,\text{total},9} \\ P_{\triangle,\text{total},10} \\ P_{\triangle,\text{total},11} \\ P_{\triangle,\text{total},12} \\ P_{\triangle,\text{total},13} \\ P_{\triangle,\text{total},14} \\ P_{\triangle,\text{total},15} \end{pmatrix}$$

$$\Leftrightarrow \quad \begin{cases} P_{\triangle,\,\text{static}} = P_{\triangle,\text{total},1} \\ P_{\triangle,\,\text{static}} + P_{\text{comp},1} = P_{\triangle,\text{total},2} \\ P_{\triangle,\,\text{static}} + P_{\text{comp},2} = P_{\triangle,\text{total},3} \\ P_{\triangle,\,\text{static}} + P_{\text{comp},3} = P_{\triangle,\text{total},4} \\ P_{\triangle,\,\text{static}} + P_{\text{sm},1} = P_{\triangle,\text{total},5} \\ P_{\triangle,\,\text{static}} + P_{\text{sm},2} = P_{\triangle,\text{total},6} \\ P_{\triangle,\,\text{static}} + P_{\text{sm},3} = P_{\triangle,\text{total},7} \\ P_{\triangle,\,\text{static}} + P_{\text{comp},1} + P_{\text{comp},2} = P_{\triangle,\text{total},8} \\ P_{\triangle,\,\text{static}} + P_{\text{comp},1} + P_{\text{comp},2} + P_{\text{comp},3} = P_{\triangle,\text{total},9} \\ P_{\triangle,\,\text{static}} + P_{\text{sm},1} + P_{\text{sm},2} = P_{\triangle,\text{total},10} \\ P_{\triangle,\,\text{static}} + P_{\text{sm},1} + P_{\text{sm},2} + P_{\text{sm},3} = P_{\triangle,\text{total},11} \\ P_{\triangle,\,\text{static}} + P_{\text{comp},1} + P_{\text{comp},3} = P_{\triangle,\text{total},12} \\ P_{\triangle,\,\text{static}} + P_{\text{comp},2} + P_{\text{comp},3} = P_{\triangle,\text{total},13} \\ P_{\triangle,\,\text{static}} + P_{\text{sm},1} + P_{\text{sm},3} = P_{\triangle,\text{total},14} \\ P_{\triangle,\,\text{static}} + P_{\text{sm},2} + P_{\text{sm},3} = P_{\triangle,\text{total},15} \end{cases}$$

V.3.1.2 Possible additional calibration phase to extend scalability in consideration of platform dimensions

The calibration methodology presented in the previous section can be applied to any platform respecting our MoA. It allows obtaining a power model applicable to any NN deployment on the platform on which it was calibrated. When targeting platforms which respect our MoA but feature different number of tiles and private memory sizes, the methodology is still applicable, but the power model must be re-calibrated, which represents an important effort. To alleviate this, we present in this section a possible additional calibration approach to extend the scalability of the model to platforms with any tile number and private memory sizes. This extension should be considered by users willing to have a power model to jointly evaluate and optimize hardware and NN deployment. Once the additional calibration phase terminated, the power model is scalable in consideration of platform dimensions without needing re-calibrations.

This complementary calibration approach is based on our hypothesis that only the static power consumption has strong dependency with the platform's dimensions, which will be tested during the validation of model (Section V.5.3). Therefore, the methodology aims at performing a regression on a set of measurements of static power consumption for different platforms respecting the MoA to calibrate the model of $P_{\triangle/\blacktriangle,\,\text{static}}(T, \mathcal{M})$ from Equations V.6 and V.7. The following settings must be considered:

1. Different numbers of tiles: calibration of $P_{\text{tile, core}}$ by varying the parameter T of the model,
2. Different sizes of private memory of tiles: calibration of $P_{\text{tile, mem}}$ by varying the parameter $\mathcal{M}$ of the model,
3. With and without power management ($\triangle/\blacktriangle$): calibration of $P_{\blacktriangle,\,\text{components}}$.

The remaining term P_{circuit} is a constant that is independent to the parameters of the model, and is calibrated as such when performing the calibration of the other terms. The measured static power consumption data is regressed using multi-linear regression in order to calibrate the model. A minimum of 8 static consumption of different platform settings must be considered to properly calibrate the model. The 8 platforms must all be different in regards to either the number of tiles, the private memory sizes or the use of power management. We however encourage using more measurements and parameter combinations to further constraint the regression and improve the accuracy of the calibration. Once the multi-linear regression process terminated, the extended model for extension for $P_{\triangle/\blacktriangle,\,\text{static}}(T, \mathcal{M})$ is calibrated. This model can be used to predict the static power consumption of multi-core platforms of varying dimensions.

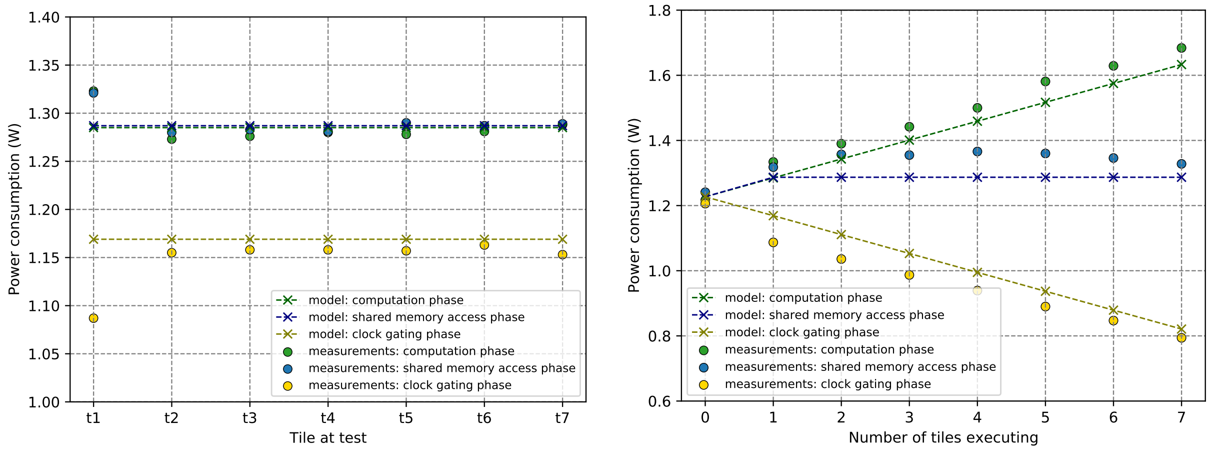
V.3.2 Application of the calibration and results

V.3.2.1 Calibrated base power model from regression of measurements

A measurement system is set up to calibrate the proposed models, using our prototype implementation platforms. As discussed in Chapter III, the power measurement infrastructure relies on probing the *VCCINT* power level of the board using the R&S HMC8012 Digital Multimeter with a sampling rate of approximately 100 samples per second. When measuring a power profile, tiles are set to execute the tested phase inside an infinite loop, and 10 000 power measurements are performed, which allows having a representative sample of the system consumption. We then verify that the standard deviation on the measured data is marginal, which ensures that the observed behavior is properly captured, and use the average value. The calibration is performed for both versions of the platform: with and without power management ($\triangle/\blacktriangle$).

Static power consumption: The multi-linear regression gives: $P_{\triangle\,\text{static}} = 1.227\,\text{W}$ and $P_{\blacktriangle\,\text{static}} = 1.260\,\text{W}$. The increase in the value on the platform with power management ($\blacktriangle$) is explained by the introduction of the interrupt controller and clock gating controllers.

Computation and clock gated phases: As shown in Figure V.3, P_{comp} and P_{cg} evolve linearly with the number of tiles being used. In this figure, it is important to note that the provided data include both static and dynamic power consumption. On plot (a), it can be seen that the tiles 2 to 7, which feature 256 kB of private memory, have a nearly identical contribution to power consumption regardless of the phase at test. The tile 1, which features a private memory of 1024 kB, has a slightly higher power consumption than the others. For example for $P_{\text{comp},1}$, the increase is 3.45 %. Because the difference of power consumption between the tiles of different private memory is so small, we decided to make the hypothesis that all tiles share the same base power consumption in computation and clock gated phases, i.e. $\forall i \in \{1, 2, ..., T\}, P_{\text{comp},i} = P_{\text{comp}}$ and $P_{\text{cg},i} = P_{\text{cg}}$. We calibrate $P_{\text{comp}} = 0.058\,\text{W}$ and $P_{\text{cg}} = 0.058\,\text{W}$. The two base power consumption are calibrated with identical value. The values obtained by regression differ only by a few tenths of mW, which is lower than the accuracy of our power measurement infrastructure. We have no particular explanation for this phenomenon, which seems to be a coincidence.



(a) Configurations II

(b) Configurations III

Figure V.3 – Extract of the measured power consumption profiles (in W) for tiles in computation, shared memory access and clock gated phases. The provided data include both static and dynamic power consumption. Graph (a) provides the profiles for tiles tested individually, which correspond to the configurations II of Equation V.19. Graph (b) provides the profiles for tiles progressively enabled all together, which corresponds to the configurations marked III. The configuration I can also be seen on the graph (b): it corresponds to the static power consumption, obtained when 0 tile is executing. The reader can refer to Equation V.19 for more information on the different tested configurations. The plots also show the proposed calibrated model.

Shared memory access phases: To properly characterize the power consumption of tiles accessing the shared memory $P_{\mathrm{sm},i}$ with $i \in \{1, 2, ..., T\}$, we followed the procedure as presented in Section V.3.1.1. However the test cases presented in this section cause the shared memory to be used 100 % of the time as long as at least one tile access it. Since the shared memory is always used at 100 %, it can lead to an incorrect calibration of P_{sm}. To alleviate this issue, we added complementary configurations, in which tiles, inside their infinite loop, are tasked to access 10 % of the time the shared memory, and then actively wait 90 % of the time. We also added configurations in which tiles are tasked to execute 100 % of the time an active wait, to properly assert the consumption of tiles in this additional activity and then correctly set the $P_{\mathrm{sm},i}$. These additional test cases allow to conveniently characterize the power consumption of tiles accessing the shared memory.

The characterization shows that, as assumed, $P_{\triangle,\,\mathrm{rwp}}(t)$ and $P_{\blacktriangle,\,\mathrm{rw}}(t)$ cannot be properly modeled as linear. As shown in Figure V.3 (b), the total power consumption of tiles accessing the shared memory stagnates with the number of tiles: it is equal to P_{sm} when one tile or more is accessing the shared memory, or 0 when none is accessing it. This allows confirming the model provided in Equations V.5 and V.4. On Figure V.3 (b) it can be noted again that the tile, which features 1024 kB of private memory instead of 256 kB like the 6 others, has a higher power consumption when accessing the shared memory, but the difference is marginal as it amounts to 2.82 %. We thus calibrate $P_{\text{sm}} = 0.060\,\text{W}$.

Conclusion regarding the characterization through measurements: The proposed methodology from Section V.3.1.1 allows calibrating the proposed power model for multi-core platforms that respects our MoA. It can then be used along our timing modeling flow as presented in Section V.4 to predict power and energy for NN deployments in consideration of the clustering and mapping on that platform. The calibration shows that the dynamic power consumption depends marginally on the tile's private memory size.

However as discussed in Section V.3.1.2, it is necessary to re-calibrate the power model when targeting another platform to keep high accuracy. We explain in the following section how we alleviate this by applying the calibration extension approach for the static power consumption model.

V.3.2.2 Calibration of the extended static power model

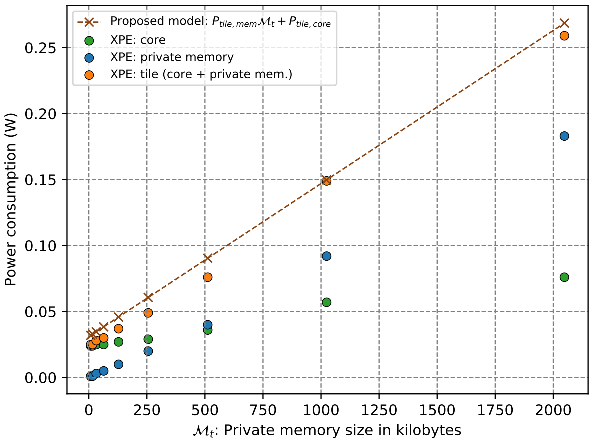


Figure V.4 – Extract of data gathered from XPE estimates showing the evolution of estimated power consumption of one tile based on its private memory size. The power consumption of the tile as estimated by XPE is depicted using orange dots. In orange dash lines with cross markers, our proposed model of one tile depending on its private memory size is provided. The power consumption of the core of the tile (MicroBlaze block) is depicted in green, and the interface between MicroBlaze and private memory (in blue).

The calibration of the extended static power model requires to gather static power consumption measurements from several platforms respecting our MoA. Due to our implementation technology, we had the possibility to alleviate this by using instead chip provider estimates. AMD/Xilinx provides indeed a tool named Xilinx Power Estimator (XPE) presented in [90]. This tool allows predicting the power consumption of a FPGA design after its synthesis and implementation phase. It is important to note that we didn't use XPE for the main calibration phase, as it isn't suitable for predicting the dynamic power consumption of multi-core platforms. For this task, it is preferable to use measurements, which accurately reflects the effects of the compiler and Instruction Set Architecture (ISA). XPE is however reliable for predicting static power consumption.

We created designs of multi-core platforms that respect our MoA with varying number of tiles and private memory sizes, as well as with and without power management, and predicted their static power consumption using this tool. We then performed a linear regression of the estimated data from XPE, to obtain a model of the static power consumption of the system based on the three identified parameters (see Equations V.8 and V.9). We provide in Figure V.4 a graph that shows a subset of the gathered data for the evolution of tile power consumption based on private memory size. This graph shows that the contribution to static power consumption of tiles evolves linearly with the private memory size. It also shows that the power consumption can be decomposed in two terms: the consumption of the MicroBlaze core and the private memory interface. This verifies our assumption regarding the use of the two terms $P_{\text{tile, mem}}$ and $P_{\text{tile, core}}$ in the model presented in Equations V.6 and V.7.

We obtain $P_{\text{tile, mem}} = 1.16 \times 10^{-3}\,\text{W}$, $P_{\text{tile, core}} = 0.031\,\text{W}$ and $P_{\blacktriangle,\,\text{components}} = 0.033\,\text{W}$ through the regression of XPE estimates. To conveniently set P_{circuit}, we calculate the difference between the measured static power consumption on our platform prototype (with parameters $T = 7$ and $\mathcal{M} = \{1024, 256, 256, 256, 256, 256, 256\}$, values provided in kB) and the model obtained from XPE estimates. Through this process we obtain $P_{\text{circuit}} = 0.678\,\text{W}$. Once the power model calibrated, it can be integrated with the timing prediction flow to provide power and energy prediction for NNs mappings onto multi-core platforms. This process is presented in the following section.

V.4 Integration in the simulation flow and energy prediction

To predict power and energy consumption, the proposed model requires knowledge regarding the phases in which tiles are throughout the execution of NNs. It also requires knowledge regarding the use of shared resources (and possible contentions). We use for this the execution traces from the simulation-based timing prediction flow presented in Chapter IV, which allows modeling the effect of shared resources for NNs deployed onto multi-core platforms. The execution traces delivered by the simulation flow contain time markers with the current phase (compute, read, write, poll or clock gated) for each tile. A new marker is generated when a tile changes phase during the execution. The execution traces from simulation allows thus knowing the state of all cores and shared resources at any time of the estimated execution. An example of traces output by the SystemC simulation and the use of the information by the proposed model to predict power can be seen in Figure V.2. The power model presented in the previous section is used as post processing of the execution traces output by the SystemC simulation. The model is used to predict the evolution of power consumption during the execution of the NN. The energy consumed during the execution of a NN on the platform is given by Equation V.21.

$$E = \frac{\int_{t_0}^{t_f} P(t)dt}{N_{\text{it}}} \tag{V.21}$$

In this equation, E is the energy in J consumed for one inference of the NN mapping on the platform. $P(t)$ is the power consumption of the system dependent on the time t. t_0 is the time at which the power acquisition begins, and t_f the time at which it ends. N_{it} denotes the number of inferences that occurs in the time frame $t_f - t_0$. If the number of inferences is sufficiently important, then $\frac{t_f - t_0}{N_{\text{it}}} \approx \mathcal{L}$ with $\mathcal{L}$ denotating the average end to end latency of the NN executed on the platform. If we use the average power consumption $\bar{P}$ over the execution of the NN on the platform, which is then constant over the duration of the NN's execution, we get Equation V.22. The mathematical relations in this equation use the mean value theorem for integrals, which serves as lemma for proving the fundamental theorem of calculus [101].

$$E = \frac{\int_{t_0}^{t_f} \bar{P} dt}{N_{\text{it}}} = \bar{P}\frac{\int_{t_0}^{t_f} dt}{N_{\text{it}}} = \bar{P}\frac{t_f - t_0}{N_{\text{it}}} = \bar{P}\mathcal{L} \tag{V.22}$$

To predict the energy consumption E with our model we thus multiply the estimated end to end latency $\mathcal{L}$ with the estimated average power consumption $\bar{P}$, obtained using the proposed power model with the execution traces from the simulation. We do the same when evaluating energy on the real experiment prototype, using the measured end-to-end latency and the power consumption for fair comparison.

V.5 Evaluation of the power modeling flow

In this section, we present and discuss the results that allow evaluating our power and energy modeling flow. In the first subsection, we present a power and energy pure analytical modeling flow, which does not model shared resources. This modeling flow is used to compare the accuracy and evaluation speed of our evaluation flow. Then, we evaluate the power modeling flow in regards to the two objectives as discussed in Section V.1:

1. Use to evaluate power and energy for NNs deployments on a user-defined multi-core platform and identify optimized mappings.
2. Use to jointly evaluate and optimize multi-core platform architectures and NN deployments under power and energy constraints.

To do so, we first validate in Section V.5.2 our flow in regards to the clustering, the number of tiles used in the mapping, the communication workload and the usage of power management on a fixed multi-core platform. In Section V.5.3, we then focus on the evaluation of the modeling flow for platforms with varying number of tiles and private memory size for tiles.

V.5.1 Analytical power and energy model for comparison

In order to compare the proposed power and energy modeling flow, we introduce a pure analytical model. It does not rely on simulation and thus do not model shared resources and contentions. In Chapter IV, Section IV.5.2, we presented a pure analytical modeling flow for timing prediction. We re-use this model and extend it for power and energy prediction. The pure analytical timing model is used to estimate the end-to-end latency and the time spent in computations, read and write transactions and waiting. These information are used to predict for each tile the average value of the power consumption during one inference of the NN with the proposed power model (Equations V.8 and V.12). Similarly to the simulation-based approach, the energy is obtained by multiplying the estimated latency with the estimated power.

V.5.2 Evaluation on a fixed multi-core platform

V.5.2.1 Tested configurations

In order to evaluate our power and energy modeling flow, we tested and stressed it in the following regards:

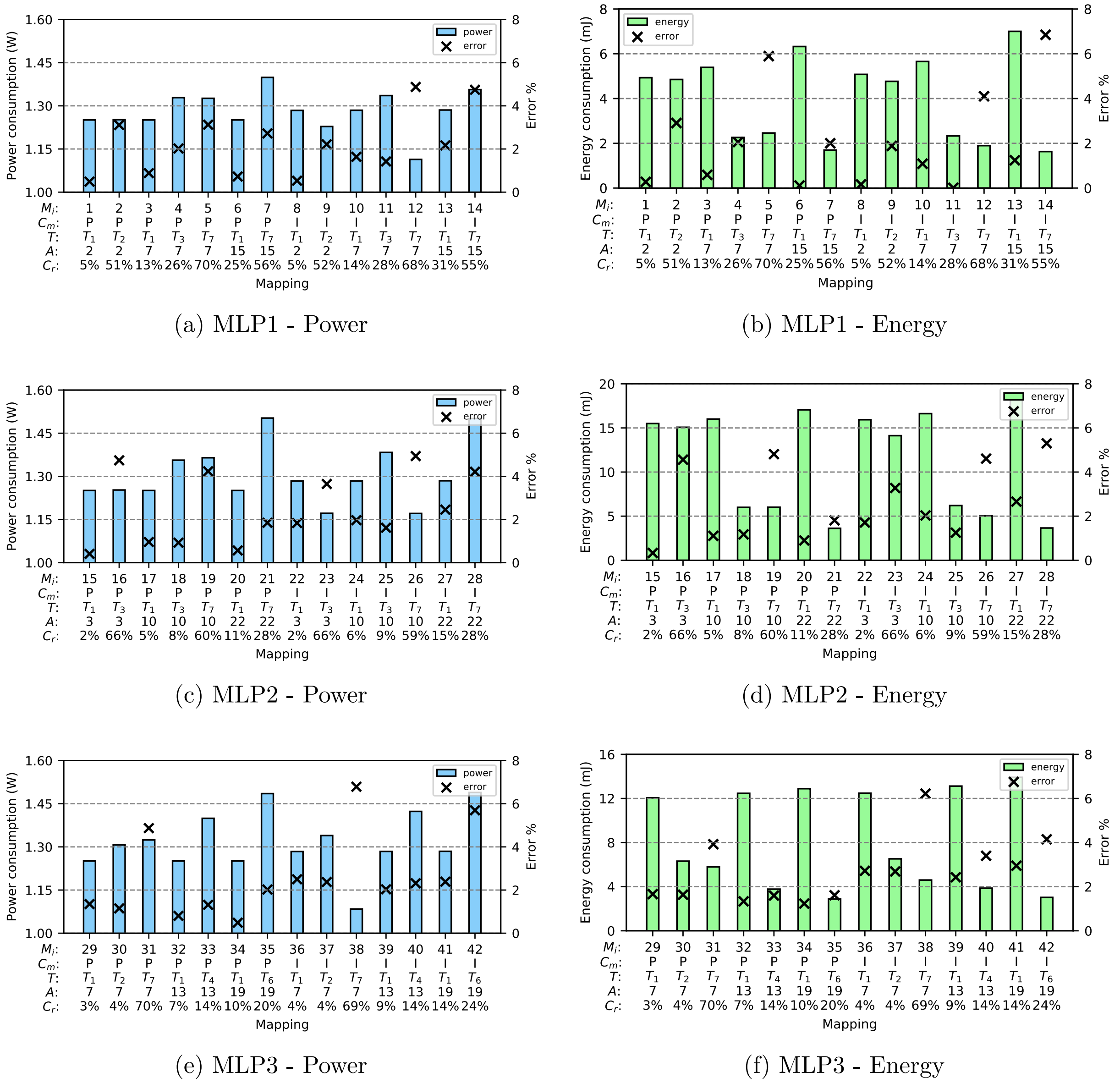
1. Consideration of applications of different complexity. We considered several NNs, with different computation and communication workloads. We used SDF graphs with a low amount of actors and communication channels, respectively 2 and 3 as lowest (MLP1 C1 as shown in Table A.1 in Appendices), as well as complex SDF graph featuring up to 22 actors and 113 communication channels (MLP2 C7 as shown in Table A.3 in Appendices).

2. Consideration of single-core mappings as well as multi-core mappings containing up to 7 tiles. The multi-core mappings leverage both NN in-layer (clustering) and inter-layer parallelism (streaming execution).

3. Consideration of mappings with high and low communication rates. The communication rate is defined as the percentage of time spent by all tiles on average in communication. The lowest observed communication rate is 2 % on one mapping and the highest is 70 % on another mapping.

4. Every mapping is tested both with and without power management.

All tested mappings are presented in details in Appendices A. In addition to these evaluation cases, we also compare the proposed power model relying on simulation against the pure analytical modeling flow from Section V.5.1. All considered mappings were tested on the main platform prototype with 7 tiles, to evaluate the applicability of the flow to search and optimize NN deployments on a user defined multi-core platform.



(a) MLP1 - Power

(b) MLP1 - Energy

(c) MLP2 - Power

(d) MLP2 - Energy

(e) MLP3 - Power

(f) MLP3 - Energy

Figure V.5 – Predicted power and energy consumption by the simulable model for the considered MLP mappings. The prediction error in absolute value is also provided for each mapping. In X-axis, information about the tested mapping is provided: the top number M_i is the mapping index, which can be used to find more information about the mapping in appendix of this thesis. The letter C_m below M_i is the communication mode - P stands for polling (without power management), while I stands for interrupt (with). The number of tiles used is indicated by the indice of T. Then respectively the number of actors A in the SDF graph and the average time spent in communication by tiles C_r (combination of read/write time and wait time) are provided.

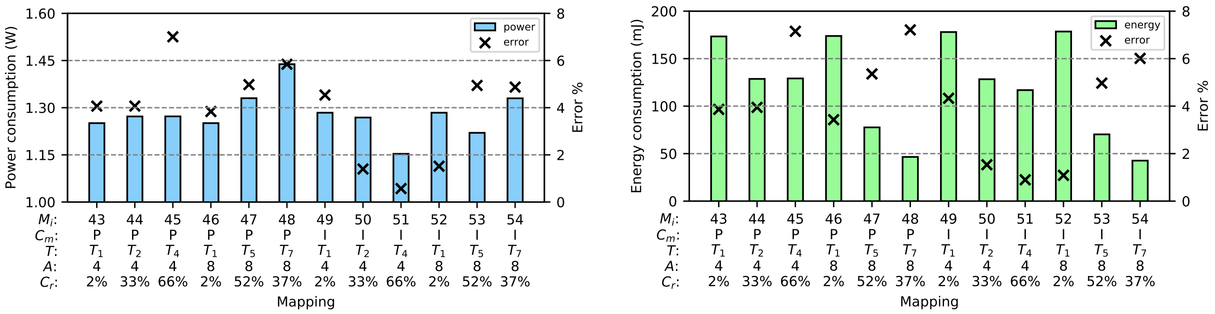
V.5.2.2 Results

We have measured the power and energy consumption of 27 different mappings, with power management and without (which amounts to 54 mappings in total), and compared the model's prediction with the measurements. The tested mappings are available in appendices of this manuscript. Table V.1 shows the average and maximum prediction error for single-core and multi-core mappings for all applications, using the simulation model and the pure analytical model. Prediction errors are computed using the formula $\frac{X_{\mathrm{P}}-X_{\mathrm{M}}}{X_{\mathrm{M}}}$ where X_{P} is the prediction and X_{M} is the measurement. The errors are provided in absolute value and percentage. In this table, the mappings with and without power management are separated to validate the model for both cases. For mappings that use less tiles than available on the platform, the unused tiles are disabled using MicroBlaze debuggers with the Xilinx Software Command-Line Tool (XSCT). When a tile is disabled, its processing core is not clocked but its private memory is. The detailed prediction of

[3] Refer to Chapter III Section III.5 for more information about the applications and mappings.

Table V.1 – Observed average and maximum error on tested mappings regarding the power consumption in W (P) and the energy consumption in mJ (E). The column titled "# tested mappings" provides the number of tested mappings. Each mapping is tested with and without power management. All details about the tested mappings can be found in appendix of this manuscript.

Application[3]	Mapping type	# tested mappings	Metric	Polling-based comm. without clock gating				Interrupt-based comm. with clock gating			
				Pure analytical model - Error %		Simulation-based flow - Error %		Pure analytical model - Error %		Simulation-based flow - Error %	
				avg	max	avg	max	avg	max	avg	max
MLP1	Single-core	3	P	0.64	0.83	1.92	2.14	1.45	2.19	1.45	2.17
			E	0.45	0.58	2.30	2.52	3.11	7.02	0.83	1.24
	Multi-core	4	P	8.42	18.59	0.53	0.72	2.06	3.88	3.32	4.88
			E	4.40	12.51	1.15	3.55	7.27	20.78	3.21	6.85
MLP2	Single-core	3	P	0.62	0.94	1.98	2.22	2.09	2.47	2.09	2.46
			E	0.92	1.28	1.84	2.30	3.78	6.91	2.13	2.66
	Multi-core	4	P	5.84	13.71	1.49	2.23	2.47	4.95	3.61	4.94
			E	5.57	9.08	1.53	2.51	6.03	13.06	3.61	5.30
MLP3	Single-core	3	P	0.85	1.34	1.74	2.14	2.31	2.50	2.31	2.50
			E	1.40	1.61	1.19	1.37	4.55	6.70	2.70	2.95
	Multi-core	4	P	5.04	16.66	1.25	2.50	2.89	6.58	4.29	6.79
			E	5.98	13.47	0.92	1.53	6.43	12.36	4.11	6.22
CNN	Single-core	2	P	3.94	4.06	1.42	1.54	3.02	4.54	3.03	4.54
			E	2.82	4.20	1.10	1.33	2.81	4.68	2.71	4.33
	Multi-core	4	P	4.69	6.92	3.12	4.59	4.13	5.71	2.95	4.94
			E	3.03	5.40	3.57	5.08	2.81	4.35	3.35	6.01



(a) CNN - Power

(b) CNN - Energy

Figure V.6 – Predicted power and energy consumption by the simulable model for the considered CNN mappings. The absolute prediction error is also provided for each mapping. More information about the legend of the plots can be found in the caption of Figure V.5

the SystemC model and prediction error for each mapping are provided in the graphs in Figures V.5 and V.6.

V.5.2.3 Discussions

Overview of the simulation-based flow evaluation results: The SystemC model has a prediction accuracy of more than 93 % compared to the measurements for both power and energy on all tested mappings. This high prediction accuracy is achieved thanks to the calibration through measurements, which allows setting the base power consumption terms with fine granularity. The power consumption linked to the different phases of tiles (NN computation, shared memory access, clock gated) are correctly modeled through the use of the execution traces from our simulation-based timing modeling flow. The overall results show the power model can be used efficiently to evaluate mappings in a DSE setup, allowing to optimize the deployment of NNs onto multi-core platforms.

It can be noted that the prediction accuracy on power and energy is slightly lower than the prediction accuracy of the timing modeling flow as seen in Chapter IV (more than 97 % accurate on all tested mappings). This is due to the following:

1. The power measurement infrastructure offers less accurate measurements than the timing measurement infrastructure. This leads to a more important error on the elementary power consumption terms (e.g. P_{comp}) used for the calibration of the model but also on the measurements used to validate the model.
2. Power predictions are performed using the execution traces issued from the timing

modeling flow, which means the timing error is also propagated on power predictions. The error on energy prediction is also slightly higher than the error on power. This is due to the propagation of the timing error a second time when predicting energy, as it is computed as the integration of the average power consumption of the system over the end-to-end latency.

Fast evaluation speed: The time needed to evaluate a mapping using our SystemC models (lower part of Figure V.1) is approximately 20 s, which is mostly spent in the compilation of the SystemC model. The power model is coded as a Python script. It processes the execution traces produced by the SystemC model after its execution. The execution time of the Python script describing the power model is negligible compared to the simulable model compilation and execution.

When probing power consumption using our power measurement infrastructure, a total of 10 000 samples are measured at a sampling rate of 100 sample/s, which corresponds to an evaluation time of 100 s, which is 5 times more than our evaluation flow. It can also be noted that this duration excludes the effort needed to synthetize the FPGA design, build the Board Support Package (BSP) necessary to execute software on the tiles, train the NN, develop and compile the code to infer the NN on the platform as well as programming the FPGA and the tiles with the compiled code. The proposed power modeling flow thus allows saving important evaluation time, which is important to allow a fast and thorough evaluation of the design space to optimize NN deployments on multi-core platforms.

Applicability to MLPs and CNNs: The average prediction error on power and energy for all considered mappings of MLPs combined is respectively 2.20 % and 2.17 %. The average prediction error for the CNN is 2.76 % on power and 2.94 % on energy. The prediction error on the CNN is in the same order as the prediction error on the MLPs, despite being slightly higher. It is important to note that these results are obtained with the same value of P_{comp} for all layer types, despite the difference of computations inside layers. It could be arguably possible to improve the accuracy of the power model by calibrating different values of P_{comp} for every layer. It must be noted that this represents a more important calibration effort and the model's accuracy is already excellent without.

The error of the model increases with the computation workload contained in the application. The average prediction error is the lowest for the MLP1, which is the application with the lowest amount of computations that we considered. On the MLP1, the model averages a prediction error of 1.82 % on power and 1.92 % on energy. On the MLP3, which is the MLP with the highest computation workload, the average prediction error is 2.45 % on power and 2.27 % error on energy. The increase in error is however minimal between the two applications, as it is less than 1 %.

These results validate the applicability of the proposed modeling flow for applications with various computation workloads and to both MLPs and CNNs.

Mappings with and without power management: The average prediction error of the SystemC modeling flow when applied to the 27 polling-based mappings (i.e. without power management) is 2.11 % on power and 2.22 % on energy. When applied to the 27 interrupt-based mappings with clock gating (i.e. with power management), the average prediction error is 3.92 % on power and 3.91 % on energy. Similarly we observe that the maximum error on power is 4.59 % and on energy is 5.08 % for polling-based communications while the maximum error on power is 6.79 % and on energy 6.85 % for the interrupt-based communications. Overall the prediction error of the model is slightly higher on interrupt-based communications than on the polling-based communications.

This higher prediction error is due to the choice to consider that during waiting phases, the cores are always clock gated in order to reduce the complexity of the model (see Equation V.10). In fact, the interrupt-based platform relies on a single interrupt signal for all tiles. When the interrupt signal is enabled, all tiles are switched on and check the availability of tokens in the shared memory. Tiles for which the token is unavailable are switched back to clock gating right after. However, this switching of modes is not captured in the proposed power modeling flow. In addition, when tiles enter the waiting phase on the real platform, they also have short delays at the beginning of the phase when they check the availability of tokens in the shared memory and switch on their sensitivity to interrupt signals. For these two reasons, the prediction error on power and energy for interrupt-based communications with the use of clock gating is slightly higher than the error for polling-based communications.

Even though the prediction error for interrupt-based communications is higher, it remains acceptable as it is overall less than 2 % higher than the polling counterpart and remains under 7 %, which is relatively low for a modeling flow with a high level of abstraction. This validates the proposed power model for both polling-based communications without clock gating and interrupt-based with clock gating.

Clustering: In this paragraph, we discuss the applicability of the proposed modeling flow to different clusterings. As shown in the tables in Appendices, we tested mappings with low feasible clustering (2 actors and 3 channels) and others with high possible clustering (22 actors and 113 channels). For the *MLP1* application (Figure V.5 (a)) for mappings with mapping index $M_i = (8, 10, 13)$, the clustering is increased progressively (they have respectively $A = 2, 7$ and 15 actors) and the error on power prediction also increases with the clustering up to approximately 2 %. However on the *MLP3* application (Figure V.5 (e)) for mappings $M_i = (29, 32, 34)$, the opposite is observed as the error decreases with the clustering (respectively $A = 7, 13$ and 19 actors). Other obtained results also show that there is no correlation between clustering and prediction error (see the results obtained on the *MLP2* for example). The same observations can also be done for the energy prediction. This allows validating that the clustering of the NN in SDF does not impact the prediction accuracy using the proposed power modeling flow.

Number of tiles and communication workload: Table V.2 provides a summary of the average and maximum prediction error on power consumption of the simulation-based power modeling flow depending on (a): the number of cores used in the mapping and (b): the communication rate. Overall, as it can be observed from Table V.2 (a), the prediction error increases with the number of cores. It can be noted that for single-core mappings the average error of the model is 1.73 %, for mappings using 2 up to 4 tiles it is approximately 2.50 % and for mappings using 5 up to 7 tiles it is approximately 4.42 %. The same observation can be drawned from Table V.2 (b) with the communication workload: the prediction error of the proposed modeling flow increases with the amount of communications in the mapping. The number of tiles used and the communication workload are closely linked, as increasing the amount of cores tend to increase the amount of shared resource contentions, which effect timing and power consumption. The increase

Table V.2 – Summary of the average and maximum prediction error on power consumption of the simulation-based power modeling flow based on (a): the number of cores used in the mapping and (b): the communication rate.

(a) Core number

Core number	1	2	3	4	5	6	7
Average error %	1.73	2.39	2.40	2.80	4.96	3.86	4.42
Max error %	4.54	4.07	4.74	7.01	4.98	5.70	6.79
Total mapping number	22	6	6	4	2	2	12

(b) Communication rate

Comm. rate	0-9%	10-19%	20-29%	30-39%	40-59%	+60%
Average error %	1.84	1.52	2.56	3.67	3.79	4.48
Max error %	4.54	2.46	5.70	5.84	4.98	7.01
Total mapping number	17	9	7	5	6	10

of shared resource contentions leads to the observed increase of prediction error. However it can be noted that despite the error increase, the model's predictions remain acceptable for up to 7 tiles and communication rates up to 70 % as the maximum observed error is 7.01 %. In this paragraph we focused on power consumption but similar observations can also be drawn for the energy prediction. This validates the applicability of the modeling flow for up to 7 tiles and communication workload up to 70 %.

Comparison of the pure analytical model and the simulation: The pure analytical model does not offer accurate predictions on all considered mappings. The error is especially high on multi-core mappings, as this model averages 6.46 % error on power consumption with a maximum error of 18.59 % and 7.55 % error on energy with a maximum of 20.78 %. These errors are so high because the pure analytical model always underestimates the effect of shared resource on the power and energy consumption. These results show the importance of modeling the influence of shared resource for power and energy prediction, and thus the importance of using a simulation-based approach. The predictions of the pure analytical flow on single-core mappings are acceptable due to the absence of shared resource influence in these scenarios. The pure analytical model has an evaluation time in the order of a millisecond, which is 10^4 an order of magnitude faster than the simulation-based approach. Due to its fast evaluation time, the pure analytical modeling flow can be used, despite its low accuracy, in conjunction with the simulation flow to offer a fast pruning of the design space, before the evaluation of relevant mappings by the SystemC simulation flow, which offers slower but more reliable predictions.

V.5.3 Evaluation of the scalability in regards to the number of tiles and private memory size

V.5.3.1 Tested configurations

In this section, we evaluate the usage of the modeling flow to jointly optimize multi-core platform's dimensions and NN deployment. To this end we evaluate the applicability of our modeling flow to platforms with different number of tiles and private memory sizes. We considered a total of four multi-core platforms and three single-core platforms, which are presented in Table V.3. These platforms allow testing the following:

- P_7 (which is the platform used in the previous section) and P_5 allows testing platform with a high number of tiles (respectively 7 and 5) and unequal distributions of private memory size among tiles,

— P_2 and P_3 allow testing several mappings with platforms with identical and low private memory size (64 kB) and reduced amount of tiles (respectively 2 and 3),

— Single-core platforms (variations of P_1) allow testing the applicability of the modeling flow when applied to single-core platforms and private memory sizes ranging from 512 kB up to 2048 kB.

Table V.3 – Dimensions of the different considered platforms. The value inside the table indicate the size in kilobits of the private memory of tile. The symbol / means that this tile is not used.

Platform	Tile number						
	T_0	T_1	T_2	T_3	T_4	T_5	T_6
P_7	1024	256	256	256	256	256	256
P_5	1024	512	512	256	256	/	/
P_3	64	64	64	/	/	/	/
P_2	64	64	/	/	/	/	/
$P_{1,2048kB}$	2048	/	/	/	/	/	/
$P_{1,1024kB}$	1024	/	/	/	/	/	/
$P_{1,512kB}$	512	/	/	/	/	/	/

It must be noted that for the P_7 platform, we implemented a version with power management (denoted $P_{7,I}$, I standing for Interrupt) and a version without (denoted $P_{7,P}$, P standing for Polling). For all other considered platforms, we implemented the interrupt controller and clock gating controllers. Doing so allows testing mappings both with and without power management. The P_7 platform, due to its memory setting, is the only one that enables the execution of the 27 considered mappings. We thus tested only a subset of mappings on each considered platforms. For the single-core platforms, we tested two sub-versions of each platform, issued from two different FPGA implementation strategies using the Xilinx tools. We used the default implementation strategy, as well as a strategy focused on power optimization when conducting placement, routing and bitstream generation steps of the FPGA design. We choose to provide a focus on static consumption in addition to dynamic power consumption in our validation approach to evaluate the hypothesis we formulated that only the static power consumption depends on the tile number and private memory size implemented in the platform.

V.5.3.2 Results

To evaluate the scalability of the proposed modeling flow to platforms with various number of tiles and private memory sizes, we provide:

- in Figure V.7 the comparison of predicted and measured static power consumption for the multi-core platforms that we considered.
- in Figure V.9 the comparison of the predicted and measured dynamic power consumption for the multi-core platform.
- in Figure V.8 the comparison of the predicted and measured static power consumption for the single-core platforms.
- in Figure V.10 the comparison of the predicted and measured dynamic power consumption for the multi-core platforms.

V.5.3.3 Discussions

Static consumption: For all considered platforms, the error on static consumption remains under 5 % regardless of the number of tiles. The model tend to underestimate

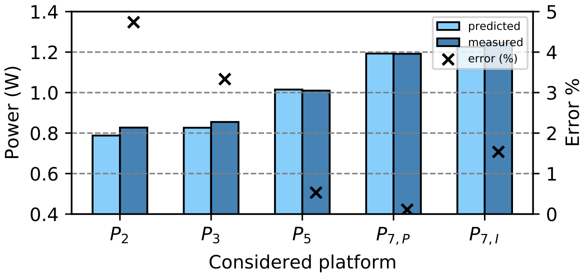


Figure V.7 – Predicted and measured static power consumption for the considered multi-core platforms.

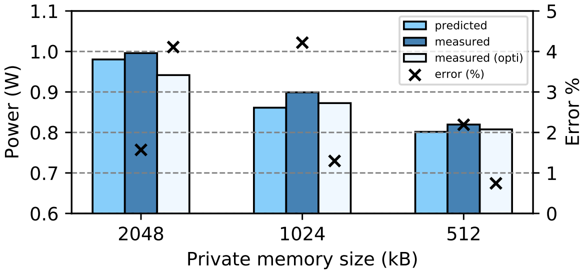


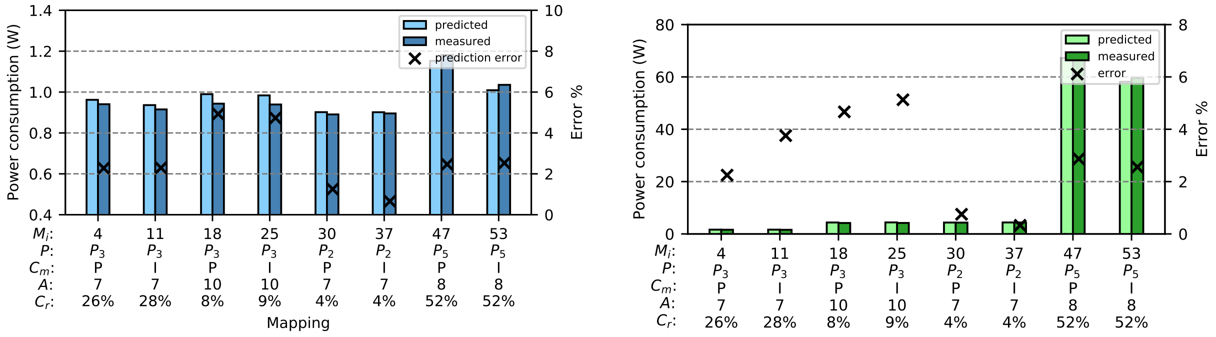
Figure V.8 – Predicted and measured static power consumption for the considered single-core platforms.

slightly the static power consumption of multi-core platforms with low amount of tiles and low private memory size as shown in Figure V.7. The error is higher on P_2 (almost 5 %) and P_3 than on the other platforms with higher tile numbers. However the error decreases from 2 tiles (P_2) to 1 tile (see Figure V.8), which does not allow establishing a correlation between the number of tiles and high error. For single-core platforms, the error is overall lower on the version with 512 kB of private memory than the two others with 1024 kB and 2048 kB. This can be explained by the fact that the place and route is more constrained with bigger memory sizes, generating overheads in power consumption [4]. The prediction error remains acceptable. We conclude that the modeling flow provides acceptable predictions of static power consumption in regards to the number of tiles and private memory size.

Dynamic consumption: As shown in Figure V.9, on multi-core mappings the prediction error seems to be not influenced by the number of cores or the private memory sizes. The highest error is indeed observed on one mapping of P_3, but on another scenario the error is lower, so it is not directly linked to core number. The same observation can be done for energy prediction, the highest errors being observed on P_3 up to more than 5 %, with very low error on P_2 and an error under 3 % for P_5.

We however observe clear limitations on single-core platforms, as shown in Figure V.10. The prediction error rises over 10 % for two mappings on the version of the platform with

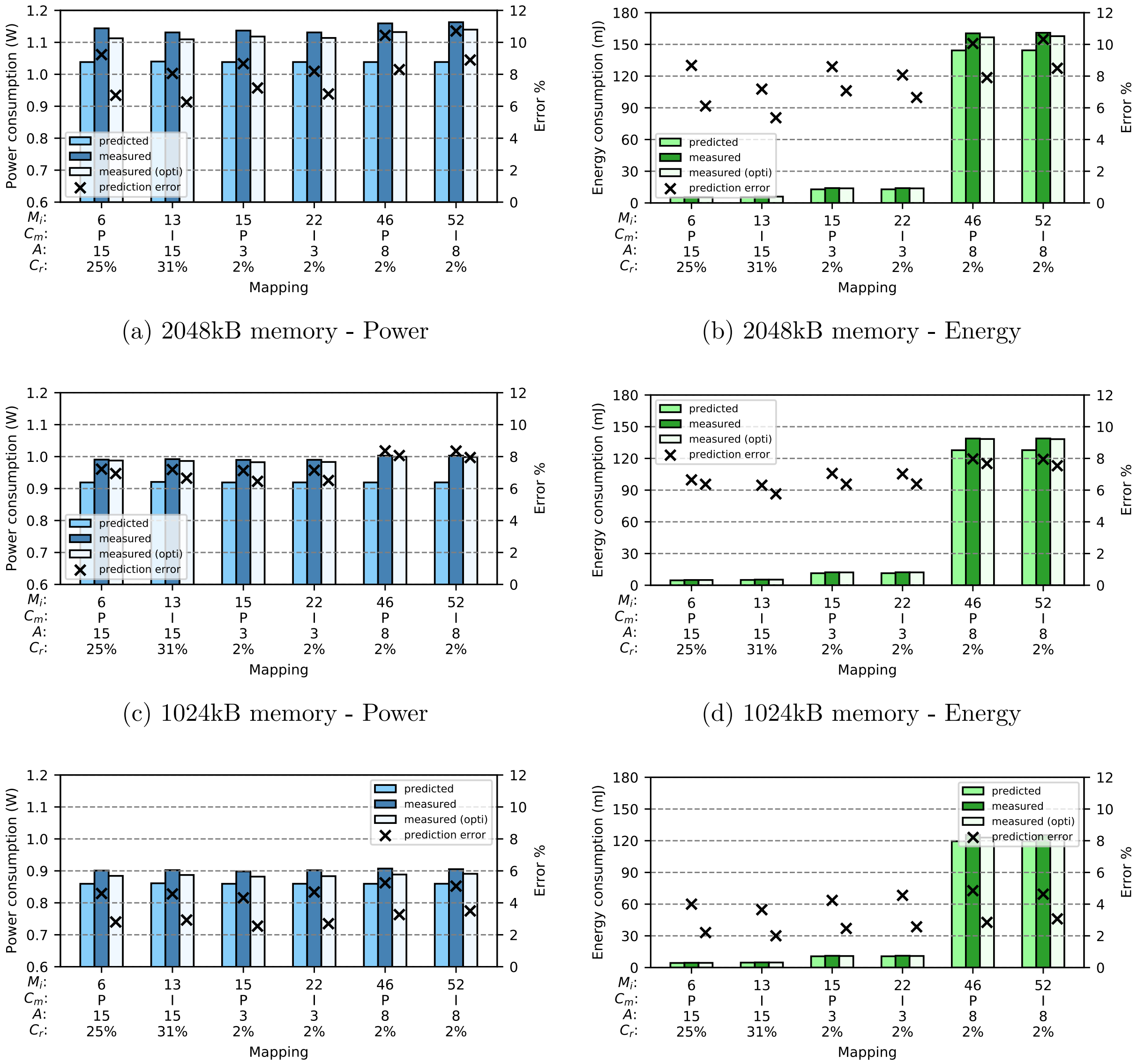
[4] The interested reader can find the place and route schematics of the different platforms in Appendices B.



(a) Power

(b) Energy

Figure V.9 – Predicted and measured system power (including static and dynamic) and energy consumption for the considered mappings on multi-core platforms.



(a) 2048kB memory - Power

(b) 2048kB memory - Energy

(c) 1024kB memory - Power

(d) 1024kB memory - Energy

(e) 512kB memory - Power

(f) 512kB memory - Energy

Figure V.10 – Predicted and measured system power (including static and dynamic) and energy consumption for the considered mappings on single-core platforms.

2048 kB of private memory. The prediction error decreases with the private memory size, as it approximates 7 % for the version of the platform with 1024 kB and 4 % for the version with 512 kB. The prediction error is slightly reduced with the versions of the platforms with optimized power consumption, with a maximum decrease of 1 %. Since the decrease in error is low, it eliminates the possibility that the observed prediction error would be due to the place and route strategy. The high prediction error shows that, for single-core platforms, the hypothesis that the dynamic power consumption of tiles is independent of tile memory is false. The error remains acceptable for memory sizes up to 512 kB for single-core platforms (as shown in Figure V.10 (e) and (f)) and for multi-core platforms containing one tile with 1024 kB as shown by the results on platforms P_5 and P_7.

Conclusion: Static power consumption is well predicted by the model in consideration of the tile number and private memory size in the tested platform. Dynamic power consumption is also well predicted for multi-core platforms. However, the model shows limitations to predict the dynamic contribution in the case of single-core platforms with important private memory (1024 kB, 2048 kB). For such platforms, the hypothesis that tile's dynamic power consumption is independent of their private memory size is incorrect. Despite this, the proposed modeling flow offers acceptable results for memory up to 512 kB on single-core platforms and has shown to provide good prediction accuracy on multi-core platforms featuring one tile with 1024 kB.

V.6 Conclusion

In this chapter we proposed a power and energy modeling flow for NNs mapped onto tile-based multi-core platforms. This modeling flow relies on calibration through power measurements, which allows offering accurate power predictions. It is then combined with estimates from the chip provider to offer scalability in regards to the platform's size. The flow relies on the execution traces generated by our simulation-based timing modeling flow to obtain the information about tile phases (compute, shared memory access, clock gated) and shared resource states in order to offer accurate power and energy predictions. The proposed flow offers more than 93 % accuracy on 27 mappings of NNs with and without power management, and with various clusterings, tile usage and communication rates. Due to its high accuracy, the modeling flow can be confidently used to identify when using power management is relevant to save power and energy. It offers acceptable predictions when applied to 7 platforms implementing 1 up to 7 tiles, and private memory sizes from 64 kB up to 2048 kB. The evaluation time using our modeling flow is approx. 20 s per mapping, which represents an important time saving compared to rapid prototyping. Due to its high prediction accuracy, evaluation speed and scalability, our timing and energy modeling flows can be used to efficiently explore the design space of NNs deployment onto multi-core platforms. They can be used both to find optimized deployments of NNs onto a user defined multi-core platform, and to design a multi-core platform for optimized NN inference. We identified two main limitations of our flow:

1. We note an influence of the communication rate on the prediction error when it becomes important. It reaches a maximum of 7 % error for a mapping featuring 70 % communication rate, which is still relatively low.
2. On single-core platforms with important private memory (1024 kB and 2048 kB), the modeling flow has high prediction error (> 10 %). This shows that the hypothesis that the influence of the private memory size on dynamic power consumption can be considered marginal, is invalid in this situation. On the other tested platforms including a single-core platform with 512 kB of private memory, as well as multi-core platforms with 2 up to 7 tiles, the prediction accuracy is acceptable.

In the next chapter, we will see how the proposed modeling flow can be used to perform a fast and efficient exploration of the design space under timing and power/energy constraints.

Chapter VI

Design space exploration using the proposed timing and energy models

In this chapter, we present a Design Space Exploration (DSE) flow to search and optimize NN mappings onto multi-core platforms regarding timing and energy. The flow can be used in two ways:

(1) Find optimized NN clustering/mapping onto a fixed platform specified by the user,

(2) Jointly optimize hardware in regards to the number of cores and private memory sizes, and software (NN clustering/mapping).

The objective of this chapter is to provide a demonstration of how the timing and energy modeling flow from Chapters IV and V can be used in a DSE setup. Through the use of analytical models, which provide very fast evaluation, the large design space is pruned to select the most promising clusterings and mappings. The selected mappings are then passed to the simulation-based timing and energy modeling flow as presented in Chapters IV and V for a rapid but more precise evaluation. The list of mappings ranked by their predicted timing and energy properties is returned to the user. The proposed flow offers automatized generation of source code in C programming language for real implementation of user's selected mappings. In the experiment, we applied our DSE flow to 5 different NNs and discuss the results.

VI.1 Proposed DSE flow overview

Finding highly optimized NN deployments into multi-core edge systems is difficult but necessary. NNs are computation and memory intensive, while multi-core edge systems have resource limitations and strong timing and power constraints to satisfy. A fast yet confident exploration of NN mappings onto edge multi-core platforms is crucial to quickly identify deployments that optimize resource usage, performance and energy. Efficient DSE for such applications is however tedious. On the one hand, the design space is vast due to the numerous NN clustering/mapping and HW dimension possibilities that must be evaluated to find optimized solutions. On the other hand, highly scalable models are required to confidently evaluate and select optimized solutions from the large design space. Possible solutions can have different computation/communication workloads, NN-parallelism usage, communication procedure, use of power management and hardware specifications. An alternative to using models is to use rapid prototyping through systematic implementation and benchmarking of possible solutions, but this requires a huge effort and limits the possibility in regards to hardware exploration.

In Chapters IV and V, we proposed and validated a timing and energy modeling flow for fast yet accurate evaluation of NN mappings onto multi-core platforms. This modeling flow allows addressing the aforementioned challenges. However, while the proposed modeling flow offers a fast evaluation time of approximately 20 s per mapping (observed for 54 mappings), this evaluation speed proves to be insufficient regarding the large design space. To illustrate this with an example, for the deployment of the CNN1 featuring 4 layers on platforms with a maximum number of cores $T_{\max} = 3$, there is a total of 27 different clusterings, and a total of 109 332 mappings [1]. Using the proposed modeling flow, it would take more than 25 days to evaluate all mappings, which is very long. This example focuses on the simple CNN1 with 4 layers and a reduced amount of cores. The size of the design space raises tremendously with the number of cores and amount of layers in the NN.

Proposed DSE flow overview: We propose the DSE flow as shown in Figure VI.1. Our approach is based on two main stages, which use two different types of models:

1. The first stage is the DSE stage, where we use purely analytical models to perform a fast exploration of the design space, and identify the most promising mappings (❶ and ❷). This search is divided into two phases: first performing an exploration of clusterings, then an exploration of mappings that can be generated from that clustering. To carry out these phases, we use the pure analytical models introduced in Chapters IV and V. Their low accuracy is minor here over their ability to perform a very fast exploration (approximately 1 ms) of the large design space. A selection of optimized mappings is returned at the end of this stage.

[1] These numbers are obtained when performing exhaustive search as shown in Section VI.4

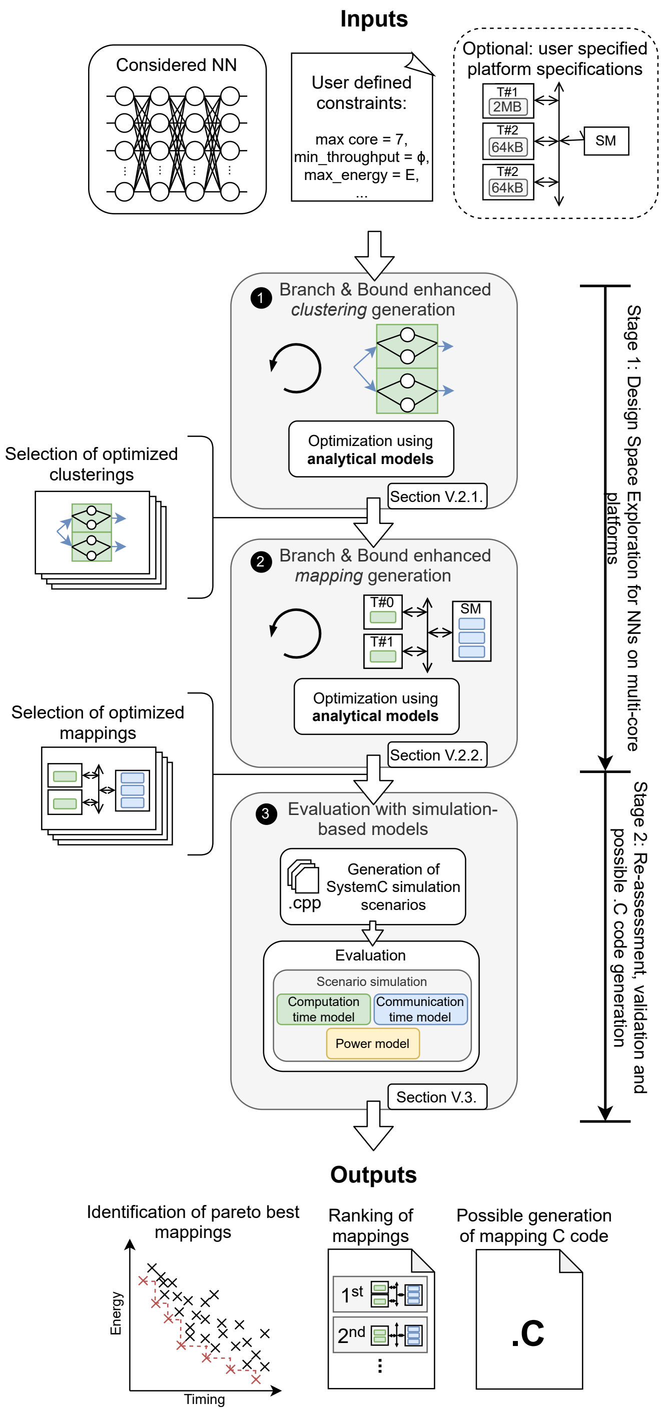

Inputs
Considered NN
User defined constraints:
max core = 7,
min_throughput = ϕ,
max_energy = E,
...
Optional: user specified platform specifications
T#1
2MB
T#2
64kB
T#2
64kB
SM
1
Branch & Bound enhanced *clustering* generation
Optimization using **analytical models**
Section V.2.1.
Selection of optimized clusterings
2
Branch & Bound enhanced *mapping* generation
T#0
T#1
SM
Optimization using **analytical models**
Section V.2.2.
Selection of optimized mappings
3
Evaluation with simulation-based models
.cpp
Generation of SystemC simulation scenarios
Evaluation
Scenario simulation
Computation time model
Communication time model
Power model
Section V.3.
Stage 1: Design Space Exploration for NNs on multi-core platforms
Stage 2: Re-assessment, validation and possible .C code generation
Outputs
Identification of pareto best mappings
Energy
Timing
Ranking of mappings
1st
2nd
⋮
Possible generation of mapping C code
.C

Figure VI.1 – Proposed DSE flow, which is organized in two main phases: first the design space is pruned using analytical models, and then selected mappings are evaluated using the simulation-based flow.

2. The second stage serves as a validation and refinement of the estimates obtained in the first stage, by using the slower but highly confident simulation-based models ❸. To this end we use the simulation-based flow, proposed in previous chapters. After their re-assessment, the mappings are ranked and are returned to the user. The flow supports the generation of the C source files needed for the execution of the mapping on the targeted platform.

Inputs of the flow:

1. The first input of our flow is the considered NN. It is important to note that the input NN does not need to be trained in order to perform the DSE (it is however, necessary, if the user wants to generate the .C code at the end of stage 2). Our flow relies on parameters of the NN's application: the number of layers, type of layers, input and output size of each layer and number of features [2]. This is rendered possible by the proposed modeling flow and use of the SDF MoC, which allow predicting the timing and energy cost of computations and communications of NNs with only these parameters.
2. The second input of our modeling flow are the user defined constraints regarding the timing and power properties. The user can specify maximum/minimum threshold for evaluated quantities (e.g. the energy must not exceed a threshold value). The user must also define T_{max}: the maximum number of tiles implemented in the platform. In our work, we do not consider constraints regarding functional properties such as the NN's classification/regression accuracy.
3. The third and last input is optional: it corresponds to the platform specifications in the case the user wants to find the best mapping of a given NN for a fixed platform. This input fixes the number of tiles inside the platform, size of private memory of tiles and its support for power management.

Stage 1 - Step ❶ - Branch & Bound enhanced *clustering* exploration: The step ❶ of the proposed DSE flow aims at exploring the clustering of NNs. As presented in Chapter III, the notion of *clustering* refers to the way NNs are described in the SDF MoC, and more specifically to the number of actors generated per layer. It corresponds to the intra-layer parallelism expression. Coarsed-grained clusterings correspond to SDF graphs in

[2] The number of features correspond to the number of neurons for dense layers and number of convolution filters for convolution layers.

which NN layers are split into limited amount of actors, and consequently limited amount of channels between actors. On the opposite, fine-grained clusterings correspond to SDF graphs in which the layers of the NN are split into important amount of actors, and thus important amount of channels between actors. One must find a good compromise between intra-layer parallelism expression, which is necessary to accelerate the NN's processing, and communication overheads caused by the additional channels. In our case, we use the Branch & Bound algorithm to optimize the clustering search. While the design space of clusterings itself is not massive, numerous mappings can be generated for every clustering. Selecting clusterings is therefore needed to reduce the number of mappings to evaluate.

Stage 1 - Step ❷ - Branch & Bound enhanced *mapping* exploration: For every considered clustering, a tremendous amount of mappings is possible. Despite the pruning of lesser clusterings performed in Step ❶, it is also necessary to prune the mapping design space using an optimization algorithm. For each selected clustering from ❶, ❷ uses Branch & Bound to select mappings in consideration to user preferences and prune the design space out of less optimized ones. Step ❷ is presented in Section VI.2.2.

In Step ❷, when jointly optimizing hardware and software implementation, correctly sizing tiles private memory is necessary due to their non-negligible influence on energy properties. We thus introduced a memory size model to predict the size of private memory (instructions and data) of tiles in regards to the requirements of mappings. This model is presented in Appendices C. It was built and validated by analyzing the memory needs of tiles for NN execution. This model is proposed for the MicroBlaze, but could be applied to any processing core with minimal porting effort.

Stage 2 - Step ❸ - Evaluation with simulation-based models After the DSE in stage 1, selected mappings are passed to the simulation-based modeling flow to perform highly accurate evaluation of the tested properties. The models used in this step and their contribution were presented in Chapters IV and V. Once the evaluation performed, mappings that do not satisfy the constraints specified by the user are eliminated. The remaining mappings are ranked and returned to the user.

VI.2 DSE using high level pure analytical models

VI.2.1 Proposed clustering optimization approach

As discussed in Section VI.1, an optimized clustering exploration is necessary. It allows reducing the number of considered clusterings with consequence to greatly reduce the number of mappings to evaluate, as well as guaranteeing that the most relevant clusterings are still considered. First we consider that the maximum number of clusters that can be generated for every layer is $T_{\max}$. This assumption is well justified because having a number of clusters higher than the number of cores mean that they cannot be executed simultaneously. For this reason, the execution does not benefit from the additional intra-layer parallelism expression of the higher clustering, but is hindered by the additional communication channels that are implemented. Despite this first assumption, the number of clusterings can be further decreased. For example with $T_{\max} = 7$, the CNN1 has 245 clustering possibilities with each of them leading to numerous mappings. Reducing the number of clusterings by pruning less optimized ones through the use of Branch & Bound is thus imperative in order to save important exploration time and effort. Step ❶ is presented in Section VI.2.1.

To optimize the clustering search process in this work, we use Branch & Bound. The way we implemented this algorithm to enhance the clustering search is illustrated in Figure VI.2. To allow comparing clusterings and performing a selection, we introduce the notion of *clustering score*. We choose for the clustering score to consider only a timing quantity, as energy properties are highly dependent on the mapping and platform specifications. Energy properties will be instead considered and optimized inside the mapping exploration flow. The clustering score is defined as the sum of the computation time of the actor with the highest number of neurons (or kernel computation for convolution layers) inside the clustering with the total communication time of channels read/write inside the clustering. To account for the amount of intra-layer paralleism expressed from the NN, only one actor is considered by layer. If all actors were considered, then all clusterings would share the same computation time, as they all share the same total computation workload: the computation workload of the considered NN. The communication time of clusterings depends on the number of channels to implement, which is linked to the number of actors, and the size of data. The predictions of those delays are done using analytical models introduced in Chapter IV: the models for computation time of NN layers from Section IV.2.1 and the model for token production/reading in shared memory model

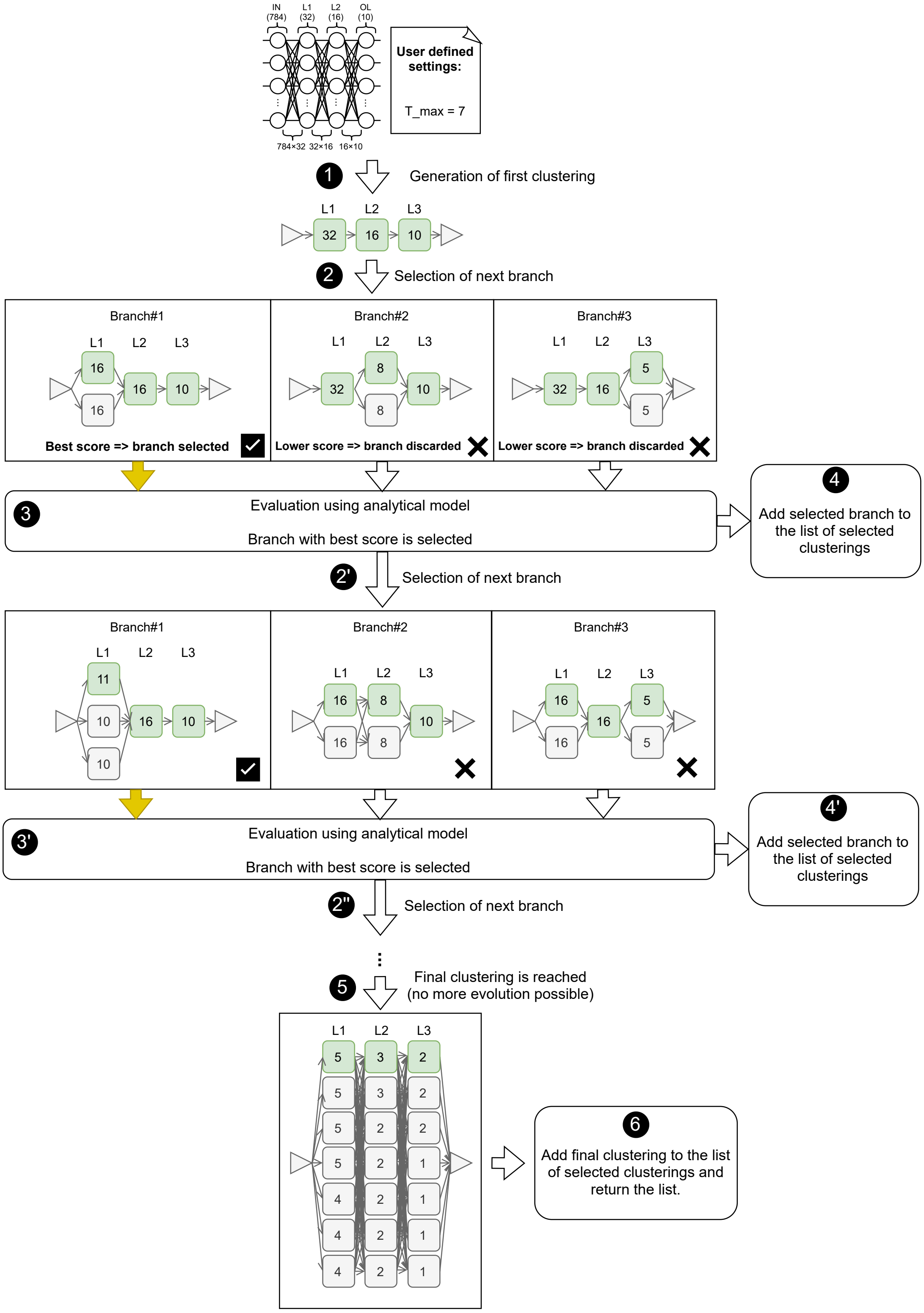

IN (784)
L1 (32)
L2 (16)
OL (10)
784×32 32×16 16×10
User defined settings:
T_max = 7
1
Generation of first clustering
L1 L2 L3
32 16 10
2
Selection of next branch
Branch#1
Best score => branch selected
Branch#2
Lower score => branch discarded
Branch#3
Lower score => branch discarded
3
Evaluation using analytical model
Branch with best score is selected
4
Add selected branch to the list of selected clusterings
2'
Selection of next branch
3'
Evaluation using analytical model
Branch with best score is selected
4'
Add selected branch to the list of selected clusterings
2"
Selection of next branch
5
Final clustering is reached (no more evolution possible)
6
Add final clustering to the list of selected clusterings and return the list.

Figure VI.2 – Illustration of the clustering exploration process on the MLP2. The number inside actors is the number of neurons they contain.

presented in Equation IV.4. The algorithm is presented in Figure VI.2:

- Inside the main procedure, the first clustering is generated, in which every layer is composed of only 1 actor. On Figure VI.2 this corresponds to ❶.
- Until the last clustering (❹) is reached, the next possible branches are generated and evaluated ❷ and ❸. The one with the highest *clustering score* is selected.
    - For every layer in the NN, if the layer can be clustered (i.e. if it is a convolution or dense layer) and if the current number of clusters for this layer is less than the maximum number of clusters, then a new branch is created, in which the number of clusters for this layer is increased by 1. This can be observed for all branches on the figure.
    - If no new branch was found, this means that the last clustering has been reached ❹. If branches were found, the selected clustering is stored in the list of all selected clusterings, and a new iteration is executed.
- When the last clustering is found (❺), the list of all selected clusterings is returned.

Several clusterings are discarded at each branch. The branches are stored, which enables resuming the execution of the Branch & Bound clustering search to consider additional clusterings from the remaining ones. The next section explains how an optimized mapping search is conducted from the selection of clusterings. In Section VI.4, we will evaluate the contribution of the Branch & Bound algorithm for clustering exploration.

VI.2.2 Proposed mapping optimization approach

Despite the reduction of clusterings from Section VI.2.1, the design space remains large as numerous mappings are generated for each clustering. Evaluating every one of them using the simulation-based flow is time-intensive. A major proportion can be discarded as they offer less optimized timing and power properties. We thus propose in this section a methodology to allow an early identification and pruning of lesser mappings. As a reminder from Chapter III mappings are self scheduled for platforms using our MoA, and since NNs are directed graphs, by always mapping in ascending order, we avoid deadlocks.

To enable the exploration, we must introduce a formulation to encode mappings. Figure VI.3 provides an example of the formulation we use for a mapping of the CNN1. On this figure, one clustering of the CNN1 is considered. Possible mappings of layers for this clustering are described as tables denoted $M[l]$, in which $l \in \{1, 2, ..., L\}$ denotes the layer number. In these tables, the indexes correspond to the cluster number $c \in \{1, 2, ..., C\}$ and the values correspond to the tile number $t \in \{1, 2, ..., T\}$ on which the cluster is mapped.

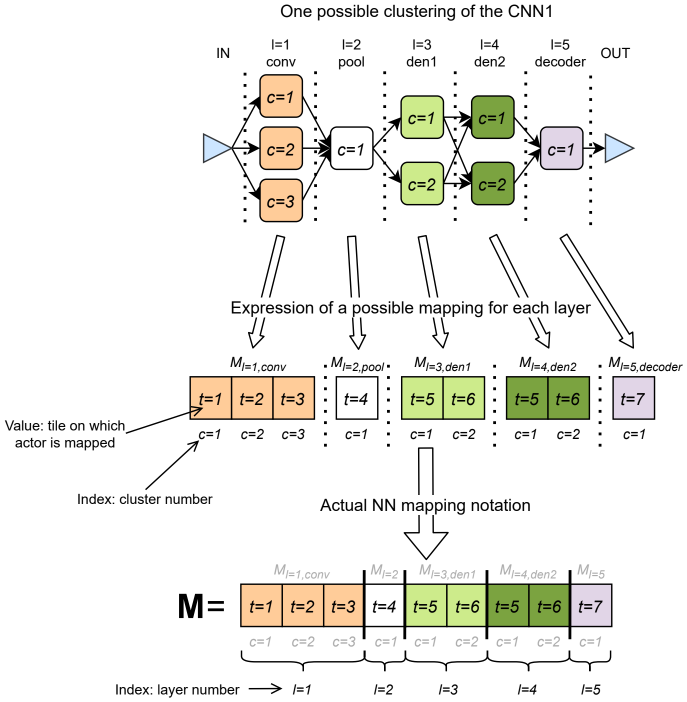


Figure VI.3 – Example of the mapping formulation used in our DSE flow for a mapping of CNN1.

As shown at the bottom of the figure, the NN mapping **M** is a table of the L tables $M[l]$. It is important to note that all communication channels are mapped on the unique shared memory, hence they are not considered in the provided formulation.

To prune the design space of lesser mappings, we use the Branch & Bound algorithm and the analytical models introduced in Chapters IV and V. The analytical models are used to evaluate the latency, throughput, power and energy of the mappings and perform their selection. The selection is done by comparing *mapping scores*, which we define as the latency multiplied by the energy. The score allows taking into account both timing and power properties. It can be modified by the user in the code. An illustration of the mapping search flow is shown in Figure VI.4:

1. Inside the main procedure, the first mapping is generated ❶. In the first mapping, all actors are mapped on tile 1. If the clustering of the last layer is higher than 1, then the decoder actor is also added to the mapping.
2. The Branch & Bound-enhanced mapping search then starts. While the last mapping is not reached, the next possible branches are evaluated and the best one is selected.
   (a) Possible next branches from the provided mapping are generated and evalu-

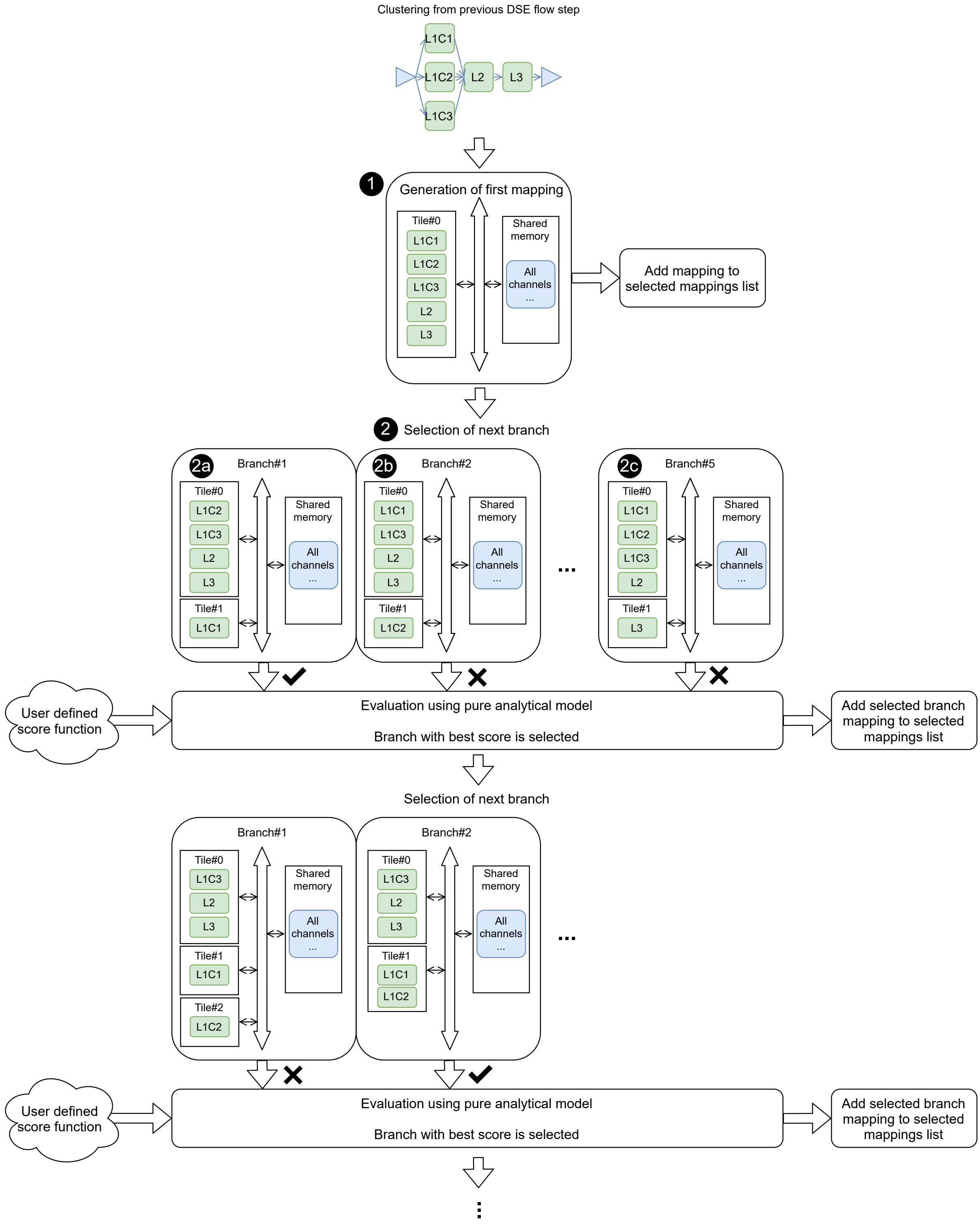

Clustering from previous DSE flow step
L1C1
L1C2
L1C3
L2
L3
1
Generation of first mapping
Tile#0
Shared memory
All channels ...
Add mapping to selected mappings list
2
Selection of next branch
2a
Branch#1
2b
Branch#2
2c
Branch#5
Tile#1
User defined score function
Evaluation using pure analytical model
Branch with best score is selected
Add selected branch mapping to selected mappings list
Selection of next branch
Tile#2

Figure VI.4 – Mapping exploration flow

ated ❷. The branches are generated following two criterias:

  i. Intra-layer parallelism leverage: the algorithm checks for every NN layer if all clusters of this layer are mapped on different tiles. If this is not the case, then the algorithm increases by 1 the tile number on which one of the cluster is mapped, within the limit of the maximum number of tiles $T_{\max}$. 2a and 2b are examples of intra-layer parallelism optimization induced branches.

  ii. Inter-layer parallelism leverage: the algorithm checks if the actors from different NN layers are mapped on different tiles. If this is not the case, the algorithm increases by 1 the tile number on which the actors of a layer, unless it exceeds $T_{\max}$. The mapping of layers on different tiles allows ensuring that they are executed simultaneously, thus leading to an optimized pipeline execution. 2c is an example of mapping issued from this optimization.

(b) The different branches are then evaluated and ranked according to their score. The branch with the highest score is added to the selected mapping list (❸ and ❹).

(c) When no new branch is generated, it means that the optimization process as finished for one full branch.

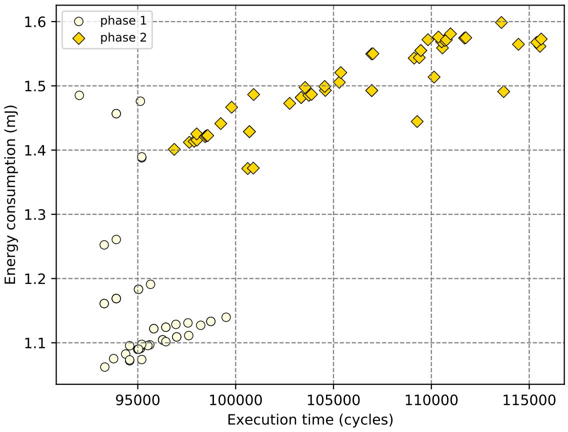
3. The list of all mappings is returned.

The discarded branches are stored, and can be re-used when performing additional iterations of the mapping search in case additional mappings must be considered. Once stage 1 terminated, the DSE loop is finished. The selected mappings are passed to the second phase. In stage 2, mappings are re-evaluated using the simulation-based modeling flow to offer a more accurate assessment of non functional properties. Mappings that do not meet user defined constraints are discarded, and the remaining mappings are ranked by their timing and energy properties and returned to the user. In the next section, we provide a demonstration of the use of the proposed DSE flow on several NNs.

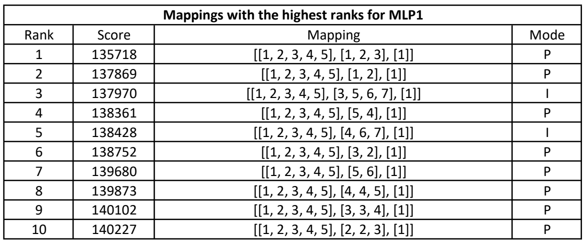
VI.3 Demonstration of the use of the DSE flow

In order to demonstrate the use of the proposed DSE flow, we tested its application on the MLP1 and two CNNs: the CNN1 and the CNN2, which features the LeNet5 topology, as shown in Figure III.12. We left the platform dimensions unspecified i.e. we let the DSE flow jointly optimize the platform in regards to the number of tiles and private memory size of each tile[3], and the NN's deployment. We set the maximum number of tiles $T_{\max} = 7$. Figures VI.5, VI.6 and VI.7 give the scores of the highest ranked selected mappings of the considered NNs, their encoding with respects to Figure VI.3 and their communication mode (i.e. use of power management). For the MLP1, 181 mappings were selected, for the CNN1 over 500 and for LeNet5 over 1000.

Results analysis: The results show that the first approach considered when deploying NN, which aims at leveraging the intra and extra-layer parallelisms at most on the maximum number of tiles possible, is not systematically the best way to deploy NNs.



(a) MLP1 - 50 best mappings

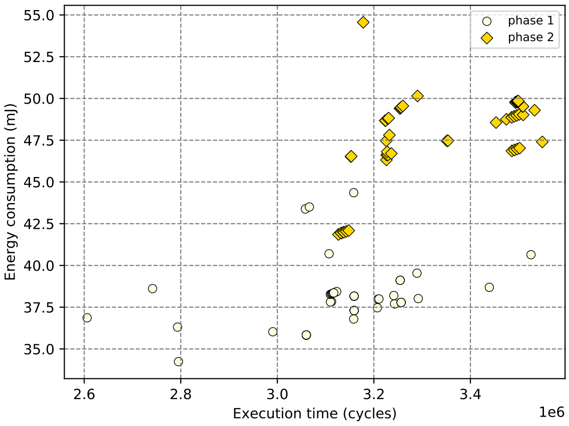


Mappings with the highest ranks for MLP1			
Rank	Score	Mapping	Mode
1	135718	[[1, 2, 3, 4, 5], [1, 2, 3], [1]]	P
2	137869	[[1, 2, 3, 4, 5], [1, 2], [1]]	P
3	137970	[[1, 2, 3, 4, 5], [3, 5, 6, 7], [1]]	I
4	138361	[[1, 2, 3, 4, 5], [5, 4], [1]]	P
5	138428	[[1, 2, 3, 4, 5], [4, 6, 7], [1]]	I
6	138752	[[1, 2, 3, 4, 5], [3, 2], [1]]	P
7	139680	[[1, 2, 3, 4, 5], [5, 6], [1]]	P
8	139873	[[1, 2, 3, 4, 5], [4, 4, 5], [1]]	P
9	140102	[[1, 2, 3, 4, 5], [3, 3, 4], [1]]	P
10	140227	[[1, 2, 3, 4, 5], [2, 2, 3], [1]]	P

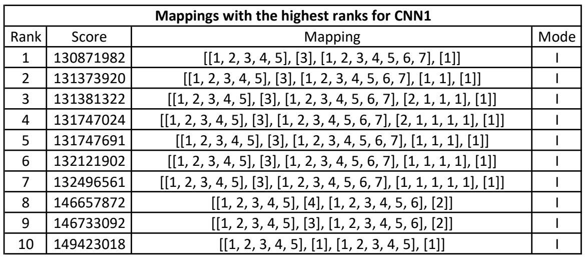
(b) MLP1 - 10 best mappings details

Figure VI.5 – Graph and table showing the highest score mappings found for the MLP1. In (a), the graph shows the predicted execution time and energy of the 50 highest ranked mappings based on the phase in the flow. As a reminder, in phase 1, pure analytical models are used, whereas in phase 2, the simulation-based evaluation flow is used. In (b), the score, encoding with respects to Figure VI.3 and communication mode of the 10 highest ranked mappings are provided. For the communication mode, P stands for polling (without power management), I stands for interrupt (with power management).

[3] The memory needs of tiles are predicted using the private memory size model provided in Appendices C



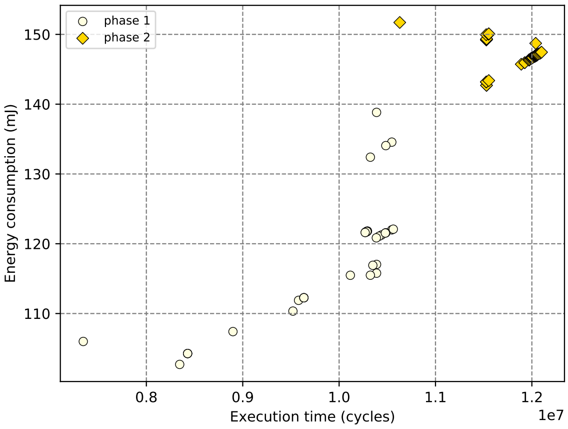
(a) CNN1 - 50 best mappings



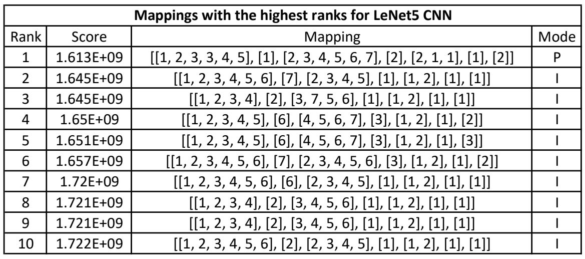
Mappings with the highest ranks for CNN1			
Rank	Score	Mapping	Mode
1	130871982	[[1, 2, 3, 4, 5], [3], [1, 2, 3, 4, 5, 6, 7], [1]]	I
2	131373920	[[1, 2, 3, 4, 5], [3], [1, 2, 3, 4, 5, 6, 7], [1, 1], [1]]	I
3	131381322	[[1, 2, 3, 4, 5], [3], [1, 2, 3, 4, 5, 6, 7], [2, 1, 1, 1], [1]]	I
4	131747024	[[1, 2, 3, 4, 5], [3], [1, 2, 3, 4, 5, 6, 7], [2, 1, 1, 1, 1], [1]]	I
5	131747691	[[1, 2, 3, 4, 5], [3], [1, 2, 3, 4, 5, 6, 7], [1, 1, 1], [1]]	I
6	132121902	[[1, 2, 3, 4, 5], [3], [1, 2, 3, 4, 5, 6, 7], [1, 1, 1, 1], [1]]	I
7	132496561	[[1, 2, 3, 4, 5], [3], [1, 2, 3, 4, 5, 6, 7], [1, 1, 1, 1, 1], [1]]	I
8	146657872	[[1, 2, 3, 4, 5], [4], [1, 2, 3, 4, 5, 6], [2]]	I
9	146733092	[[1, 2, 3, 4, 5], [3], [1, 2, 3, 4, 5, 6], [2]]	I
10	149423018	[[1, 2, 3, 4, 5], [1], [1, 2, 3, 4, 5], [1]]	I

(b) CNN1 - 10 best mappings details

Figure VI.6 – Graph (a) and table (b) showing the highest score mappings found for the CNN1. Refer to the caption of Figure VI.5 for more details.



(a) LeNet5 - 50 best mappings



Mappings with the highest ranks for LeNet5 CNN			
Rank	Score	Mapping	Mode
1	1.613E+09	[[1, 2, 3, 3, 4, 5], [1], [2, 3, 4, 5, 6, 7], [2], [2, 1, 1], [1], [2]]	P
2	1.645E+09	[[1, 2, 3, 4, 5, 6], [7], [2, 3, 4, 5], [1], [1, 2], [1], [1]]	I
3	1.645E+09	[[1, 2, 3, 4], [2], [3, 7, 5, 6], [1], [1, 2], [1], [1]]	I
4	1.65E+09	[[1, 2, 3, 4, 5], [6], [4, 5, 6, 7], [3], [1, 2], [1], [2]]	I
5	1.651E+09	[[1, 2, 3, 4, 5], [6], [4, 5, 6, 7], [3], [1, 2], [1], [3]]	I
6	1.657E+09	[[1, 2, 3, 4, 5, 6], [7], [2, 3, 4, 5, 6], [3], [1, 2], [1], [2]]	I
7	1.72E+09	[[1, 2, 3, 4, 5, 6], [6], [2, 3, 4, 5], [1], [1, 2], [1], [1]]	I
8	1.721E+09	[[1, 2, 3, 4], [2], [3, 4, 5, 6], [1], [1, 2], [1], [1]]	I
9	1.721E+09	[[1, 2, 3, 4], [2], [3, 4, 5, 6], [1], [1, 2], [1], [1]]	I
10	1.722E+09	[[1, 2, 3, 4, 5, 6], [2], [2, 3, 4, 5], [1], [1, 2], [1], [1]]	I

(b) LeNet5 - 10 best mappings details

Figure VI.7 – Graph (a) and table (b) showing the highest score mappings found for the CNN2. Refer to the caption of Figure VI.5 for more details.

For example, on all the highest ranked mappings of the CNN2 (LeNet5) as shown in Figure VI.7 (b), the second convolution layers and all the dense layers are never clusterized at the maximum. The first convolution layer, which has the biggest computation workload of all layers for this NN, is also not always split into the maximum of actor, as seen in mappings ranked 3, 4, 5, 8 and 9. On the highest ranked mapping (1), this layer is split in 6 actors, which is the maximum for this layer, but two of the actors are mapped onto the same tile. Regarding the streaming aspect, it can be observed that it is leveraged in most mappings. However, we can observe that the mappings ranked 7 up to 10 use less than 7 tiles, which shows that favoring the use of as many tiles as possible to support a streaming execution does not necessarily lead to better mapping score.

The same observations can be done for the MLP1 (Figure VI.5), for which the two highest ranked mappings only use 5 tiles, and only two of the 10 highest ranked mappings use 7 tiles. The 10 highest ranked mappings of the MLP1 also do not maximize the use of clustering: the first layer is split into 5 actors when it could be split into more, and the second layer is split in either 2, 3 or 4 actors.

The results also show the importance of having a flow that allows modeling the effect of using power management. For example, we can observe on the results of the LeNet5 application on Figure VI.7 (b) that the highest ranked mapping does not use power management, while the next 9 highest ranked mappings uses it. For the MLP1, most of the highest ranked mappings do not use power management. For the CNN1 (Figure VI.6) on the contrary, all of the 10 highest ranked mappings use power management.

Comparison of score in first phase and second phase: The pure analytical models tend to be too optimistic and underestimate the overhead in timing and energy due to shared resource contention. For example in Figure VI.7 (a), the design points as evaluated by the simulation-based flow (phase 2) are found on the top right of the graph, with systematically higher execution time and energy consumption than the points evaluated by the pure analytical models (phase 1). Similar observations can be done on all different provided graphs. This shows the importance of re-evaluating the mappings using the simulation-based flow, to conveniently rank them and select the most optimized ones.

Evaluation speed: The pruning phase using pure analytical models (phase 1) takes on average less than 1 minute for every considered application. This fast pruning step is really important as it allows quickly disregarding less optimized mappings, which would take much more time to evaluate in the second phase (we evaluate in the following section the pruning efficiency). The evaluation phase using the simulation-based flow (phase 2) takes approximately 2 hours and 15 minutes to evaluate 500 different mappings, per application. The flow thus allows having a fast evaluation of NN mappings onto multi-core platforms ahead of deployment phase, which is crucial to optimize both the hardware and software implementation of NN at design phase.

VI.4 DSE flow evaluation

In order to evaluate the contributions and possible limitations of using Branch & Bound for clustering/mapping search with pure analytical models, we perform different tests:

1. Comparison of Branch & Bound-enhanced and exhaustive clustering search,
2. Comparison of Branch & Bound-enhanced and exhaustive mapping search,
3. Comparison of the use of pure analytical modeling flow and simulation-based modeling flow to identify optimized mappings with the Branch & Bound algorithm.

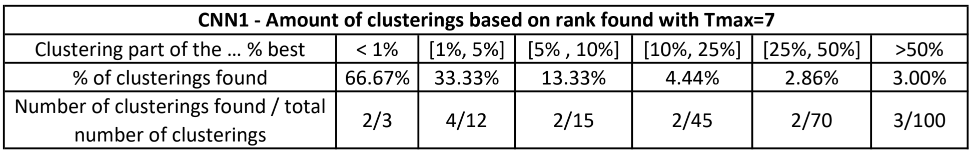
VI.4.1 Comparison of Branch & Bound-enhanced and exhaustive clustering search

Test configuration: To test the gain regarding exploration effort of using Branch & Bound and verify that it allows reliably selecting high rank clusterings, we run the exhaustive clustering search, and rank clusterings by score. We then run the Branch & Bound-enhanced clustering search, and evaluate the ranking of the selected clusterings using Branch & Bound. This test is done for the three NNs considered for DSE: the MLP1, the CNN1 and LeNet5. It is done with $T_{max} = 7$.

Results for the MLP1: The total number of clusterings selected using the Branch & Bound enhanced search is 9 while the total number of clusterings found using exhaustive search is 49. More than 80 % of the clustering design space is pruned. The clusterings with the following ranks are found using Branch & Bound, compared to exhaustive: 1, 3, 5, 6, 7, 20, 35 and 42. The highest ranked clustering (1) and five clusterings from the top 7 are found. We observe that the Branch & Bound enhanced search converges diligently towards high ranked clusterings, as only three clusterings were found outside of the 10 highest ranked ones.

Results for the CNN1: The total number of clusterings selected using the Branch & Bound enhanced search is 15 while the total number of clusterings is 245, which represents a reduction of the design space of more than 93 %. Table VI.1 provides the number of clusterings found in regards to their ranking. For example, in the first column, we can read the amount of clusterings found by the Branch & Bound algorithm that are part of the 1 % best mappings found through exhaustive search: 2 out of 3.

Table VI.1 – Number of clusterings selected by the Branch & Bound enhanced search based on their rank, with $T_{max} = 7$

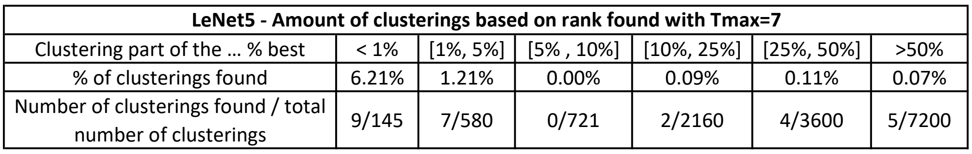


CNN1 - Amount of clusterings based on rank found with Tmax=7						
Clustering part of the … % best	< 1%	[1%, 5%]	[5% , 10%]	[10%, 25%]	[25%, 50%]	>50%
% of clusterings found	66.67%	33.33%	13.33%	4.44%	2.86%	3.00%
Number of clusterings found / total number of clusterings	2/3	4/12	2/15	2/45	2/70	3/100

It can be noted that an important proportion of highly ranked clusterings are found and most of the low rank clusterings are eliminated. Regarding the 10 clusterings at highest rank, the Branch & Bound algorithm managed to find the 1st, 3rd, 5th, 6th and 7th compared to exhaustive.

Results for the CNN2 (LeNet5): The total number of clusterings selected using the Branch & Bound enhanced search is 27 while the total number of clusterings is 14 406, which represents a reduction of the design space of more than 99.8 %. Table VI.2 provides the number of clusterings found in regards to their ranking. In addition to this, the Branch & Bound algorithm managed to find the 1st, 3rd, 5th and 6th clustering of the 10 highest ranked clusterings.

Table VI.2 – Number of clusterings found by the enhanced search with Branch & Bound based on their rank with $T_{max} = 7$



LeNet5 - Amount of clusterings based on rank found with Tmax=7						
Clustering part of the … % best	< 1%	[1%, 5%]	[5% , 10%]	[10%, 25%]	[25%, 50%]	>50%
% of clusterings found	6.21%	1.21%	0.00%	0.09%	0.11%	0.07%
Number of clusterings found / total number of clusterings	9/145	7/580	0/721	2/2160	4/3600	5/7200

Discussions: The first observation that can be made regarding the enhanced clustering search flow is that it offers an important pruning of the clustering design space up to more than 99.8 % for the LeNet5 on 7 tiles. This reduction of the design space is very important, as reducing the number of clusterings will also drastically reduce the amount of mappings considered, leading to faster evaluation times.

The proposed clustering exploration flow also finds reliably most optimized clusterings. It can be observed that for all tested configurations, the clustering with the highest rank is always found. The found clusterings are also in higher proportions in the intervals corresponding to highest ranks. In this study we considered three NNs with different number and type of layers, which offers different computation/communication workload. This allows validating the proposed clustering exploration flow.

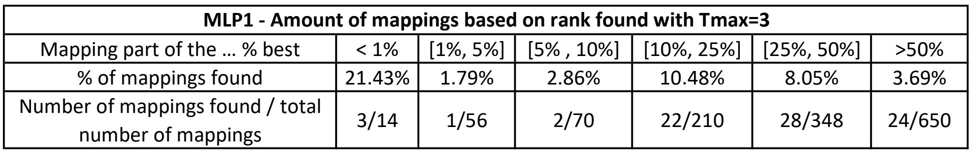
VI.4.2 Comparison of Branch & Bound-enhanced and exhaustive mapping search

Test configuration: We test the gain regarding exploration effort of using Branch & Bound and verify that it allows reliably selecting high rank mappings. For the clusterings obtained through the use of the Branch & Bound algorithm, we run the exhaustive mapping search. The mappings obtained through the exhaustive search are ranked in regards to their score. Then we run the Branch & Bound enhanced mapping search and evaluate the number of found mappings and their ranks in the list of all mappings obtained through exhaustive search. This test is done with the MLP1 and the CNN1, and $T_{max} = 3$ in order to reduce the important time needed to perform the exhaustive mapping search.

Results with the MLP1: With $T_{max} = 3$, the total number of mappings selected using the Branch & Bound enhanced search is 40 while the total number of mappings is 1348, which represents a reduction of the design space of more than 97 %. Table VI.3 provides the number of mappings found in different rank intervals. For example, in the first column, we can read the amount of mappings found by the Branch & Bound algorithm that are part of the 1 % best mappings found through exhaustive search. The Branch & Bound algorithm allows finding 3 of the 14 mappings in this category, which corresponds to 21.43 %. In addition to this, the Branch & Bound algorithm managed to find the 2nd, and 8th mappings of the 10 highest ranked mappings.

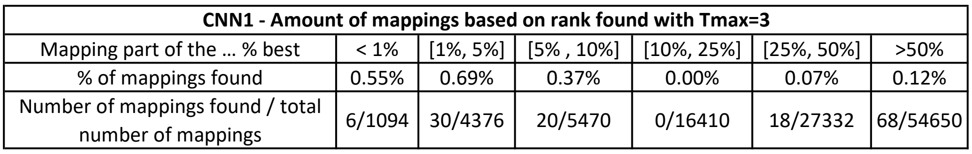
Results with the CNN1: With $T_{max} = 3$, the total number of mappings selected using the Branch & Bound enhanced search is 71 while the total number of mappings is 109 332, which represents a reduction of the design space of more than 99.9 %. Table VI.4 provides the number of mappings found in different rank intervals.

Table VI.3 – Number of mappings found by the Branch & Bound-enhanced search based on their rank with T_{max} = 3. The lower in the rank interval the higher the score of the mapping. E.g. mappings that belongs in the < 1 % rank range have a highest score than 99 % than the other mappings.



MLP1 - Amount of mappings based on rank found with Tmax=3						
Mapping part of the ... % best	< 1%	[1%, 5%]	[5% , 10%]	[10%, 25%]	[25%, 50%]	>50%
% of mappings found	21.43%	1.79%	2.86%	10.48%	8.05%	3.69%
Number of mappings found / total number of mappings	3/14	1/56	2/70	22/210	28/348	24/650

Table VI.4 – Number of mappings found by the enhanced search with Branch & Bound based on their rank with T_{max} = 3



CNN1 - Amount of mappings based on rank found with Tmax=3						
Mapping part of the ... % best	< 1%	[1%, 5%]	[5% , 10%]	[10%, 25%]	[25%, 50%]	>50%
% of mappings found	0.55%	0.69%	0.37%	0.00%	0.07%	0.12%
Number of mappings found / total number of mappings	6/1094	30/4376	20/5470	0/16410	18/27332	68/54650

Unfortunately for this NN, the Branch & Bound algorithm couldn't find any of the 10 best mappings. When evaluating different branches with Branch & Bound, the best one is selected. But sometimes, a less optimized branch at the time of selection, which is then discarded, can eventually lead to a better score after several optimizations. In our case, we only go down one main branch until the mapping cannot be optimized anymore, and then we stop exploring. Considering additional iterations afterwards by exploring discarded branches would allow finding better mappings. Despite not finding the 10 best mappings, it is possible to note that the Branch & Bound enhanced search allows finding a notable proportion of the most optimized mappings in Table VI.4.

Discussions: The design space for mappings is significantly vaster than the clusterings' one, as discussed in Section VI.2.2. We can see that the proposed DSE flow allows reducing the huge design space by more than 97 % in both studied applications. Similarly to what was observed with clusterings, we can expect the design space pruning to be even more important with more complex applications and higher numbers of tiles.

The proposed search algorithm allows finding in majority mappings within the 5 % with highest scores. A non marginal proportion of less optimized mappings is also selected, due to this algorithm requiring more iterations than the clustering search algorithm to reach optimized mappings. This is especially true since two optimizations (intra and inter-layer leverage) are considered simultaneously. One can note that the mapping search does not systematically finds the mapping with the first rank returned by the exhaustive search. This can be justified by the fact that branches with a lower score at the time of selection are disregarded, but could actually lead to a better score than the selected branch after additional optimization steps. This phenomena is amplified by the two optimizations (intra and inter-layer parallelization) leveraged simultaneously, which lead to more branches disregarded at each step, and thus a faster convergence time with a smaller number of selected mappings - but also a higher risk of disregarding important branches. To alleviate this, and especially if the user constraints are not met, it is possible to perform a new iteration of the DSE flow. During this new iteration, the remaining branches (i.e. the pruned out ones) are re-evaluated and the best ones are explored to select additional mappings. The possibility to find more optimized mappings, including the first ranked ones, is then higher.

While the best mappings are not guaranteed to be found after the first clustering/mapping search, a notable proportion of optimized mappings from the $< 5\,\%$ range are found, which should be sufficient to satisfy user defined constraints without requiring a second execution of the DSE.

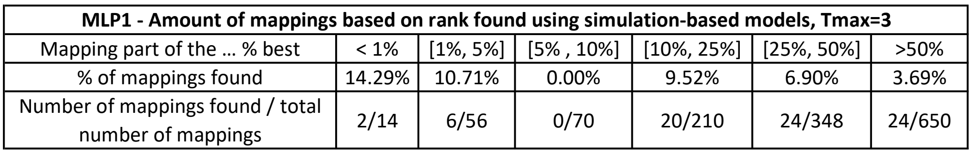
VI.4.3 Use of pure analytical models for pruning

To assess the advantages/drawbacks of using pure analytical models to perform the design space pruning ahead of evaluation using SystemC, we perform the following test: we run the Branch & Bound mapping exploration process for a limited number of tiles $T_{\max} = 3$ using the simulation-based flow and the pure analytical flow. We then evaluate the proportion of the mappings that were found by both modeling flows. We also compare the proportion of mappings found based on rank by the simulation flow and the pure analytical model against the exhaustive search, as shown in Tables VI.3 and Table VI.4 for the analytical models, and VI.5 and VI.6 for the simulation.

For the MLP1, the analytical flow found 80 mappings, while the simulation flow found 76. For the CNN1, the analytical flow found 142 and the simulation flow found 120. The analytical modeling flow tend to find slightly more mappings than the simulation-based flow. For the MLP1, the flows have approximately 72 % mappings in common, and for the CNN1, they have approximately 57 % mappings in common. Overall, both flows tend to go for the same branches in the first iterations, before they split and find different mappings.

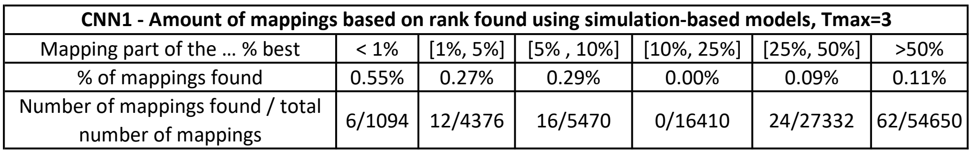
Regarding the proportion of mappings found based on ranking, we can observe that they are nearly identical for both flows on both NNs. For the MLP1, we note that the simulation flow found a bigger proportion of mappings in the range $[1\%, 5\%]$ (6 mappings found against 1 for the analytical modeling flow), while for the CNN1, the analytical modeling flow found a bigger proportion of mappings in the range $[1\%, 5\%]$ (30 against 12 for the simulation). These observations allow confirming that, despite the low reliability of the pure analytical models, they can be used confidently to prune the design space.

Table VI.5 – Number of mappings found by the enhanced search with Branch & Bound using the simulation-based flow with $T_{max} = 3$



MLP1 - Amount of mappings based on rank found using simulation-based models, Tmax=3						
Mapping part of the … % best	< 1%	[1%, 5%]	[5% , 10%]	[10%, 25%]	[25%, 50%]	>50%
% of mappings found	14.29%	10.71%	0.00%	9.52%	6.90%	3.69%
Number of mappings found / total number of mappings	2/14	6/56	0/70	20/210	24/348	24/650

Table VI.6 – Number of mappings found by the enhanced search with Branch & Bound using the simulation-based flow with $T_{max} = 3$



CNN1 - Amount of mappings based on rank found using simulation-based models, Tmax=3						
Mapping part of the … % best	< 1%	[1%, 5%]	[5% , 10%]	[10%, 25%]	[25%, 50%]	>50%
% of mappings found	0.55%	0.27%	0.29%	0.00%	0.09%	0.11%
Number of mappings found / total number of mappings	6/1094	12/4376	16/5470	0/16410	24/27332	62/54650

VI.5 Conclusion

In this chapter, we proposed an efficient DSE flow to evaluate NN mappings on multi-core platforms. The flow demonstrates how the models proposed in Chapter IV and Chapter V can be used to efficiently explore the design space. The results show that the proposed DSE flow allows to quickly and reliably find mappings with optimized timing and energy properties. It allows evaluating mappings with and without power management and see when implementing it is beneficial, as shown for example in Figure VI.7 (b). It is important to note that the analytical models used for phase 1 are currently an important limitation. As shown in previous chapters, the pure analytical models bear a prediction error of almost 30 % on timing and 20 % on power. While they allow finding a notable proportion of the highest ranked clusterings/mappings obtained through exhaustive search, more confident models would offer better results. Additional research work is required to propose analytical models that can confidently prune the design space of lesser mappings and guarantee finding a bigger proportion of the high-ranked clusterings/mappings.

Chapter VII

Conclusion

VII.1 Synthesis

This thesis work takes place in a context where the need for deploying NNs on edge devices is growing. Most edge devices feature multi-core SoCs. The evaluation of NNs mappings on multi-core platforms is necessary to find configurations that optimize temporal properties (latency, throughput) as well as power consumption and energy under user-defined constraints. It needs to be carried out using models that allow rapid yet confident evaluation early in the design process so as to avoid time-consuming prototyping and testing phases, which can lead to sub-optimal solutions. These models must offer scalability in regard to the following aspects, in order to allow efficient exploration of a large solution space:

(a) the NN complexity in regards to the types (e.g. convolution, dense) and number of layers and amount of neurons.

(b) the expression and use of intra- and inter-layer parallelism from the NN (clustering/mapping) - and thus various computation and communication workloads,

(c) the contentions for shared resources such as memories and communication buses,

(d) the dynamic behavior of the system and use of power management techniques such as clock gating and different communication procedures (polling-based or interrupt-based).

In Chapters IV and V, we have presented a modeling flow for the prediction of timing properties and energy of NN deployed on multi-core platforms. The modeling flow is hybrid: it relies on system level simulation described using the SystemC library and analytical models characterized through multi-linear regression of measurements. The calibration of the models through measurements offers a fine-grained modeling of the effect of the compiler and hardware over the use of estimations for calibration. The use of analytical models along with system level simulation allows a fast evaluation of interesting mappings while modeling shared resources contentions. The flow was tested against a real implementation of 54 different mappings of 4 NNs. The modeling flow offers timing prediction with more than 97 % accuracy and energy prediction with more than 93 % accuracy. The high accuracy is rendered possible by the separation of computation and communication using the SDF model of computation and respected by our composable model of architecture.

The predictions are fast with an average simulation time of 0.23 s for 100 iterations of simulation - 20 s when considering also the compilation time of the SystemC framework. The proposed models can be applied to architectures featuring MicroBlaze cores equiped with ISA extensions to support the use of FPUs and Mult, as well as AXI bus. To support other components, the characterization methodology presented in this thesis must be re-done. More specifically to support other processor cores, the base delays (e.g. D_φ and D_Σ) of the computation time model must be re-calibrated by performing a regression of the execution time of layers with varying size of input and number of neurons. In case the communication bus IP is changed, then the calibration of the communication time model must be re-performed, by measuring the base delays used in the model with the new component. For other types of communication medium, the model's structure should be adapted. For the power model, changing components inside the platform also requires re-performing the characterization phase as documented in Sections V.3.1.1 and V.3.1.2.

In Chapter VI we provide a demonstration of how models of a high level of abstraction such as the ones presented in this work can be used to optimize the Hardware / Software implementation of NNs on multi-core platforms. We show that the proposed DSE flow allows finding mappings that offer optimized timing and energy, while discarding less optimized solutions to guarantee fast exploration time, due to Branch & Bound. Our approach allows determining when the use of power management should be considered on the platform to optimize timing properties and energy. The proposed modeling and DSE flows can be used for two distinct objectives:

1. Evaluate and find optimized NN mappings on a fixed multi-core platform.
2. Jointly optimize multi-core platforms' dimensions (number of cores, private memory sizes) and NN deployments.

The work presented in this thesis provides the following answers to the identified research challenges:

1. How to provide fast yet accurate evaluation early in design phases of timing and energy properties for streaming NNs deployments onto multi-core platforms?

→ The proposed methodology allows obtaining fast yet accurate timing and power models to evaluate NN deployments on multi-core platforms early in design phase. This is rendered possible by:

— Analytical formulas for both timing and power models, which allows offering a fast evaluation time.

— Characterization through measurements for both models to offer accurate prediction, thanks to the fine-grained analysis of the compiler's and hardware effects on non functional properties.

— Simulation to model the shared resources.

— Strict separation of computation and communication phases using the SDF MoC, and supported by our MoA.

2. Is the evaluation of possible solutions through a model-based approach relevant against rapid prototyping approaches?

   → The proposed modeling flow can evaluate a NN mapping (without requiring the NN to be trained) in 20 s with high accuracy. Under the assumption that the evaluation of NN mappings using implementation and testing takes approximately 2 minutes, which is the time required by our automatized measurement infrastructure [1]. In that case, the proposed modeling flow is 6 times faster than the rapid prototyping approach. For the evaluation of 1000 mappings, the models would take approximately 5 hours and 30 minutes while the rapid prototyping approach would take approximately 28 hours. It is also worth remembering that the number of mappings evaluated in a single loop of the proposed DSE flow (which disregards a large part of the less optimized design space) exceeds 1000 mappings. High level models are thus a necessity to explore the design space of NN deployments on multi-core platforms.

3. Is a model-based approach suited for early, fast and confident DSE of streaming NNs deployments on multi-core platforms?

   → We demonstrate how such a DSE flow can be setup using Branch & Bound to rapidly find most optimized clusterings and mappings, while discarding less promising branches. We use high level analytical models due to their very high evaluation speed. They are then complemented by our simulation-based flow for

[1] Note: In this duration, we only consider the time to compile the application's C code for MicroBlaze, program the FPGA, program the MicroBlaze cores, run the application and perform the timing and power measurements. It excludes the training of the NN, generation of the FPGA design bitstream and generation of the BSP, required libraries and NN inference source files in C for MicroBlaze cores.

confident assessment and ranking of selected mappings. The list of mappings sorted by rank is then provided to the user. The flow offers the possibility to automatically generate C code needed to infer the mappings selected by the user.

VII.2 Identified limitations

We identified the following limitations to our flow:

1. We note an influence of the communication rate on power and energy prediction error when it becomes important. The maximum prediction error is 7 % for a mapping in which tiles are 70 % of the time in communication. This error remains relatively low.
2. On single-core platforms with important private memory allocated (1024 kB and 2048 kB), the modeling flow has high prediction error (> 10 %). It shows that the assumption we made that dynamic power consumption does not depend on private memory size is invalid in this situation. On other tested platforms featuring 5 and 7 tiles including one tile featuring 1024 kB, the prediction error was however low.
3. The demonstration in Chapter VI is done with the pure analytical models from Chapters IV and V. However additional work should be done on these models to render them more efficient for the DSE process.

VII.3 Perspectives

In this thesis, we propose and validate a modeling approach for the prediction and optimization of NN mappings on multi-core platforms. We identify several research perspectives that could enhance the proposed methodology:

Neural Architectural Search (NAS): One major prospect for our prediction and optimization flow would be to integrate NAS. NAS aims at exploring the topology of the input NN, i.e. the number of layers, number of neurons per layer, and possibly the use of compression techniques (see Figure II.1). This is usually done under QoS constraints as altering the NN's architecture can lead to loss of classification accuracy. Recent studies [29, 19, 39] propose to lead the NAS process simultaneously to the DSE for NN executed on embedded systems In these approaches, NAS is led with regards to measured timing properties and energy on a real platform. In comparison to these approaches, using our modeling flow instead of rapid prototyping on real target would highly reduce search effort. Especially since our flow does not require the NN to be trained to evaluate its timing and energy properties.

The DSE flow proposed in this thesis could be used with an input NN with known classification accuracy. It could improve the NN's architecture and evaluate the effect on resource use, timing and energy. After full exploration of the flow, the best mapping found with the starting NN would be compared with the best mapping with the NN updated through NAS. The updated NN could be trained using automatized deep learning flows, such as TensorFlow Lite [43, 44], Apache TVM [48] or N2D2 [50]. This would allow providing the difference in classification accuracy. Our flow could then generate the C source code corresponding to the mapping selected by the user.

External memories (DDR): Another perspective could be to extend this work to architectures with external memories such as DDR, along with caches and/or DMAs. When considering the use of such memories, the focus of the optimization approach is however different: since external memory accesses bear a high timing and energy cost, the optimization process focuses mainly on reducing and optimizing those accesses. The focus is then less on finding a good compromise between the leverage of intra-layer parallelism (clustering) and communications in internal shared memories. Fine-grained optimizations optimizations and compromises offered by our flow bear a reduced effect on timing/power optimizations compared to optimizing external memory accesses, while they can lead to important timing and power savings on devices when not using external memory.

Variability due to data dependent paths : In embedded systems, variability in execution time and energy can be observed due to the hardware architecture and data dependent paths. For example, certain ISA optimize multiplications in trivial cases, e.g. one of the term is 1 or 0. In that case, some multiplication operations are faster to execute than others, which can have important repercussions on non functional properties.

One of the original hypothesis formulated at the start of this thesis work was that variability in execution time and energy for NNs on multi-core platforms was not marginal We however observed marginal variability of execution time and energy in our experiments focused on the platform prototype.

When performing the characterization of our timing and power models on other multicore platforms, engineers might observe non-marginal variability. It can be caused by having different data dependent paths in the targeted hardware. In such case, the proposed methodology must be extended. To deal with variability, approaches relying on Statistical Model Checking (SMC) and probabilistic models are commonly used. A possible prospect would be to identify cases in which data variability is observed and discuss the application and performance of such modeling techniques.

VII.4 Ouverture

In many fields, NNs are now helping to solve problems that were unthinkable only a decade ago. For example, hearing aids are now implementing NNs, which can recognize environments (e.g.: work place, restaurant, concert) in order to adapt the frequency treatment in real time [102]. NNs can help radiologists detect different types of cancer with high confidence [103]. Regarding agriculture, NNs can be used to keep track of livestock and detect abnormal behaviors [96] or to help enhance sowing, watering and harvesting [104]. In the IoT field, there are many applications [3]. In the field of transportation, NNs are used for semantic segmentation [39], and automobile systems now integrate NNs into Advanced Driving Assistance Systems (ADAS) [105, 106]. The flow proposed in this work can enable solutions to be found quickly and confidentially for deploying NNs for this type of application, satisfying their time and energy constraints.

List of publications

Lectures in international conferences:

- — [107] Quentin Dariol, Sébastien Le Nours, Domenik Helms, Ralf Stemmer, Sébastien Pillement, and Kim Grüttner. « Fast Yet Accurate Timing and Power Prediction of Artificial Neural Networks Deployed on Clock-Gated Multi-Core Platforms ». Conference: *RAPIDO'23: Rapid Simulation and Performance Evaluation for Design Optimization: Methods and Tools*. Toulouse, France, Association for Computing Machinery, January 2023, pp. 79–86. DOI: 10.1145/3579170.3579263.
- — [108] Quentin Dariol, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, Domenik Helms, and Kim Grüttner. « A Hybrid Performance Prediction Approach for Fully-Connected Artificial Neural Networks on Multi-core Platforms ». In: *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2022)*. Ed. by Alex Orailoglu, Marc Reichenbach, and Matthias Jung. Springer International Publishing, 2022, pp. 250–263. DOI: 10.1007/978-3-031-15074-6_16

Communications in national colloquium:

- — [109] Quentin Dariol, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, Domenik Helms, and Kim Grüttner. « Early Performance and Energy Prediction of Neural Networks Deployed on Multi-Core Platforms ». In: *29° Colloque sur le traitement du signal et des images*. 2023-1144. Grenoble: GRETSI - Groupe de Recherche en Traitement du Signal et des Images, Aug. 2023, p. 309–312. Available at: https://gretsi.fr/data/colloque/pdf/2023_dariol1144.pdf.
- — [110] Quentin Dariol, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, Domenik Helms and Kim Grüttner. «Hybrid Performance Prediction Models for Fully-Connected Neural Networks on MPSoC». In: *16ème Colloque National du GDR SOC2*. June 2022. Available at: https://hal.science/hal-03758026
- — [111] Quentin Dariol, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, Kim Grüttner, and Domenik Helms. « A Measurement-based Performance Evaluation Framework for Neural Networks on MPSoCs ». In: *15ème Colloque National du GDR SOC2*. June 2021. Available at: https://hal.archives-ouvertes.fr/hal-03248152

Technical report:

- — [112] Quentin Dariol, Sebastien Le Nours, Sebastien Pillement, Kim Grüttner, Domenik Helms, and Ralf Stemmer. « Setup of an Experimental Framework for Performance Modeling and Prediction of Embedded Multicore AI Architectures ». Tech. rep. Nantes University, IETR UMR CNRS 6164, France and Deutsches Zentrum für Luft und Raumfahrt (DLR), Germany, 2022. Available at: https://hal.science/hal-03546804

Open source Git repository: https://gitlab.univ-nantes.fr/lenours-s/pssim4ai

BIBLIOGRAPHY

[1] Nestor Maslej, Loredana Fattorini, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Helen Ngo, Juan Carlos Niebles, Vanessa Parli, Yoav Shoham, Russell Wald, Jack Clark, and Raymond Perraul. *The AI Index 2023 Annual Report*. Tech. rep. Stanford Institute for Human-Centered Artificial Intelligence (HAI), Apr. 2022. URL: `https://aiindex.stanford.edu/wp-content/uploads/2023/04/HAI_AI-Index-Report_2023.pdf`.

[2] John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. *Dartmouth Summer Research Project on Artificial Intelligence*. Last accessed: 11.10.2023. 1956. URL: `https://home.dartmouth.edu/about/artificial-intelligence-ai-coined-dartmouth`.

[3] Nick G. *How Many IoT Devices Are There in 2023?* Last accessed: 27.09.2023. Aug. 2023. URL: `https://techjury.net/blog/how-many-iot-devices-are-there/`.

[4] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. *Hello Edge: Keyword Spotting on Microcontrollers*. 2018. arXiv: `1711.07128 [cs.SD]`.

[5] Warren S. McCulloch and Walter Pitts. « A logical calculus of the ideas immanent in nervous activity ». In: *The bulletin of mathematical biophysic* 5 (Dec. 1943), pp. 115–133.

[6] Frank Rosenblatt. « The perceptron: a probabilistic model for information storage and organization in the brain. » In: *Psychological review* 65 6 (1958), pp. 386–408.

[7] Frank Rosenblatt. *Principles of Neurodynamics Perceptrons and the Theory of Brain Mechanisms*. Washington DC: Spartan Books, 1962.

[8] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[9] D. Rumelhart, G. Hinton, and R. Williams. « Learning representations by back-propagating errors ». In: *Nature 323* (1986), pp. 533–536. DOI: `https://doi.org/10.1038/323533a0`.

[10] Yann Lecun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L.D. Jackel. « Backpropagation applied to handwritten zip code recognition ». English (US). In: *Neural Computation* 1.4 (1989), pp. 541–551. ISSN: 0899-7667.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. « Gradient-based learning applied to document recognition ». In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: `10.1109/5.726791`.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. « ImageNet Classification with Deep Convolutional Neural Networks ». In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. « Imagenet large scale visual recognition challenge ». In: *International journal of computer vision* 115 (2015), pp. 211–252.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. « ImageNet: A large-scale hierarchical image database ». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[15] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. « Design Space Exploration of FPGA-based Deep Convolutional Neural Networks ». In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)* (Jan. 2016), pp. 575–580. ISSN: 2153-697X. DOI: 10.1109/ASPDAC.2016.7428073.

[16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. « Edge Computing: Vision and Challenges ». In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

[17] Nilupulee A. Gunathilake, William J. Buchanan, and Rameez Asif. « Next Generation Lightweight Cryptography for Smart IoT Devices: : Implementation, Challenges and Applications ». In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. 2019, pp. 707–710. DOI: 10.1109/WF-IoT.2019.8767250.

[18] Muhammad Habib ur Rehman, Ibrar Yaqoob, Khaled Salah, Muhammad Imran, Prem Prakash Jayaraman, and Charith Perera. « The role of big data analytics in industrial Internet of Things ». In: *Future Generation Computer Systems* 99 (2019), pp. 247–259. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.04.020.

[19] Ioannis Galanis, Iraklis Anagnostopoulos, Chinh Nguyen, Guillermo Bares, and Dona Burkard. « Inference and Energy Efficient Design of Deep Neural Networks for Embedded Devices ». In: *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (July 2020), pp. 36–41. ISSN: 2159-3469. DOI: 10.1109/ISVLSI49217.2020.00017.

[20] Ourania Spantidi, Ioannis Galanis, and Iraklis Anagnostopoulos. « Frequency-based Power Efficiency Improvement of CNNs on Heterogeneous IoT Computing Systems ». In: *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. 2020, pp. 1–6. DOI: 10.1109/WF-IoT48130.2020.9221252.

[21] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. « FINN: A Framework for Fast, Scalable Binarized Neural Network Inference ». In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2017).

[22] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. « Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices ». In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.2 (July 2018), pp. 292–308. DOI: 10.1109/JETCAS.2019.2910232. arXiv: 1807.07928 [cs.DC].

[23] Salita Sombatsiri, Jaehoon Yu, Masanori Hashimoto, and Yoshinori Takeuchi. « A Design Space Exploration Method of SoC Architecture for CNN-based AI Platform ». In: *SASIMI 2019 Proceedings* (2019).

[24] Angelos Kyriakos, Elissaios-Alexios Papatheofanous, Bezaitis Charalampos, Evangelos Petrongonas, Dimitrios Soudris, and Dionysios Reisis. « Design and Performance Comparison of CNN Accelerators Based on the Intel Movidius Myriad2 SoC and FPGA Embedded Prototype ». In: *2019 International Conference on Control, Artificial Intelligence, Robotics Optimization (ICCAIRO)*. 2019.

[25] Mohammad Hosseinabady and Jose Luis Nunez-Yanez. « Energy optimization of FPGA-based stream-oriented computing with power gating ». In: *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. 2015.

[26] Foivos Tsimpourlas, Lazaros Papadopoulos, Anastasios Bartsokas, and Dimitrios Soudris. « A Design Space Exploration Framework for Convolutional Neural Networks Implemented on Edge Devices ». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018).

[27] Adrian Osterwind, Julian Droste-Rehling, Manoj Rohit Vemparala, and Domenik Helms. « Hardware Execution Time Prediction for Neural Network Layers ». In: *ITEM 2022*. Machine Learning and Principles and Practice of Knowledge Discovery in Databases. Springer Nature Switzerland, 2022, pp. 582–593. DOI: 10.1007/978-3-031-23618-1_39. URL: https://elib.dlr.de/188922/.

[28] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. *An Analysis of Deep Neural Network Models for Practical Applications*. 2017. arXiv: 1605.07678 [cs.CV].

[29] Lennart Heim, Andreas Biri, Zhongnan Qu, and Lothar Thiele. « Measuring what Really Matters: Optimizing Neural Networks for TinyML ». In: *arXiv preprint, arXiv:2104.10645* (Apr. 2021). arXiv: 2104.10645.

[30] Delia Velasco-Montero, Jorge Fernandez-Berni, Ricardo Carmona-Galan, and Angel Rodriguez-Vazquez. « PreVIous: A Methodology for Prediction of Visual Inference Performance on IoT Devices ». In: *IEEE Internet of Things Journal* (2020), pp. 1–1. DOI: 10.1109/jiot.2020.2981684.

[31] Thomas Garbay, Petr Dobias, Wilfried Dron, Pedro Lusich, Imane Khalis, Andrea Pinna, Khalil Hachicha, and Bertrand Granado. « CNN Inference Costs Estimation on Microcontrollers: the EST Primitive-based Model ». In: *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. 2021, pp. 1–5. DOI: 10.1109/ICECS53924.2021.9665540.

[32] Xiaotian Guo, Andy D. Pimentel, and Todor Stefanov. « Automated Exploration and Implementation of Distributed CNN Inference at the Edge ». In: *IEEE Internet of Things Journal* (2023), pp. 1–1. DOI: 10.1109/JIOT.2023.3237572.

[33] Jan Rabaey. *Low Power Design Essentials*. Ed. by Springer. Springer New York, NY, 2009.

[34] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. « Neural Architecture Search: A Survey ». In: *Journal of Machine Learning Research (JMLR)* 20.1 (Jan. 2019), pp. 1997–2017. ISSN: 1532-4435. URL: https://arxiv.org/abs/1808.05377.

[35] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. « Pruning and Quantization for Deep Neural Network Acceleration: A survey ». In: *Neurocomputing* 461 (2021), pp. 370–403. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.07.045.

[36] Zhuo Li, Hengyi Li, and Lin Meng. « Model Compression for Deep Neural Networks: A Survey ». In: *Computers* 12.3 (2023). ISSN: 2073-431X. DOI: 10.3390/computers12030060.

[37] Florent Crozet, Stéphane Mancini, and Marina Nicolas. « Compression par pseudo-randomisation partielle des réseaux de neurones convolutifs sous fortes contraintes mémoire ». In: *28° Colloque sur le traitement du signal et des images*. 001-054. Nancy, Sept. 2022, p. 217–220. URL: https://gretsi.fr/data/colloque/pdf/2022_crozet932.pdf.

[38] Manoj Rohit Vemparala, Nael Fasfous, Pierpaolo Mori, Saptarshi Mitra, Sreetama Sarkar, Alexander Frickenstein, Lukas Frickenstein, Domenik Helms, Naveen Shankar Nagaraja, Claudio Passerone, and Walter Stechele. « Accelerating and Pruning CNNs for Semantic Segmentation on FPGA ». In: *Design Automation Conferene 2022*. Vol. 59. Proceedings of the ACM/EDAC/IEEE Design Automation Conference. IEEE Press, July 2022. URL: https://elib.dlr.de/185416/.

[39] Agathe Archet, François Orieux, Nicolas Ventroux, and Nicolas Gac. « Exploration d'architectures de réseaux de neurones pour la segmentation sémantique d'images aériennes ». In: *29eme Colloque sur le traitement du signal et des images*. 2023-1158. Grenoble: GRETSI - Groupe de Recherche en Traitement du Signal et des Images, Aug. 2023, p. 361–364. URL: https://gretsi.fr/data/colloque/pdf/2023_archet1158.pdf.

[40] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Vol. 103. Springer Texts in Statistics 1. Part of the book series: Springer Texts in Statistics (STS, volume 103). Springer New York, NY, June 2013. ISBN: 978-1-4614-7138-7. DOI: https://doi.org/10.1007/978-1-4614-7138-7.

[41] Muhammad Mudussir Ayub and Franz Kreupl. « A Modular and Distributed Setup for Power and Performance Analysis of Multi-Processor System-on-Chip at Electronic System Level ». In: *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*. 2020, pp. 1–8. DOI: 10.1109/IPCCC50635.2020.9391516.

[42] Paul Pop, Petru Eles, and Zebo Peng. « System-Level Design and Modeling ». In: *Analysis and Synthesis of Distributed Real-Time Embedded Systems*. Boston, MA: Springer US, 2004, pp. 15–40. ISBN: 978-1-4020-2873-1. DOI: 10.1007/978-1-4020-2873-1_2.

[43] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. « TensorFlow: A System for Large-Scale Machine Learning ». In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283.

[44] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, Rocky Rhodes, Tiezhen Wang, and Pete Warden. « TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems ». In: *Proceedings of the 4 th MLSys Conference*. San Jose, CA, USA, 2021. arXiv: `2010.08678 [cs.LG]`.

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. « PyTorch: An Imperative Style, High-Performance Deep Learning Library ». In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[46] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. « Caffe: Convolutional Architecture for Fast Feature Embedding ». In: *Proceedings of the 22nd ACM International Conference on Multimedia*. MM '14. Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 675–678. ISBN: 9781450330633. DOI: `10.1145/2647868.2654889`.

[47] Francois Chollet et al. *Keras*. Last accessed: 10.10.2023, commit: a108358. 2015. URL: `https://github.com/fchollet/keras`.

[48] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. « TVM: An Automated End-to-End Optimizing Compiler for Deep Learning ». In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 578–594. ISBN: 978-1-939133-08-3.

[49] V. V. Zunin. « Intel OpenVINO Toolkit for Computer Vision: Object Detection and Semantic Segmentation ». In: *2021 International Russian Automation Conference (RusAutoCon)*. 2021, pp. 847–851. DOI: `10.1109/RusAutoCon52004.2021.9537452`.

[50] CEA-LIST. *Neural Network Design & Deployment (N2D2)*. N2D2 is an open source CAD framework for Deep Neural Network simulation and full DNN-based applications building. Last accessed: 10.10.2023, 2019. URL: `https://github.com/CEA-LIST/N2D2`.

[51] Stylianos I. Venieris and Christos-Savvas Bouganis. « fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs ». In: *IEEE Transactions on Neural Networks and Learning Systems* (2019).

[52] E.A. Lee and D.G. Messerschmitt. « Synchronous data flow ». In: *Proceedings of the IEEE* 75.9 (1987), pp. 1235–1245. DOI: `10.1109/PROC.1987.13876`.

[53] Stylianos Venieris and Christos Bouganis. « Latency-driven design for FPGA-based convolutional neural networks ». In: *27th International Conference on Field Programmable Logic and Applications (FPL)*. Sept. 2017, pp. 1–8. DOI: `10.23919/FPL.2017.8056828`.

[54] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. « Timeloop: A Systematic Approach to DNN Accelerator Evaluation ». In: *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2019, pp. 304–315. DOI: 10.1109/ISPASS.2019.00042.

[55] Erqian Tang, Svetlana Minakova, and Todor Stefanov. « Energy-Efficient and High-Throughput CNN Inference on Embedded CPUs-GPUs MPSoCs ». In: *Embedded Computer Systems: Architectures, Modeling, and Simulation*. Ed. by Alex Orailoglu, Matthias Jung, and Marc Reichenbach. Cham: Springer International Publishing, 2022, pp. 127–143. ISBN: 978-3-031-04580-6.

[56] Jonas Ney, Dominik Loroch, Vladimir Rybalkin, Nico Weber, Jens Krüger, and Norbert Wehn. « HALF: Holistic Auto Machine Learning for FPGAs ». In: *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. 2021, pp. 363–368. DOI: 10.1109/FPL53798.2021.00069.

[57] Liu Ke, Xin He, and Xuan Zhang. « NNest: Early-Stage Design Space Exploration Tool for Neural Network Inference Accelerators ». In: *Proceedings of the International Symposium on Low Power Electronics and Design*. ISLPED '18. New York, NY, USA: Association for Computing Machinery, 2018. DOI: 10.1145/3218603.3218647.

[58] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. « MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings ». In: *IEEE Micro* 40.3 (2020), pp. 20–29. DOI: 10.1109/MM.2020.2985963. URL: https://maestro.ece.gatech.edu/.

[59] Thomas Garbay. « Zip-CNN ». Thèse de doctorat dirigée par Granado, BertrandHachicha, Khalil et Pinna, Andrea Sciences et technologies de l'information et de la communication Sorbonne université 2023. PhD thesis. Sorbonne Université, 2023. URL: http://www.theses.fr/2023SORUS210.

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. « Deep Residual Learning for Image Recognition ». In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[61] Jooyeon Lee, Junsang Park, Seunghyun Lee, and Jaeha Kung. « Implication of Optimizing NPU Dataflows on Neural Architecture Search for Mobile Devices ». In: *ACM TODAES - Transactions on Design Automation of Electronic Systems* 27.5 (June 2022). ISSN: 1084-4309. DOI: 10.1145/3513085.

[62] Yi-Che Lee, Ting-Shuo Hsu, Chun-Tse Chen, Jing-Jia Liou, and Juin-Ming Lu. « NNSim: A Fast and Accurate SystemC/TLM Simulator for Deep Convolutional Neural Network Accelerators ». In: *2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT),* Hsinchu, Taiwan. Hsinchu, Taiwan: IEEE, 2019, pp. 1–4. ISBN: 978-1-7281-0656-4. DOI: 10.1109/VLSI-DAT.2019.8741950.

[63] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. « Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks ». In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (2016).

[64] Gilles Kahn. « The semantics of a simple language for parallel programming ». In: *Information processing* 74.471-475 (1974), pp. 15–28.

[65] Salita Sombatsiri, Yoshinori Takeuchi, and Masaharu Imai. « An efficient performance estimation method for configurable multi-layer bus-based SoC ». In: *Information and Media Technologies* 10.2 (2015), pp. 192–203.

[66] Jeronimo Castrillon, Rainer Leupers, and Gerd Ascheid. « MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs ». In: *IEEE Transactions on Industrial Informatics* 9.1 (2013), pp. 527–545. DOI: `10.1109/TII.2011.2173941`.

[67] Ralf Stemmer, Hai-Dang Vu, Sébastien Le Nours, Kim Grüttner, Sébastien Pillement, and Wolfgang Nebel. « A Measurement-Based Message-Level Timing Prediction Approach for Data-Dependent SDFGs on Tile-Based Heterogeneous MPSoCs ». In: *Applied Sciences* (2021).

[68] Hai-Dang Vu. « Fast and Accurate Performance Models for Probabilistic Timing Analysis of SDFGs on MPSoCs ». PhD thesis. Université de Nantes, 2021.

[69] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. « Approximate probabilistic model checking ». In: *Verification, Model Checking, and Abstract Interpretation: 5th International Conference, VMCAI 2004 Venice, Italy, January 11-13, 2004 Proceedings 5*. Springer. 2004, pp. 73–84.

[70] Ayoub Nouri, Marius Bozga, Anca Molnos, Axel Legay, and Saddek Bensalem. « Building faithful high-level models and performance evaluation of manycore embedded systems ». In: *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)* (Oct. 2014), pp. 209–218. DOI: `10.1109/MEMCOD.2014.6961864`.

[71] Sergio Mazzola, Thomas Benz, Björn Forsberg, and Luca Benini. « A Data-Driven Approach to Lightweight DVFS-Aware Counter-Based Power Modeling for Heterogeneous Platforms ». In: *Embedded Computer Systems: Architectures, Modeling, and Simulation: 22nd International Conference, SAMOS 2022, Samos, Greece, July 3–7, 2022, Proceedings*. 2022.

[72] Andy D. Pimentel. « Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration ». In: *IEEE Design & Test* 34.1 (2017), pp. 77–90. DOI: `10.1109/MDAT.2016.2626445`.

[73] A. H. Land and A. G. Doig. « An Automatic Method of Solving Discrete Programming Problems ». In: *Econometrica* 28.3 (1960), pp. 497–520. DOI: `https://doi.org/10.2307%2F1910129`. (Visited on 10/01/2023).

[74] Shobana Padmanabhan, Yixin Chen, and Roger D. Chamberlain. « Optimal design-space exploration of streaming applications ». In: *ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*. 2011, pp. 227–230. DOI: `10.1109/ASAP.2011.6043274`.

[75] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[76] Ralf Niemann and Peter Marwedel. « An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming ». In: *Des. Autom. Embedded Syst.* 2.2 (Mar. 1997), pp. 165–193. ISSN: 0929-5585. DOI: 10.1023/A:1008832202436.

[77] John H. Holland. « Genetic Algorithms ». In: *Scientific American* 267.1 (1992), pp. 66–73. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24939139 (visited on 10/02/2023).

[78] Fred Glover. « Future paths for integer programming and links to artificial intelligence ». In: *Computers & Operations Research* 13.5 (1986). Applications of Integer Programming, pp. 533–549. ISSN: 0305-0548. DOI: https://doi.org/10.1016/0305-0548(86)90048-1. URL: https://www.sciencedirect.com/science/article/pii/0305054886900481.

[79] LM Rasdi Rere, Mohamad Ivan Fanany, and Aniati Murni Arymurthy. « Simulated annealing algorithm for deep learning ». In: *Procedia Computer Science* 72 (2015), pp. 137–144.

[80] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. « Ant colony optimization ». In: *IEEE computational intelligence magazine* 1.4 (2006), pp. 28–39.

[81] Fabrizio Ferrandi, Pier Luca Lanzi, Christian Pilato, Donatella Sciuto, and Antonino Tumeo. « Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems ». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.6 (2010), pp. 911–924. DOI: 10.1109/TCAD.2010.2048354.

[82] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. « The irace package: Iterated racing for automatic algorithm configuration ». In: *Operations Research Perspectives* 3 (2016), pp. 43–58.

[83] L. Steiner, G. Delazeri, I. Prando Da Silva, M. Jung, and N. Wehn. « Automatic DRAM Subsystem Configuration with irace ». In: *International Conference on High-Performance and Embedded Architectures and Compilers 2020 (HiPEAC), Workshop on: Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO).* 2023.

[84] Linyan Mei, Pouya Houshmand, Vikram Jain, Sebastian Giraldo, and Marian Verhelst. « ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators ». In: *IEEE Transactions on Computers* (2021).

[85] Daniel Luenemann, Maher Fakih, and Kim Gruettner. « Capturing Neural-Networks as Synchronous Dataflow Graphs ». In: *MBMV 2020 - Methods and Description Languages for Modelling and Verification of Circuits and Systems; GMM/ITG/GI-Workshop.* Stuttgart, Germany: VDE, 2020, pp. 1–10.

[86] Mohammad Hosseinabady and Jose Luis Nunez-Yanez. « Run-Time Power Gating in Hybrid ARM-FPGA Devices ». In: *2014 24th International Conference on Field Programmable Logic and Applications (FPL).* 2014, pp. 1–6. DOI: 10.1109/FPL.2014.6927503.

[87] *UltraScale™ Architecture and Product Data Sheet: Overview (DS890 v4.4.1).* Last accessed: 03.10.2023. July 2023. URL: https://docs.xilinx.com/v/u/en-US/ds890-ultrascale-overview.

[88] *Rohde & Schwarz HMC8012 Digital Multimeter User Manual.* Last accessed: 03.10.2023. URL: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/pdm/cl_manuals/user_manual/5800_4505_01/HMC8012_UserManual_de_en_06.pdf.

[89] Behzad Salami, Erhan Baturay Onural, Ismail Emir Yuksel, Fahrettin Koc, Oguz Ergin, Adrian Cristal Kestelman, Osman Unsal, Hamid Sarbazi-Azad, and Onur Mutlu. « An Experimental Study of Reduced-Voltage Operation in Modern FPGAs for Neural Network Acceleration ». In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* 2020.

[90] Anup Kumar Sultania, Chun Zhang, Darshak Kumarpal Gandhi, Fan Zhang, Anup Kumar Sultania, Chun Zhang, Darshak Kumarpal Gandhi, and Fan Zhang. « Designing with Xilinx® FPGAs: Using Vivado ». In: ed. by Sanjay Churiwala and Sanjay Churiwala. Springer International Publishing, 2017. Chap. Power Analysis and Optimization, pp. 177–187.

[91] Christof Schlaak, Maher Fakih, and Ralf Stemmer. « Power and Execution Time Measurement Methodology for SDF Applications on FPGA-based MPSoCs ». In: *International Workshop on High Performance Energy Efficient Embedded Systems (HIP3ES)* (Jan. 2017). arXiv: 1701.03709 [cs.DC]. URL: http://arxiv.org/abs/1701.03709.

[92] *Xilinx Integrated Logic Analyzer v2.0 Data Sheet (DS875).* Last accessed: 03.10.2023. July 2012. URL: https://docs.xilinx.com/v/u/en-US/ds875-ila.

[93] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. « Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark ». In: *International Joint Conference on Neural Networks.* 2013.

[94] Oriane Thiery and Jules Bouton. « Réseau CNN sur MicroBlaze ». MA thesis. Polytech'Nantes - Graduate School of Engineering of Nantes Université, 2022.

[95] Steffen Nissen. *Implementation of a Fast Artificial Neural Network library (FANN).* Tech. rep. Last accessed: 10.10.2023, commit: 8409b42. Department of Computer Science, University of Copenhagen (DIKU), Dec. 2003. URL: https://github.com/libfann/fann.

[96] Daniel Gutierrez-Galan, Juan Pedro Dominguez-Morales, Elena Cerezuela-Escudero, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Manuel Rivas Perez, Manuel Dominguez-Morales, Angel Jimenez-Fernandez, and Alejandro Linares-Barranco. « Embedded neural network for real-time animal behavior classification ». In: *Neurocomputing* 272 (2018), pp. 17–26. DOI: 10.1016/j.neucom.2017.03.090.

[97] Nico Zengeler, Thomas Kopinski, and Uwe Handmann. « Hand Gesture Recognition in Automotive Human–Machine Interaction Using Depth Cameras ». In: *Sensors* 19.1 (2019). ISSN: 1424-8220. DOI: 10.3390/s19010059.

[98] Juan Borrego-Carazo, David Castells-Rufas, Ernesto Biempica, and Jordi Carrabina. « Resource-Constrained Machine Learning for ADAS: A Systematic Review ». In: *IEEE Access* 8 (2020), pp. 40573–40598. DOI: 10.1109/ACCESS.2020.2976513.

[99] Blair Newman. « Neuton TinyML: Automatic Design of Ultra-tiny ML Models ». In: *tinyML EMEA Innovation Forum.* Last accessed: 28.09.2023. June 2023. URL: `https://cms.tinyml.org/wp-content/uploads/ew2023/tinyML-EMEA_Blair-Newman.pdf`.

[100] Hai-Dang Vu, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, and Kim Grüttner. « A Fast Yet Accurate Message-level Communication Bus Model for Timing Prediction of SDFGs on MPSoC ». In: *Asia and South Pacific Design Automation Conference_ASP-DAC 2021 (Virtual Conference).* ASP-DAC 2021. Tokyo, Japan, Jan. 2021, p. 1183. URL: `https://hal.archives-ouvertes.fr/hal-02938566`.

[101] David M. Bressoud. « The Fundamental Theorem of Calculus ». In: *Second Year Calculus: From Celestial Mechanics to Special Relativity.* New York, NY: Springer New York, 1991. Chap. 10, pp. 279–332. ISBN: 978-1-4612-0959-1. DOI: `10.1007/978-1-4612-0959-1_10`.

[102] David A Fabry and Achintya K Bhowmik. « Improving speech understanding and monitoring health with hearing aids using artificial intelligence and embedded sensors ». In: *Seminars in Hearing.* Vol. 42. 03. Thieme Medical Publishers, Inc. 333 Seventh Avenue, 18th Floor, New York, NY . . . 2021, pp. 295–308.

[103] Charnpreet Kaur and Urvashi Garg. « Artificial intelligence techniques for cancer detection in medical image processing: A review ». In: *Materials Today: Proceedings* 81 (2023). International Virtual Conference on Sustainable Materials (IVCSM-2k20), pp. 806–809. ISSN: 2214-7853. DOI: `https://doi.org/10.1016/j.matpr.2021.04.241`. URL: `https://www.sciencedirect.com/science/article/pii/S2214785321031618`.

[104] Tanha Talaviya, Dhara Shah, Nivedita Patel, Hiteshri Yagnik, and Manan Shah. « Implementation of artificial intelligence in agriculture for optimisation of irrigation and application of pesticides and herbicides ». In: *Artificial Intelligence in Agriculture* 4 (2020), pp. 58–73. ISSN: 2589-7217. DOI: `https://doi.org/10.1016/j.aiia.2020.04.002`. URL: `https://www.sciencedirect.com/science/article/pii/S258972172030012X`.

[105] Oliver Klemp, Maher Fakih, Kim Grüttner, Ralf Stemmer, and Wolfgang Nebel. « Experimental Evaluation of Scenario Aware Synchronous Data Flow Based Power Management ». In: *Proceedings of the International Conference on Omni-Layer Intelligent Systems.* COINS '19. Crete, Greece: Association for Computing Machinery, 2019, pp. 80–85. ISBN: 9781450366403. DOI: `10.1145/3312614.3312634`. URL: `https://doi.org/10.1145/3312614.3312634`.

[106] Abdallah Moujahid, Mounir ElAraki Tantaoui, Manolo Dulva Hina, Assia Soukane, Andrea Ortalda, Ahmed ElKhadimi, and Amar Ramdane-Cherif. « Machine Learning Techniques in ADAS: A Review ». In: *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE).* 2018, pp. 235–242. DOI: `10.1109/ICACCE.2018.8441758`.

[107] Quentin Dariol, Sebastien Le Nours, Domenik Helms, Ralf Stemmer, Sebastien Pillement, and Kim Grüttner. « Fast Yet Accurate Timing and Power Prediction of Artificial Neural Networks Deployed on Clock-Gated Multi-Core Platforms ». In: RAPIDO '23. Toulouse, France: Association for Computing Machinery, 2023, pp. 79–86. DOI: `10.1145/3579170.3579263`. URL: `https://doi.org/10.1145/3579170.3579263`.

[108] Quentin Dariol, Sebastien Le Nours, Sebastien Pillement, Ralf Stemmer, Domenik Helms, and Kim Grüttner. « A Hybrid Performance Prediction Approach for Fully-Connected Artificial Neural Networks on Multi-core Platforms ». In: *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2022).* Ed. by Alex Orailoglu, Marc Reichenbach, and Matthias Jung. Springer International Publishing, 2022, pp. 250–263. DOI: `10.1007/978-3-031-15074-6_16`.

[109] Quentin Dariol, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, Domenik Helms, and Kim Grüttner. « Early Performance and Energy Prediction of Neural Networks Deployed on Multi-Core Platforms ». In: *29° Colloque sur le traitement du signal et des images.* 2023-1144. Grenoble: GRETSI - Groupe de Recherche en Traitement du Signal et des Images, Aug. 2023, p. 309–312. URL: `#https://elib.dlr.de/196733/#`.

[110] Quentin Dariol, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, Kim Grüttner, and Domenik Helms. « Hybrid Performance Prediction Models for Fully-Connected Neural Networks on MPSoC ». In: *16ème Colloque National du GDR SOC2.* June 2022. URL: `https://hal.science/hal-03758026`.

[111] Quentin Dariol, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, Kim Grüttner, and Domenik HELMS. *A Measurement-based Performance Evaluation Framework for Neural Networks on MPSoCs.* 15ème Colloque National du GDR SOC2. Poster. June 2021. URL: `https://hal.archives-ouvertes.fr/hal-03248152`.

[112] Quentin Dariol, Sebastien Le Nours, Sebastien Pillement, Kim Grüttner, Domenik Helms, and Ralf Stemmer. *Setup of an Experimental Framework for Performance Modeling and Prediction of Embedded Multicore AI Architectures.* Tech. rep. Nantes University, IETR UMR CNRS 6164, France and Deutsches Zentrum für Luft und Raumfahrt (DLR), Germany, 2022.

Appendices

A Considered NN clusterings and mappings to validate our models

Table A.1 – Considered clusterings of MLP1 for the validation of the models. The resulting clusterings are illustrated in the Figure A.1.

Clustering name	Number of clusters (actors) per layer		Actor number	Comm. channel number
	Hidden Layer	Output Layer		
C1	1	1	2	3
C3	3	3	7	16
C7	7	7	15	64

Table A.2 – Considered mappings of MLP1 for the validation of the models. The considered clusterings identified based on "C_ID" are provided in Table A.1. "#T" is the number of tiles for the mapping. "M_ID" is the identifier of the mapping and "P" stands for polling-based communications whereas "I" stands for interrupt-based communications.

<table>
<tr><th rowspan="2">C_ID</th><th rowspan="2">#T</th><th colspan="2">M_ID</th><th colspan="15">Mapping of actors on platform</th></tr>
<tr><th>P</th><th>I</th><th colspan="7">Hidden layer</th><th colspan="7">Output layer</th><th>Decoder</th></tr>
<tr><td rowspan="2">MLP1
C1</td><td>1</td><td>1</td><td>8</td><td colspan="7">0</td><td colspan="7">0</td><td>/</td></tr>
<tr><td>3</td><td>2</td><td>9</td><td colspan="7">0</td><td colspan="7">1</td><td>/</td></tr>
<tr><td rowspan="3">MLP1
C3</td><td>1</td><td>3</td><td>10</td><td colspan="2">0</td><td colspan="2">0</td><td colspan="3">0</td><td colspan="2">0</td><td colspan="2">0</td><td colspan="3">0</td><td>0</td></tr>
<tr><td>3</td><td>4</td><td>11</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="3">2</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="3">2</td><td>0</td></tr>
<tr><td>7</td><td>5</td><td>12</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="3">2</td><td colspan="2">3</td><td colspan="2">4</td><td colspan="3">5</td><td>6</td></tr>
<tr><td rowspan="2">MLP1
C7</td><td>1</td><td>6</td><td>13</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>7</td><td>7</td><td>14</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>0</td></tr>
</table>

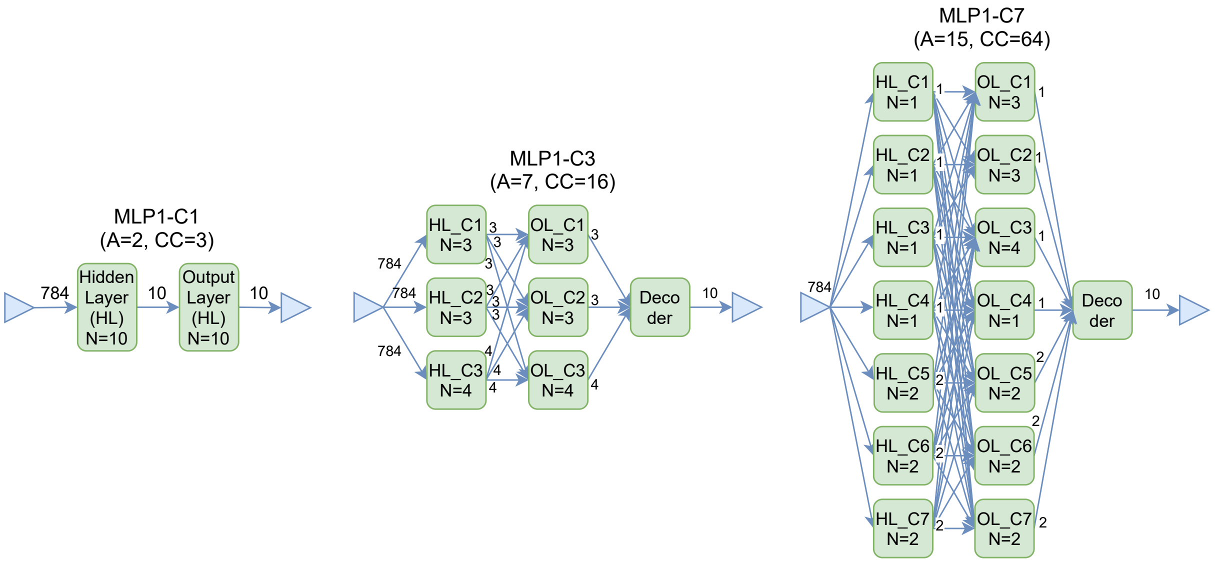


Figure A.1 – Illustration of the different clusterings considered for the validation of the models for MLP1. "A" corresponds to the number of actors in the clustering and "CC" to the number of communication channels. Note: due to the density of the communication channels on the clustering MLP1-C7 the number of tokens have been indicated only once for each actor (all communication channels issued by the same actor have the same number of tokens).

Table A.3 – Considered clusterings of MLP2 for the validation of the models.

Clustering name	Number of clusters (actors) per layer			Actor number	Comm. channel number
	Hidden Layer 1	Hidden Layer 2	Output Layer		
C1	1	1	1	3	4
C3	3	3	3	10	25
C7	7	7	7	22	113

Table A.4 – Considered mappings of MLP2 for the validation of the models. The considered clusterings identified based on "C_ID" are provided in Table A.3. "#T" is the number of tiles for the mapping. "M_ID" is the identifier of the mapping and "P" stands for polling-based communications whereas "I" stands for interrupt-based communications.

C_ID	#T	M_ID		Mapping of actors on platform																					
		P	I	Hidden Layer 1							Hidden Layer 2							Output Layer							Decoder
MLP2 C1	1	15	22	0							0							0							/
	3	16	23	0							1							2							/
MLP2 C3	1	17	24	0		0		0			0		0		0			0		0		0			0
	3	18	25	0		1		2			0		1		2			0		1		2			0
	7	19	26	0		1		2			3		4		5			3		4		5			6
MLP2 C7	1	20	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	21	28	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0

Table A.5 – Considered clusterings of MLP3 for the validation of the models.

Clustering name	Number of clusters (actors) per layer			Actor number	Comm. channel number
	Hidden Layer 1	Hidden Layer 2	Output Layer		
C2	2	2	2	6	13
C4	4	4	4	12	41
C6	6	6	6	18	85

Table A.6 – Considered mappings of MLP3 for the validation of the models. The considered clusterings identified based on "C_ID" are provided in Table A.5. "#T" is the number of tiles for the mapping. "M_ID" is the identifier of the mapping and "P" stands for polling-based communications whereas "I" stands for interrupt-based communications

C_ID	T#	M_ID		Mapping of actors on platform																		
		P	I	Hidden Layer 1						Hidden Layer 2						Output Layer						Decoder
MLP3 C2	1	29	36	0			0			0			0			0			0			NA
	2	30	37	0			1			0			1			0			1			NA
	7	31	38	0			1			2			3			4			5			6
MLP3 C4	1	32	39	0		0		0	0	0		0		0	0	0		0		0	0	0
	4	33	40	0		1		2	3	0		1		2	3	0		1		2	3	0
MLP3 C6	1	34	41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	35	42	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0

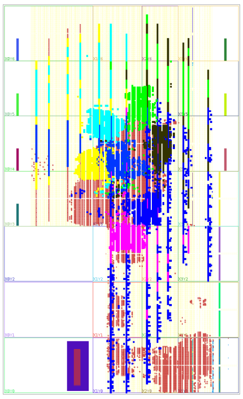
Table A.7 – Considered clusterings of the CNN for the validation of the models.

Clustering name	Number of clusters (actors) per layer				Actor number	Comm. channel number
	Convo lution	Pooling	Dense1	Dense2		
G1	1	1	1	1	4	5
G2	5	1	1	1	8	13

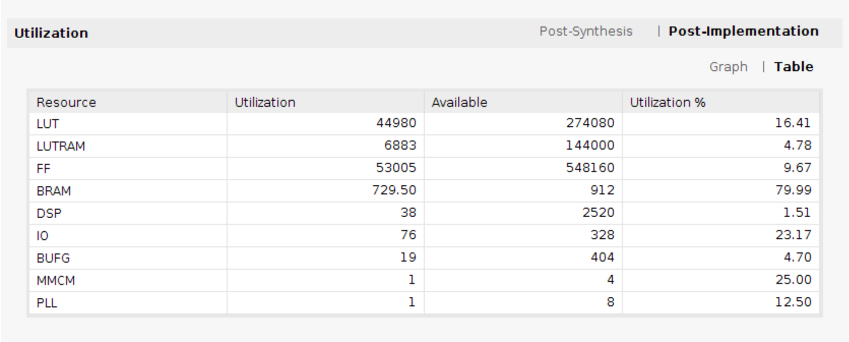
Table A.8 – Considered mappings of the CNN for the validation of the models. The considered clusterings identified based on "C_ID" are provided in Table A.7. "#T" is the number of tiles for the mapping. "M_ID" is the identifier of the mapping and "P" stands for polling-based communications whereas "I" stands for interrupt-based communications.

C_ID	#T	M_ID		Mapping of actors on platform							
		P	I	Convolution					Pooling	Dense 1	Dense 2
CNN1 G1	1	43	49	0					0	0	0
	2	44	50	1					0	0	0
	4	45	51	1					2	0	3
CNN1 G2	1	46	52	0	0	0	0	0	0	0	0
	5	47	53	0	1	2	3	4	0	0	0
	7	48	54	1	2	3	4	5	6	0	0

B Place and route and utilization results of the different prototype platforms



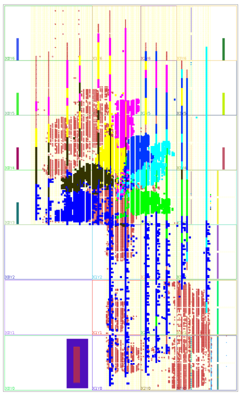
(a) $\mathcal{P}_{7,\,\mathrm{I}}$ - floorplan



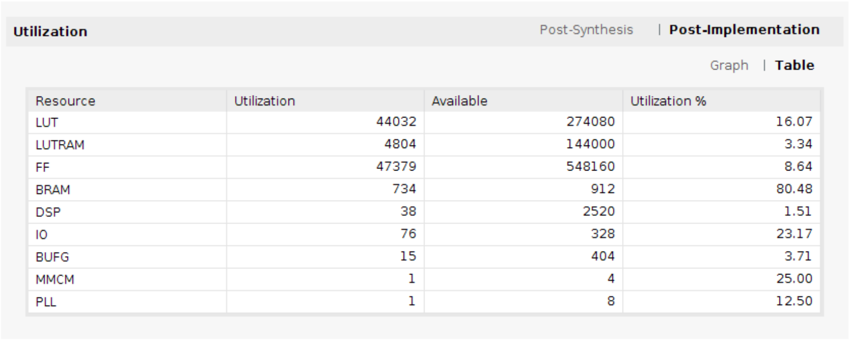
Utilization

Resource	Utilization	Available	Utilization %
LUT	44980	274080	16.41
LUTRAM	6883	144000	4.78
FF	53005	548160	9.67
BRAM	729.50	912	79.99
DSP	38	2520	1.51
IO	76	328	23.17
BUFG	19	404	4.70
MMCM	1	4	25.00
PLL	1	8	12.50

(b) $\mathcal{P}_{7,\,\mathrm{I}}$ - utilization



(c) $\mathcal{P}_{7,\,\mathrm{P}}$ - floorplan



Utilization

Resource	Utilization	Available	Utilization %
LUT	44032	274080	16.07
LUTRAM	4804	144000	3.34
FF	47379	548160	8.64
BRAM	734	912	80.48
DSP	38	2520	1.51
IO	76	328	23.17
BUFG	15	404	3.71
MMCM	1	4	25.00
PLL	1	8	12.50

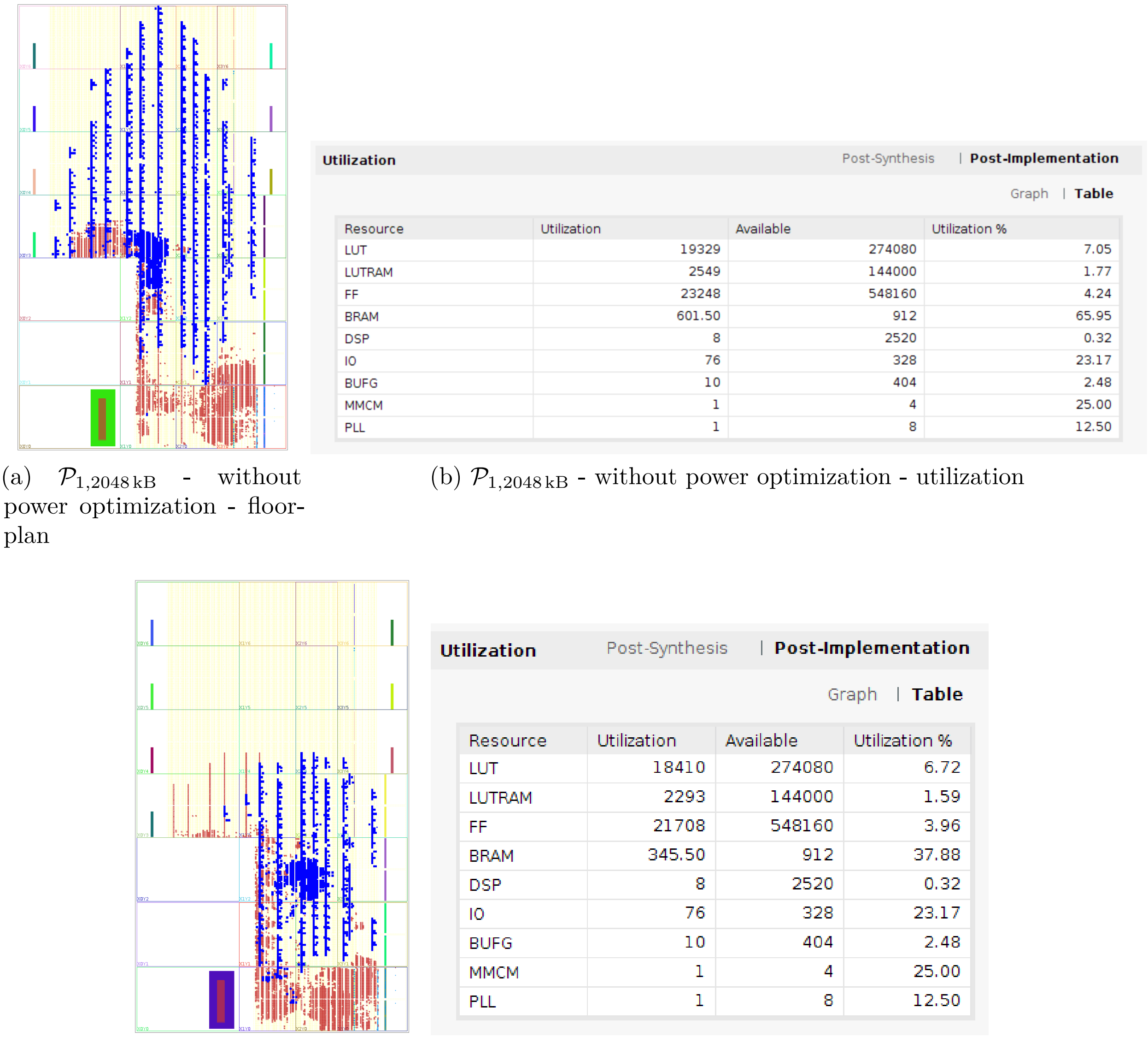
(d) $\mathcal{P}_{7,\,\mathrm{P}}$ - utilization

Legend

🟦 : Tile0	🟨 : Tile4
🟩 : Tile1	🟦 : Tile5
🟪 : Tile2	🟫 : Tile6
🟦 : Tile3	🟥 : Remainder of the platform

(e) Legend (color map) of the provided placement & route plans

Figure B.2 – Place and route and utilization results of the different prototype platforms we considered for the evaluation of the power modeling flow. These platforms are used to evaluate scalability of the power modeling flow in consideration of multi-core platforms with varying sizes (in regards to number of tiles and private memory size).



Utilization | Post-Synthesis | Post-Implementation

Graph | Table

Resource	Utilization	Available	Utilization %
LUT	19329	274080	7.05
LUTRAM	2549	144000	1.77
FF	23248	548160	4.24
BRAM	601.50	912	65.95
DSP	8	2520	0.32
IO	76	328	23.17
BUFG	10	404	2.48
MMCM	1	4	25.00
PLL	1	8	12.50

(a) $\mathcal{P}_{1,2048\,\mathrm{kB}}$ - without power optimization - floorplan

(b) $\mathcal{P}_{1,2048\,\mathrm{kB}}$ - without power optimization - utilization

Utilization | Post-Synthesis | Post-Implementation

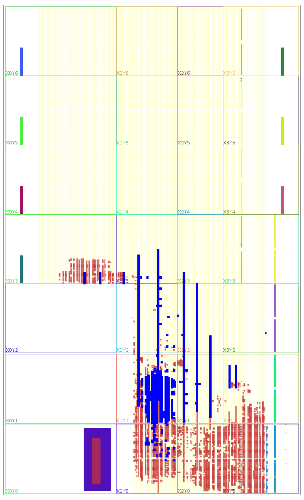
Graph | Table

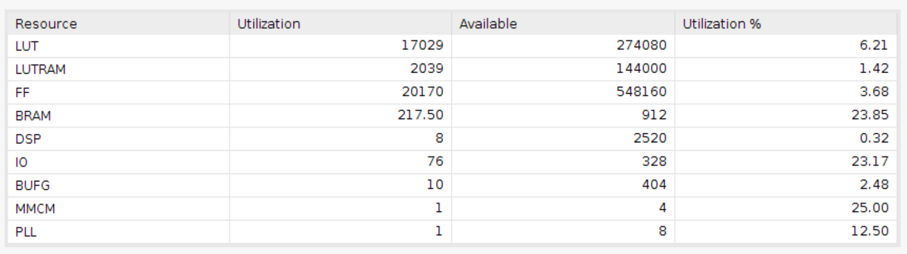
Resource	Utilization	Available	Utilization %
LUT	18410	274080	6.72
LUTRAM	2293	144000	1.59
FF	21708	548160	3.96
BRAM	345.50	912	37.88
DSP	8	2520	0.32
IO	76	328	23.17
BUFG	10	404	2.48
MMCM	1	4	25.00
PLL	1	8	12.50

(c) $\mathcal{P}_{1,1024\,\mathrm{kB}}$ - without power optimization - floorplan

(d) $\mathcal{P}_{1,1024\,\mathrm{kB}}$ - without power optimization - utilization

Figure B.3 – See Figure B.2 caption and legend.

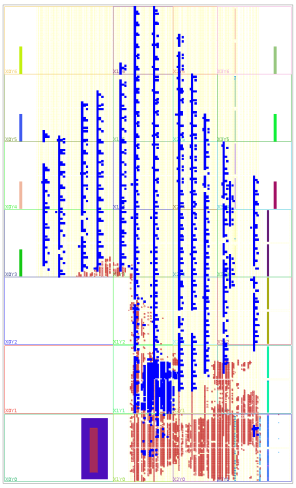


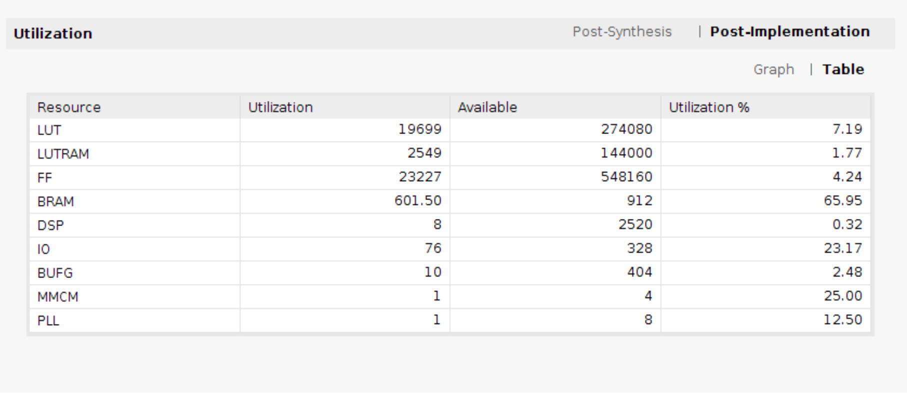


Resource	Utilization	Available	Utilization %
LUT	17029	274080	6.21
LUTRAM	2039	144000	1.42
FF	20170	548160	3.68
BRAM	217.50	912	23.85
DSP	8	2520	0.32
IO	76	328	23.17
BUFG	10	404	2.48
MMCM	1	4	25.00
PLL	1	8	12.50

(a) $\mathcal{P}_{1,512\,\mathrm{kB}}$ - without power optimization - floorplan

(b) $\mathcal{P}_{1,512\,\mathrm{kB}}$ - without power optimization - utilization





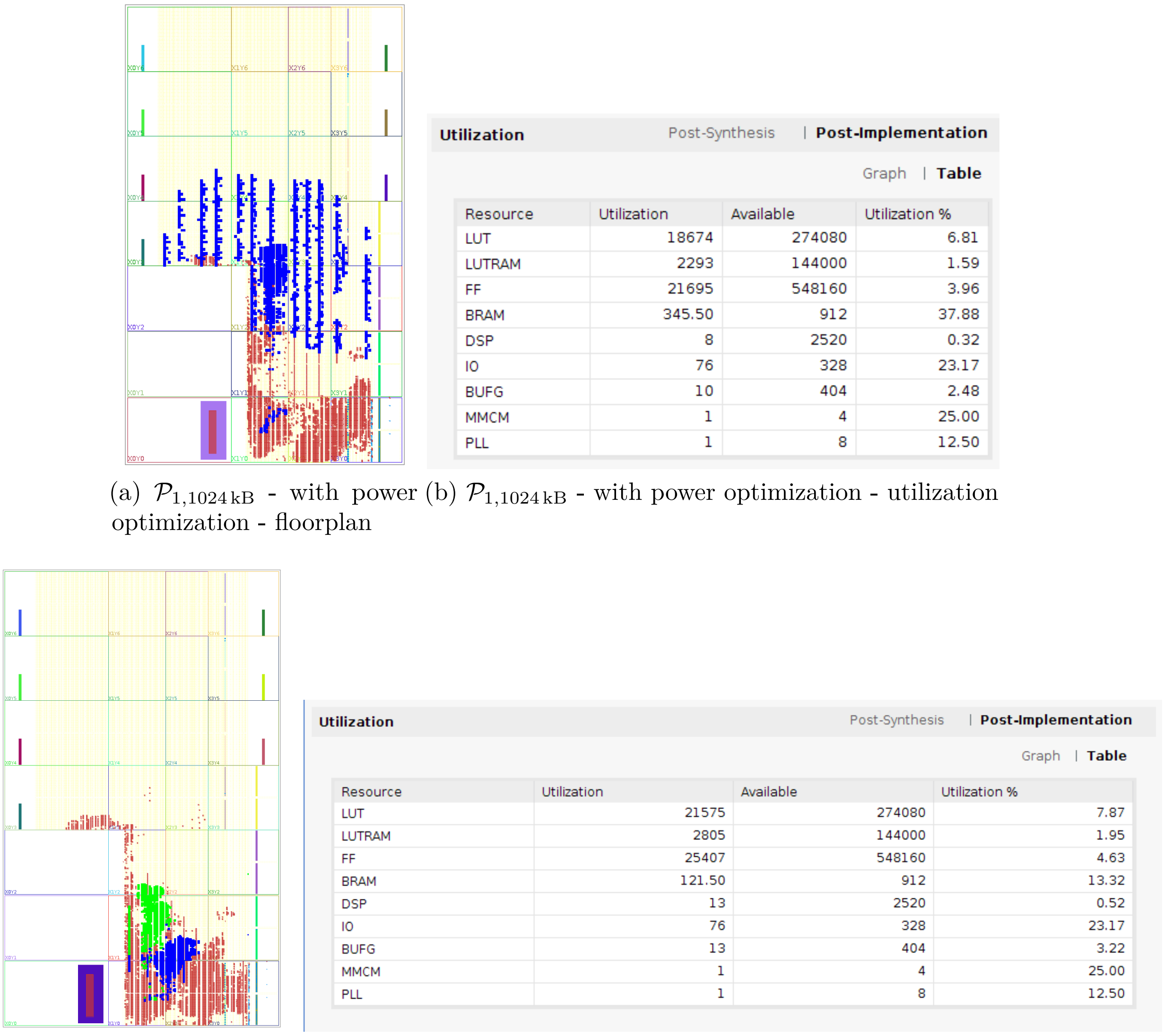
Utilization
Post-Synthesis | Post-Implementation
Graph | Table

Resource	Utilization	Available	Utilization %
LUT	19699	274080	7.19
LUTRAM	2549	144000	1.77
FF	23227	548160	4.24
BRAM	601.50	912	65.95
DSP	8	2520	0.32
IO	76	328	23.17
BUFG	10	404	2.48
MMCM	1	4	25.00
PLL	1	8	12.50

(c) $\mathcal{P}_{1,2048\,\mathrm{kB}}$ - with power optimization - floorplan

(d) $\mathcal{P}_{1,2048\,\mathrm{kB}}$ - with power optimization - utilization

Figure B.4 – See Figure B.2 caption and legend.



Utilization — Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	18674	274080	6.81
LUTRAM	2293	144000	1.59
FF	21695	548160	3.96
BRAM	345.50	912	37.88
DSP	8	2520	0.32
IO	76	328	23.17
BUFG	10	404	2.48
MMCM	1	4	25.00
PLL	1	8	12.50

(a) $\mathcal{P}_{1,1024\,\mathrm{kB}}$ - with power optimization - floorplan

(b) $\mathcal{P}_{1,1024\,\mathrm{kB}}$ - with power optimization - utilization

Utilization — Post-Synthesis | **Post-Implementation**

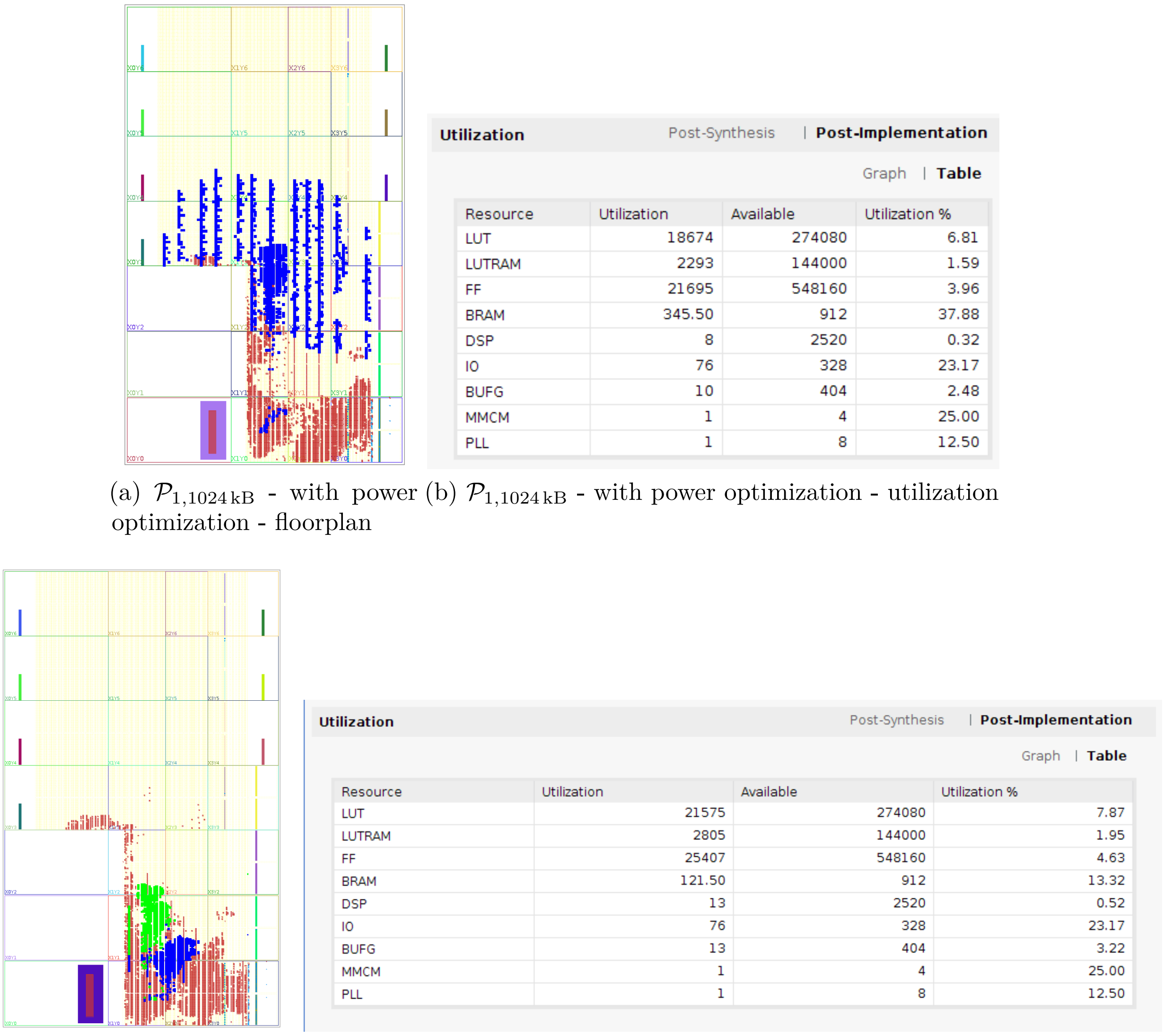
Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	21575	274080	7.87
LUTRAM	2805	144000	1.95
FF	25407	548160	4.63
BRAM	121.50	912	13.32
DSP	13	2520	0.52
IO	76	328	23.17
BUFG	13	404	3.22
MMCM	1	4	25.00
PLL	1	8	12.50

(c) $\mathcal{P}_2$ - floorplan

(d) $\mathcal{P}_2$ - utilization

Figure B.5 – See Figure B.2 caption and legend.



Utilization   Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	18674	274080	6.81
LUTRAM	2293	144000	1.59
FF	21695	548160	3.96
BRAM	345.50	912	37.88
DSP	8	2520	0.32
IO	76	328	23.17
BUFG	10	404	2.48
MMCM	1	4	25.00
PLL	1	8	12.50

(a) $\mathcal{P}_{1,1024\,\text{kB}}$ - with power optimization - floorplan

(b) $\mathcal{P}_{1,1024\,\text{kB}}$ - with power optimization - utilization

Utilization   Post-Synthesis | **Post-Implementation**

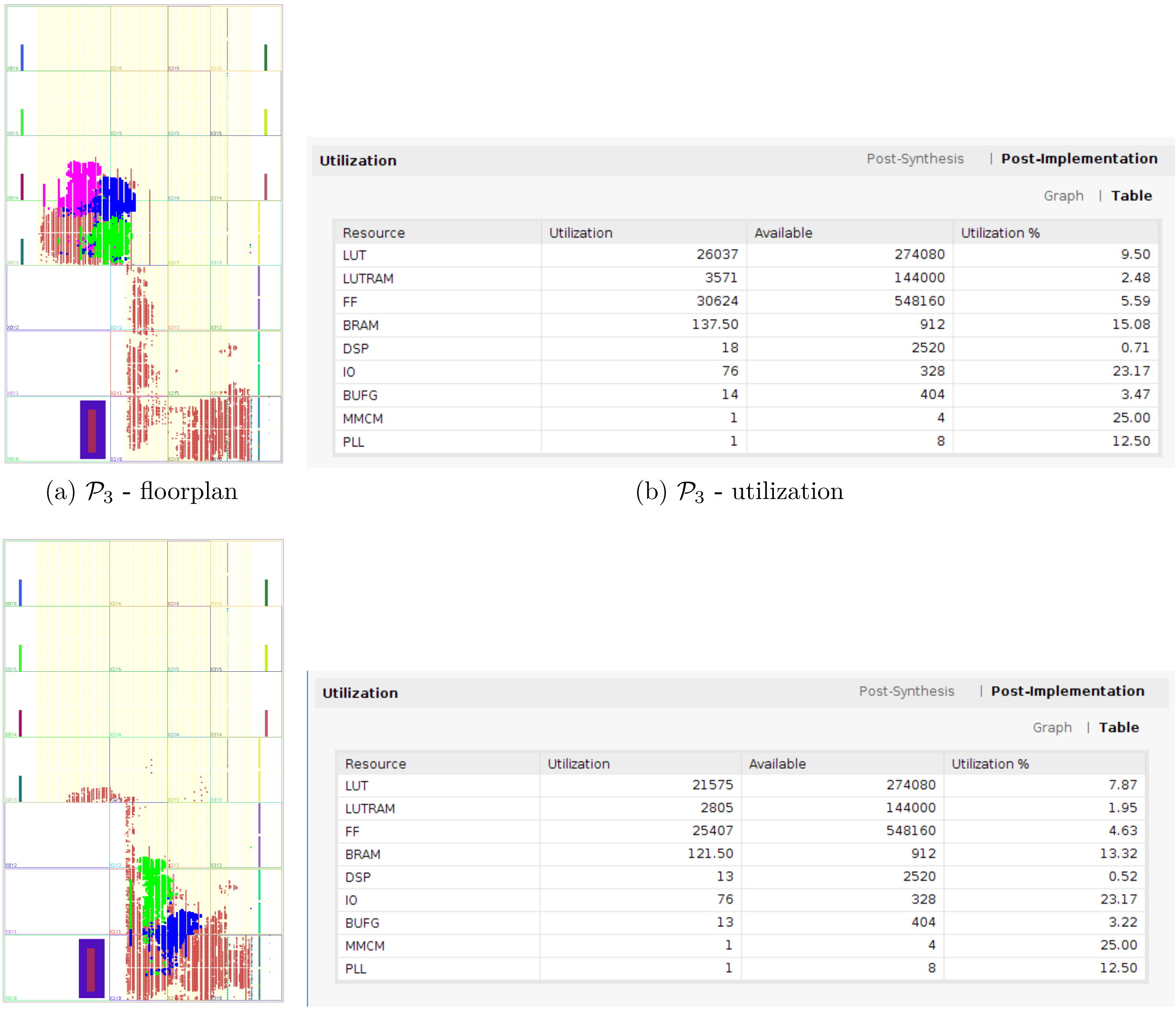
Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	21575	274080	7.87
LUTRAM	2805	144000	1.95
FF	25407	548160	4.63
BRAM	121.50	912	13.32
DSP	13	2520	0.52
IO	76	328	23.17
BUFG	13	404	3.22
MMCM	1	4	25.00
PLL	1	8	12.50

(c) $\mathcal{P}_2$ - floorplan

(d) $\mathcal{P}_2$ - utilization

Figure B.6 – See Figure B.2 caption and legend.



(a) $\mathcal{P}_3$ - floorplan

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	26037	274080	9.50
LUTRAM	3571	144000	2.48
FF	30624	548160	5.59
BRAM	137.50	912	15.08
DSP	18	2520	0.71
IO	76	328	23.17
BUFG	14	404	3.47
MMCM	1	4	25.00
PLL	1	8	12.50

(b) $\mathcal{P}_3$ - utilization

(c) $\mathcal{P}_5$ - floorplan

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	21575	274080	7.87
LUTRAM	2805	144000	1.95
FF	25407	548160	4.63
BRAM	121.50	912	13.32
DSP	13	2520	0.52
IO	76	328	23.17
BUFG	13	404	3.22
MMCM	1	4	25.00
PLL	1	8	12.50

(d) $\mathcal{P}_5$ - utilization

Figure B.7 – See Figure B.2 caption and legend.

C Model of private memory size of tile

To help and propose accurate power and energy prediction with scalability in regards to the dimensions of multi-core platforms, we propose a coarsed-grained private memory size model. This model is characterized by retrieving and analyzing the utilization of private memory based on executed code for the considered NN mappings presented in Section A. The model is presented in Table C.9.

Table C.9 – Proposed model for private memory size needed for tile execution

Tile private memory sections (in order)	**Actual content**	**Memory size model (bytes)**
.vectors	SW/HW exceptions management, reset, etc.	128
.text	Instructions	$8192 + 512 \cdot N_{\text{actor}}$
.init, .fini, .ctors, .dtors, .rodata, .sdata2	/	Marginal, neglected
.data	Initialized global variables: Weights and input image	🅐
.sdata, .sbss	/	Marginal, neglected
.bss	Uninitialized global variables	256
.heap	Dynamically allocated space	2048
.stack	Local variables used inside functions, SDF token_buffers and channels	🅑

Equation C.1 gives the model used to approximate the size of the .data section of a tile (🅐 in the table):

$$\text{🅐} : B_{l=0} + \sum_{a=1}^{A} \lambda_a W_a \tag{C.1}$$

$$\text{with} \quad W_a = \begin{cases} 4 \cdot N_{\text{neuron},a} \cdot (N_{\text{inputs}} + 1) & \text{if } a \text{ is an actor from a dense layer} \\ 4 \cdot K_a \cdot (F_{\text{h}} \cdot F_{\text{w}} + N_{\text{inputs}}) & \text{if } a \text{ is an actor from a convolution layer} \end{cases}$$

$$\text{and} \quad \lambda_a = \begin{cases} 1 & \text{if actor } a \text{ is mapped on the considered tile} \\ 0 & \text{otherwise} \end{cases}$$

In this equation, $B_{l=0}$ denotes the size of the input layer (i.e. the size of the input image). A designates the total number of actors. λ_a is a variable used to indicate if actor a is mapped on the considered tile. W_a denotes the memory footprint of the weights needed to execute the actor a. N_{inputs} denotes the number of inputs of actor a. $N_{\text{neuron},a}$ is the number of neurons inside the actor a, if a is issued from a dense layer. K_a denotes the number of convolution kernels inside the actor a, if a is issued from a convolution layer. F_h and F_h respectively denotes the height and width of the convolution kernel, if a is issued from a convolution layer. The multiplication by 4 is due to the encoding of all these values as floats on 32 bits i.e. 4 bytes. Equation C.2 gives the model used to approximate the size of the .stack section (🅑 in the table):

$$\text{🅑} : 4 \cdot \left(2 \cdot N_{\text{channels}} + \sum_{l=0}^{L-1} B_l \right) \tag{C.2}$$

In this equation, N_{channels} is the number of channels in the SDF graph. The formula for N_{channels} is presented in Section III.2.2. Channels are encoded as a structure containing two elements: first address and total buffer size, so N_{channels} is multiplied by 2. B_l denotes the sizes of buffer l: it corresponds to the number of tokens exchanged between l and $l+1$, $l = 0$ corresponding to the input layer of the NN, and $l = L-1$ to the output layer. The multiplication by 4 is due to the encoding of all these values as floats (4 bytes).

The estimation using this model is rounded to the next higher power of 2, and this value is used as the private memory size for the considered tile. For example, if the considered tile processes the input image of the MNIST dataset (784 floats, so 3136 bytes), using a dense layer that is composed of 300 neurons, the estimated memory cost using the model is 979 184 B i.e. 956 kB. The next higher power of 2 is $2^{10} = 1024$ kB, which corresponds to the private memory size to implement for the tile.

COLLEGE DOCTORAL PAYS DE LA LOIRE
ED MATHEMATIQUES STIC

Nantes Université

Titre : Prédiction et optimisation des propriétés temporelles et de l'énergie des réseaux de neurones artificiels implémentés sur les plateformes multicœurs

Mot clés : Intelligence artificielle embarquée, conception au niveau système, prédiction des propriétés temporelles et de l'énergie

Résumé : Le besoin de mettre en oeuvre les Réseaux de Neurones artificiels (NNs) sur des plates-formes multicœurs embarquées est devenu fondamental. La prédiction des propriétés temporelles (temps d'inférence, latence, débit) et énergétiques au plus tôt dans le processus de conception est nécessaire pour trouver des solutions qui optimisent l'utilisation des ressources et respectent les contraintes imposées au système. Une difficulté majeure de cette modélisation vient de la nécessité de décrire correctement l'influence du partage de ressources (processeur, mémoire, bus de communication) au sein des plateformes multicoeurs. Dans cette thèse, nous présentons un flot complet de prédiction et d'optimisation des propriétés temporelles et de l'énergie qui combine plusieurs approches de modélisation. Ce flot conduit à optimiser l'occupation des ressources sans dégrader les performances des NNs mis en oeuvre. Les prédictions sont confrontées à des expérimentations sur cibles réelles. Les modèles proposés ont une précision de plus de 97% sur le temps et 93% sur l'énergie sur 54 mappings de 4 NNs, avec un temps de prédiction de 20s par mapping. Nous montrons comment utiliser les modèles pour explorer efficacement l'espace de conception et trouver des solutions optimisées qui satisfont les contraintes imposées au système.

Title: Early Timing and Energy Prediction and Optimization of Artificial Neural Networks on Multi-Core Platforms

Keywords: Embedded artificial intelligence, system level design, timing and energy prediction

Abstract: The need to implement artificial Neural Networks (NNs) on embedded multi-core platforms has become fundamental. Predicting timing properties (inference time, latency, throughput) and energy as early as possible in the design process is necessary to find solutions that optimize resource use and respect the constraints imposed on the system. A major modeling difficulty comes from the need to correctly describe the influence of resource sharing (processor, memory, communication bus) within multi-core platforms. In this thesis, we present a complete flow for predicting and optimizing timing properties and energy, combining several modeling approaches. This flow leads to optimized resource occupancy without degrading the performance of implemented NNs. Predictions are compared with measurements on real targets. The proposed models have an accuracy of over 97% on timing and 93% on energy for 54 mappings of 4 NNs, with a prediction time of 20s per mapping. We show how to use the models to efficiently explore the design space and find optimized solutions that satisfy the constraints imposed on the system.