Techniques d'apprentissage pour le contrôle adaptatif multi-niveaux du calcul distribué

Maxime Mirka

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

En SyAM – Systèmes Automatiques et Micro-Électroniques

École doctorale I2S – Information, Structures, Systèmes

Unité de recherche LIRMM

Techniques d'apprentissage pour le contrôle adaptatif multi-niveaux du calcul distribué

Présentée par Maxime MIRKA
Le 12 Novembre 2021

Sous la direction de Gilles SASSATELLI et Abdoulaye GAMATIE

Devant le jury composé de

François BERRY, Professeur, Institut Pascal – UCA/CNRS	Président du jury
Gilles SASSATELLI, Directeur de Recherche au CNRS, LIRMM - UM/CNRS	Directeur de thèse
Abdoulaye GAMATIE, Directeur de Recherche au CNRS, LIRMM - UM/CNRS	Co-directeur de thèse
Alberto BOSIO, Professeur, INL - Ecole Centrale Lyon - CNRS	Rapporteur
Maxime PELCAT, Maître de Conférences, INSA Rennes	Rapporteur
Maria MENDEZ REAL, Maître de Conférences, Polytech - Université de Nantes	Examinatrice

UNIVERSITÉ DE MONTPELLIER

Remerciements

Je souhaite tout d'abord remercier l'ensemble des membres du jury de thèse pour avoir accepté d'évaluer mes travaux. Je remercie Maxime PELCAT et Alberto BOSIO d'avoir accepté la fonction de rapporteur et de m'avoir ainsi permis de profiter de leurs précieux retours sur le manuscrit. Merci à Maria MENDEZ REAL pour l'honneur qu'elle me fait d'être dans mon jury de thèse. Je remercie également François BERRY pour avoir accepté de participer à mon jury de thèse et d'en assurer la présidence.

Je voudrais remercier grandement mes directeurs de thèse, Gilles SASSATELLI et Abdoulaye GAMATIE pour m'avoir guidé durant ces trois années de recherche. Leurs conseils et leur disponibilité ont été précieux au cours de l'élaboration de cette thèse. Vous avez toute ma gratitude.

Je tiens à remercier Maxime FRANCE-PILLOIS pour son aide et le temps qu'il a consacré à ma recherche.

Merci à Jan Moritz JOSEPH et Christophe PAUL pour nos échanges scientifiques enrichissants.

Il m'est impossible d'oublier Francesco DI GREGORIO et Guillaume DEVIC, collègues de bureau et partenaires de galères, pour toutes nos discussions autour d'un café.

Je remercie toutes les personnes avec qui j'ai partagé mes études et notamment ces années de thèse. Merci également à tous mes proches et amis.

Un remerciement particulier pour ma compagne, Sophie, qui m'a toujours soutenu. Merci pour avoir rendu ma vie en dehors du laboratoire des plus agréables.

Je voudrais remercier ma sœur, Sarah, pour son éternel soutient et son écoute dans les moments difficiles.

Enfin, mes derniers remerciements vont à mes parents. Merci pour votre soutien indéfectible et pour avoir toujours cru en moi. Vous m'avez toujours accompagné et encouragé dans l'ensemble de mes projets. Merci du fond du cœur.

Résumé

L'essor des techniques et algorithmes d'apprentissage (i.e. Machine Learning) et leur utilisation dans des domaines de plus en plus variés montrent des capacités souvent surprenantes – dans l'interprétation des données d'entrée et la capacité de construire des représentations abstraites pertinentes (e.g. apprentissage supervisé), mais aussi dans le contrôle dynamique de systèmes complexes (e.g. apprentissage par renforcement). L'optimisation de l'efficacité énergétique des systèmes de calcul est devenue un enjeu majeur. De la conception matérielle au contrôle logiciel, différents leviers existent pour agir sur l'exécution de calculs. Nous considérons dans cette thèse l'optimisation du calcul parallèle, s'exécutant sur des architectures complexes, du processeur multicoeur au cluster de calcul. Ces systèmes possèdent un nombre de paramètres de conception et de contrôle important, donnant lieu à un nombre de combinaisons souvent trop large pour pouvoir être considérées de façon exhaustive.

Ainsi, cette thèse a pour objectif de s'appuyer sur les techniques d'apprentissage automatique récentes, basées sur les réseaux de neurones, pour construire des solutions de contrôle et de conception de systèmes de calcul. Ces techniques peuvent considérer un ensemble significatif de paramètres afin de proposer des solutions optimales. Les techniques proposées ont vocation à être multi-niveaux et pourront à ce titre être appliquées à l'échelle d'un système embarqué ou de ses divers sous-composants mais aussi à l'échelle d'un système distribué, comme par exemple un cluster de calcul.

Des solutions prometteuses sont proposées selon deux axes de recherche distincts. Le premier axe s'adresse au contrôle dynamique du calcul parallèle. Il est question de l'optimisation en temps réel de l'efficacité énergétique d'un système exécutant une application parallélisée avec OpenMP, à l'aide, entre autres, d'un apprentissage par renforcement. Le deuxième axe concerne la conception de réseaux de communication optimisés. En effet, les réseaux de communication représentent une part non négligeable de la consommation énergétique des systèmes de calcul. Ainsi nous proposons un outil d'aide à la conception basé sur une IA générative, pour la génération de réseaux optimisés selon des critères utilisateurs tels que l'efficacité énergétique.

Abstract

The rise of machine learning techniques and algorithms and their use in a large variety of domains show often surprising capabilities – in the interpretation of input data and the ability to build relevant abstract representations (e.g. supervised learning), but also in the dynamic control of complex systems (e.g. reinforcement learning). Optimizing the energy efficiency of computing systems has become a major issue. From hardware design to software control, different levers exist to act on the execution of calculations. We consider in this thesis the optimization of parallel computing, running on complex architectures, from multicore processors to computing clusters. These systems present a large number of design and control parameters, giving rise to a number of combinations often too large to be considered exhaustively.

Thus, this thesis aims at using recent machine learning techniques, based on neural networks, to build solutions for the control and design of computing systems. These techniques can consider a significant set of parameters in order to propose optimal solutions. The proposed techniques are intended to be multi-level and can therefore be applied to embedded systems or various sub-components but also at the scale of a distributed system, such as a computing cluster.

Promising solutions are proposed along two distinct research axes. The first axis addresses the dynamic control of parallel computing. It deals with the real-time optimization of the energy efficiency of a system running an application parallelized with OpenMP. Among others, a control based on reinforcement learning is proposed. The second axis concerns the design of optimized communication networks. Indeed, communication networks represent a significant part of the energy consumption of computing systems. Thus, we propose a design tool based on generative AI, for the generation of optimized networks according to user criteria such as energy efficiency.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Systèmes de calcul

1.1.1 De la quête de performance

Depuis l'apparition des premiers microprocesseurs dans les années 1970, les performances des CPU monolithiques n'ont cessé de croître, conséquence direct de l'augmentation de la densité de transistors suivant l'allure prédite par la loi de Moore, i.e. le nombre de transistors présents sur un microprocesseur double tous les deux ans. Cependant, les sources traditionnelles d'amélioration des performances – e.g. parallélisme d'instruction (instructions indépendantes pouvant être exécutées en parallèle par différentes unités de calcul) et

augmentation de la fréquence d'horloge – semblent avoir atteint leur limite. En effet, des contraintes physiques telles que la chaleur dissipée par le système doivent être respectées afin d'assurer l'intégrité du système. Ainsi, dû à ces limitations, il s'avère inefficace d'augmenter la fréquence de fonctionnement ainsi que la densité du nombre de transistors. On notera sur la figure 1.1 que la fréquence de fonctionnement des CPU commercialisés stagne autour de 4 GHz depuis 2005, confirmant ces limitations.

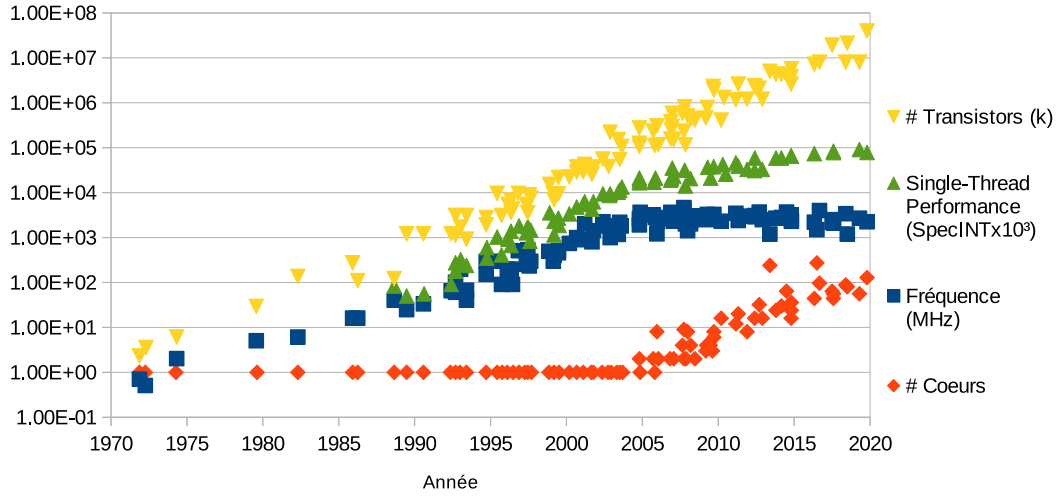


FIGURE 1.1 Évolution sur 48 ans des microprocesseurs commercialisés. *source :* [147]

En réponse à ces fortes contraintes, la conception des systèmes de calcul est entrée, au début des années 2000, dans l'ère des systèmes multi-cœurs [130]. Cette transition a permis aux performances des systèmes de continuer de croître via l'augmentation du nombre de CPU (i.e. cœur). Ainsi, la taille des cœurs est maintenue voire réduite, au profit d'un plus grand nombre de cœurs permettant l'accroissement global du nombre de transistors sur un même support silicium (voire figure 1.1). L'augmentation des performances est ainsi maintenue, tout en respectant les contraintes physiques évoquées précédemment. On remarque d'ailleurs sur la figure 1.1 une forte croissance du nombre de cœurs implémentés sur un même multiprocesseurs depuis 2005 – i.e. fin de l'augmentation de la fréquence d'horloge.

1.1.2 Vers la quête d'efficacité énergétique

De nos jours, ce ne sont plus seulement les performances qui motivent les travaux de recherches sur les systèmes de calcul, mais aussi la consommation énergétique. Ainsi, les progrès tendent vers l'amélioration de l'efficacité énergétique, définie comme la quantité de travail effectué relative à l'énergie dépensée à la tâche, i.e. $\frac{Performance}{Puissance}$. Améliorer l'efficacité

énergétique d'un système de calcul consiste donc à réduire la quantité d'énergie consommée et/ou augmenter les performances du système. Le développement des systèmes multi-cœurs s'est avéré utile dans cette quête pour l'efficacité énergétique. Par exemple, des études ont montré que l'utilisation d'un système double-cœur à 80 % de sa fréquence maximale permet de quasiment doubler les performances du système comparé à un processeur à un seul cœur fonctionnant à 100 % de sa fréquence maximale, pour une consommation énergétique similaire [165]. Pour ces raisons évidentes d'amélioration de l'efficacité énergétique, les architectures multi-cœurs se retrouvent sur tout type de systèmes de calcul, allant des systèmes embarqués aux systèmes distribués tels que les clusters de calcul.

Cependant, à cet usage multi-niveaux s'ajoute une complexité croissante des systèmes multi-cœurs. Il est désormais question de systèmes *manycore* possédant de la dizaine à plusieurs milliers de cœurs, ainsi que de systèmes hétérogènes composés de ressources aux performances, besoins et usages différents. Ainsi, de nouveaux enjeux existent pour pouvoir optimiser ces systèmes de calcul, allant des problématiques de conception aux défis liés à leur utilisation et contrôle.

1.2 L'optimisation du calcul parallèle

Les performances du calcul parallèle sont directement liées aux ressources de calcul sur lesquelles la charge de travail est partagée et exécutée. Ainsi, la complexité des systèmes multi-cœurs ne cessent de croître, et les systèmes *manycore* – i.e. possédant des centaines d'unités de calcul – pourrait bientôt devenir la norme. En effet, on peut citer le processeur AMD EPYC 7H12 commercialisé en 2019, possédant 64 cœurs physiques et 128 cœurs logiques, permettant à un particulier de s'offrir la puissance d'un serveur sur un unique processeur. Ce développement du nombre de ressources permet d'atteindre des niveaux de parallélisme jusqu'alors réservés aux clusters de calcul. Cependant, afin de profiter au mieux de ces ressources grandissantes pour garantir une efficacité énergétique optimale, un ensemble de paramètres de conception et de contrôle doivent être considérés.

1.2.1 Un ensemble de paramètres grandissant

Sur la figure 1.2 extraite de l'étude de Ryan Gary Kim *et al.* [87] sur le design de systèmes many-cœurs dédiés à l'apprentissage machine, les auteurs font une classification simple mais suffisamment complète pour démontrer la difficulté de l'optimisation des systèmes de calcul complexes. En effet, les leviers d'optimisation sont classés selon deux catégories

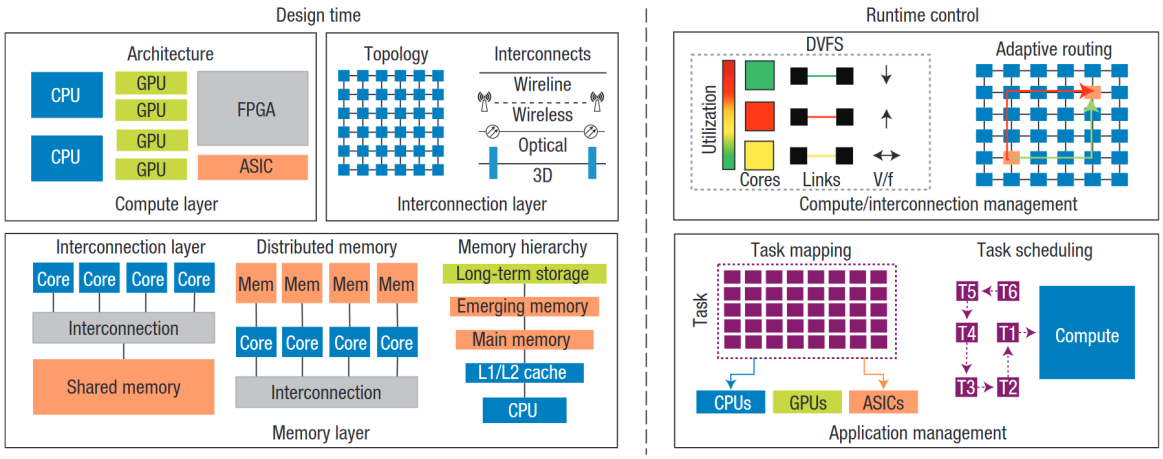


FIGURE 1.2 Une partie des leviers à considérer pour l'optimisation d'un système de calcul parallèle. Figure extraite de [87]

complémentaires : 1) l'optimisation de la conception du système, 2) l'optimisation du contrôle du système durant l'exécution du calcul.

L'étape de conception L'étape de conception du système de calcul joue un rôle primordial pour les performances énergétiques que possédera ce dernier. Les méthodes d'optimisation du design peuvent être réparties selon la partie du système adressée. On distingue généralement trois parties dans un système de calcul, avec leurs paramètres respectifs comme illustré sur la figure 1.2 :

— *Partie calculatoire :* Cette partie du système concerne les cœurs de calcul de la plate-forme. Les paramètres de conception comprennent le nombre de ressources de calcul, l'architecture de chacune de ces ressources (CPU, GPU, etc.), la répartition de ces ressources (i.e. système homogène ou hétérogène), ou encore le jeu d'instruction (e.g. *complex instruction set computer* (CISC), *reduced instruction set computer* (RISC)).

— *Partie mémoire :* Il s'agit du sous-système qui coordonne les accès aux données dans le système. Les paramètres de conception comprennent la hiérarchie (e.g. structure allant des cœurs de calcul à la mémoire principale, nombre de niveaux de caches, etc.), le type de gestion de la mémoire (e.g. mémoire distribuée ou partagée), ou encore les technologies utilisées avec un intérêt particulier pour les mémoires émergentes magnétiques lorsqu'il est question de faible consommation.

— *Partie communication :* Cela concerne le module d'interconnexion du système, assurant les échanges entre les différents composants. Ces échanges permettent entre autres la coordination entre les différents cœurs de calcul, et est donc un élément

particulièrement sensible concernant le calcul parallélisé. Les paramètres de conception comprennent la topologie du système d'interconnexion (bus, ring, mesh, etc.), l'architecture des routeurs (e.g. taille et type de buffers, nombre d'étages de pipeline) et le type de connexion (e.g. avec ou sans lien physique, bande-passante, uni ou bidirectionnelle).

Ces ensembles de solutions ne sont évidemment pas totalement distincts, et il est possible d'adresser simultanément plusieurs de ces groupes. Par exemple, la conception des buffers d'un routeur de la couche communication peut être vue comme une problématique liée à l'aspect mémoire.

L'étape de contrôle En complément de l'optimisation du design du système de calcul, ce dernier doit être ajusté en temps réel pour s'adapter aux conditions de fonctionnement variables. Comme illustré figure 1.2, les différents leviers d'optimisation en cours de fonctionnement peuvent globalement se répartir selon les catégories suivantes :

— *Gestion du support :* Cela concerne essentiellement le contrôle dynamique des paramètres des parties calculatoires et communication du système. Pour la partie calculatoire, on retrouve entre autres la sélection dynamique de la paire fréquence/tension de fonctionnement (DVFS) par cœur ou par processeur entier selon l'architecture, la gestion dynamique de la puissance (DPM) et la reconfiguration des cœurs de calcul. Pour la partie communication, on citera également la gestion de la fréquence de fonctionnement, mais aussi le routage adaptatif ou encore l'arbitrage de la communication.

— *Gestion des applications :* Cette catégorie regroupe les méthodes de gestion permettant d'adapter l'application aux caractéristiques du système. Cela comprend entre autres l'allocation des ressources aux tâches d'exécution (i.e. *task mapping*) et l'ordonnancement des tâches (*task scheduling*).

1.2.2 Des méthodes classiques limitées

Il est maintenant clair que l'espace de conception (la combinaison des décisions de conception et des politiques de contrôle et d'exécution du calcul) explose à mesure que le nombre standard de ressources augmente. La figure 1.2 montre un petit échantillon des leviers de conception disponibles aujourd'hui, tels que l'architecture des cœurs, l'architecture de la mémoire, l'allocation de tâches et le routage adaptatif. Chacun de ces leviers possède un grand nombre de paramètres découlant sur un espace de solutions où la recherche de la combinaison optimale est un challenge. En effet, les méthodes d'exploration exhaustive de l'espace de

conception ne sont plus envisageables aux regards du temps de calcul (i.e. problèmes NP-difficiles) et les techniques heuristiques devant répondre à ce problème sont sous-optimales. De plus, les méthodes heuristiques reposent généralement sur un ensemble d'hypothèses permettant de simplifier le problème, et requièrent donc le travail d'un expert afin de définir ces simplifications. De ce fait, ces méthodes ne garantissent pas l'obtention de solutions optimales et sont potentiellement sujettes à des biais de définition et des simplifications trop importantes. Par exemple, l'allocation de tâches est une méthode intéressante pour optimiser l'utilisation des ressources du système en fonction des tâches à accomplir. Cependant, lorsqu'il est question de trouver l'allocation optimale, on fait face à un problème NP-difficile [29]. Dans [78], Hoefler *et al.* exposent un ensemble de méthodes heuristiques développées pour l'allocation de tâches, et concluent sur une liste non exhaustive de challenges, dont l'incapacité à gérer des espaces de solutions toujours plus grands.

Les techniques d'apprentissage, et en particulier les réseaux de neurones montrent des capacités d'abstraction et de généralisation intéressantes, pouvant potentiellement pallier les limites des méthodes classiques pour la considération d'un espace de conception grand. Cette observation est partagée avec les auteurs de [87] qui explorent le secteur des systèmes "domain specific". Les techniques d'apprentissage machine sont donc à étudier afin de répondre au problème du nombre de paramètres.

1.3 Techniques d'apprentissage

1.3.1 Notions de base et hiérarchie des techniques

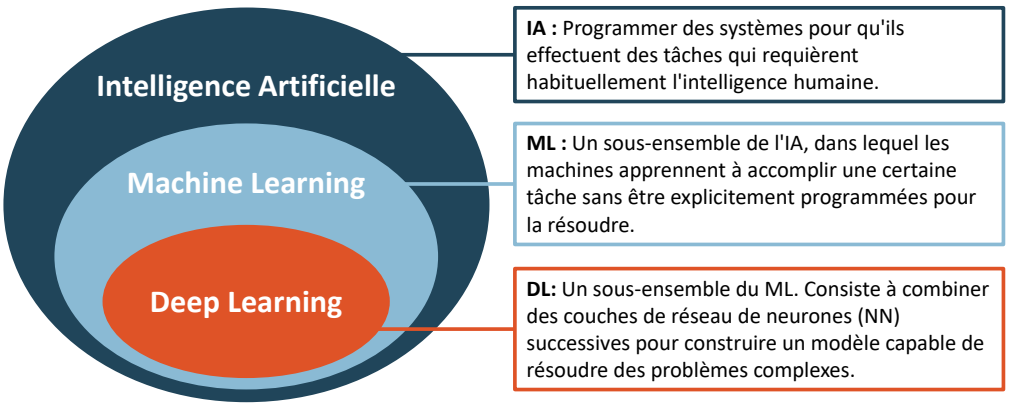


FIGURE 1.3 IA : vue d'ensemble

Le domaine de l'intelligence artificielle (IA) et plus particulièrement l'utilisation des techniques d'apprentissage s'est particulièrement développé ces dernières décennies. Pour bien comprendre la hiérarchie de ces techniques, une illustration est proposée figure 1.3. On représente l'IA comme un domaine de la science qui consiste à programmer des systèmes pour qu'ils réalisent des tâches nécessitant habituellement l'intervention de l'intelligence humaine. Dans ce domaine s'est développé un ensemble d'algorithmes et de méthodes permettant d'apprendre aux machines à réaliser des tâches sans nécessiter de programmation explicite dédiée à l'exécution de cette tâche – c'est le domaine des techniques d'apprentissage, ou ML pour *machine learning*. Les méthodes du ML reposent sur l'utilisation massive des données, ce qui explique leur développement récent, i.e. corrélé au développement des calculateurs. Les modèles prennent en entrées des données et sont entraînés pour évaluer et/ou agir sur ces données. Parmi les applications du ML, on retrouve la régression permettant de prédire la sortie d'un modèle appris (e.g. fonction complexe), ou encore la classification de données. Enfin, une catégorie particulière du ML aux capacités d'abstraction et de généralisation toujours plus impressionnantes est l'apprentissage profond, ou DL pour *Deep Learning*. Ce sous-groupe de techniques ML est basé sur l'utilisation de réseaux de neurones artificiels (NN, pour *Neural Networks*), agencés en couches. Les techniques de DL sont utilisées sur un très large panel d'applications, allant de la reconnaissance d'objet (e.g. *Convolutional Neural Networks* (CNN)) au traitement de données en série (e.g. traduction de texte avec les RNN (*Recurrent Neural Networks*)).

1.3.2 Classification des techniques

Les techniques ML, et par conséquent les techniques de DL, peuvent être classées en quatre catégories, selon leur méthode d'apprentissage :

— ***Apprentissage supervisé :*** Le système apprend à évaluer/classer des données d'entrée, sur la base d'un ensemble de données préalablement étiquetées. Le but est d'apprendre un modèle de classification prédéterminé. **e.g.** Régression linéaire, SVM, Arbre de décision.

— ***Apprentissage non-supervisé :*** Le système apprend à évaluer/classer des données d'entrée, sur la base d'un ensemble de données non étiquetées. Le but est que l'algorithme se construise seul un modèle de classification. **e.g.** K-means clustering, autoencoder.

— ***Apprentissage semi-supervisé :*** Le système se construit un modèle à l'aide d'un mélange de données étiquetées et non étiquetées. **e.g.** IA génératives.

— ***Apprentissage par renforcement :*** L'apprentissage par renforcement correspond au cas où l'algorithme apprend un comportement étant donné une observation. L'action de

l'algorithme sur l'environnement produit une valeur de retour qui guide l'algorithme d'apprentissage. C'est l'apprentissage par la pratique. **e.g.** Q-learning.

1.3.3 Le potentiel du DL

Des avancées impressionnantes ont été accomplies grâce au DL et leur utilisation des réseaux de neurones. Par exemple, *DeepMind* a pu démontrer le potentiel de l'apprentissage par renforcement avec son modèle de *AlphaGo* [159], la première IA à vaincre le N°1 mondial du jeu de Go. Les contributions de *DeepMind* ne s'arrêtent pas seulement à l'apprentissage profond par renforcement, mais concernent l'utilisation globale du deep learning pour répondre à de multiples problèmes complexes issus de domaines divers. On citera entre autres *AlphaFold* [82], une méthode révolutionnaire répondant avec une précision inédite au problème de prédiction du repliement des protéines.

Pour explorer plus loin le potentiel des réseaux de neurones, la figure 1.4 extraite du document de Miles Brundage *et al.* [34], montre l'évolution des IA génératives, en l'occurrence les GAN (*Generative Adversarial Network*), pour la production de portraits photoréalistes. Ce domaine des IA génératives est particulièrement intéressant pour ses facultés de création

FIGURE 1.4 Exemple du progrès des capacités de génération des GAN, de 2014 à 2017. *source :* [34]

de données, et permet ainsi d'apporter des solutions aux problèmes d'enrichissement d'un ensemble de données ou encore d'explorer un espace de données. De plus, on remarque sur la figure 1.4 que ces techniques d'IA génératives (ici, les GAN) se sont considérablement améliorées ces dernières années.

Alors que l'usage des techniques d'apprentissage s'est largement développé, en particulier les réseaux de neurones et les techniques de DL qui en découlent, il semble qu'il reste encore des bénéfices à en tirer dans le domaine de l'optimisation des systèmes de calcul, au sens large du terme.

1.4 Objectifs de thèse

Cette thèse a donc pour vocation d'explorer le domaine des techniques d'apprentissage pour apporter de nouvelles solutions aux enjeux liés à l'efficacité énergétique du calcul parallèle. En effet, dans le but d'optimiser la consommation liée au calcul parallèle, un ensemble grandissant de paramètres doit être considéré, limitant ainsi les performances des méthodes classiques majoritairement heuristiques. Or, les applications actuelles des techniques d'apprentissage profond (DL) montrent que ces techniques sont capables d'appréhender des problèmes complexes possédant un grand nombre de paramètres. Nous proposons donc d'étudier ces techniques pour pallier les limites des méthodes actuelles.

Un problème que nous adresserons dans un premier temps, est l'optimisation du calcul et sa gestion en temps réel. En effet, lorsqu'il est question du calcul parallèle, un des premiers points impactant la consommation d'une application parallélisée est la façon dont elle est répartie parmi les ressources de calcul. Cette répartition doit considérer à la fois les besoins de l'application, mais aussi les caractéristiques des ressources disponibles. Ainsi, l'usage de techniques d'apprentissage devra permettre de considérer ces différents paramètres afin de proposer un contrôle optimal du calcul.

Ensuite, une application parallélisée est une application exploitant largement le support de communication. Afin d'améliorer l'efficacité énergétique du calcul parallèle, il est donc nécessaire de concentrer des efforts sur l'aspect communication du calcul. Dans cette thèse, nous étudierons en particuliers les réseaux de communication sur puce et nous adresserons le problème de l'optimisation de ces réseaux.

1.5 Plan de thèse

Ce manuscrit de thèse vise à présenter le travail effectué au sein de l'équipe ADAC (*ADAptive Computing*) du LIRMM, dont l'objectif est de proposer des solutions d'amélioration de l'efficacité énergétique des systèmes numériques ciblés calcul parallèle, à l'aide d'outils d'apprentissage machine pertinents. Ce document est composé de cinq chapitres, sans compter l'introduction et la conclusion :

- *Chapitre 2 – Axes de recherche et problèmes adressés*, présente le contexte et les problématiques adressées durant cette thèse. En particulier, il est question de l'optimisation dynamique du calcul parallèle et de la conception de réseaux de communication sur puce optimisés, par le biais des techniques d'apprentissage.
- *Chapitre 3 – État de l'art*, expose l'état de l'art de chacune des problématiques adressées.

- *Chapitre 4 – Efficacité énergétique des calculs parallélisés OpenMP*, présente les travaux sur l'optimisation de l'aspect calculatoire des applications parallèles. Une métrique d'efficacité énergétique dédiée aux applications OpenMP est développée, ainsi qu'un outil de détection de phases d'exécution d'applications exploitant cette métrique. Une méthode de mapping adaptatif pour l'amélioration de l'efficacité énergétique de workloads OpenMP est proposée.
- *Chapitre 5 – Optimisation des topologies de réseaux de communication sur puce*, présente un outil de CAO développé pour la génération de topologies de NoC optimisées.
- *Chapitre 6 – Génération de réseaux sur puce hétérogènes optimisés*, propose une utilisation complémentaire de l'outil présenté *Chapitre 5* en s'adressant à la problématique de la conception de réseaux sur puce hétérogènes optimisés. Alors que dans le *Chapitre 5* il est question de la topologie des NoC, ici il sera question de leur composition (e.g. types de routeurs).

Chapitre 2

Axes de recherche et problèmes adressés

Sommaire

2.1 Introduction

L'optimisation de l'efficacité énergétique du calcul parallèle possède de nombreux challenges liés entre autres à la complexité des systèmes de calculs, ainsi qu'à la diversité des

applications parallélisées. En effet, pour améliorer l'efficacité énergétique d'un calcul, il est nécessaire de considérer les propriétés du système exécutant ce calcul. Ensuite, chaque calcul aura ses propres caractéristiques impactant son exécution sur le système considéré. Ainsi, différents challenges apparaissent, liant la conception du système de calcul et à l'exécution du calcul parallèle sur ce dernier.

Comme évoqué dans le chapitre précédent, il est possible de classer les méthodes d'optimisation existantes selon deux approches : l'optimisation à l'étape de contrôle (ou en-ligne) et l'optimisation à l'étape de la conception (ou hors-ligne). La première approche correspond à l'ensemble des solutions s'appliquant durant le fonctionnement du système. Nous nous intéressons plus précisément aux solutions durant l'exécution d'un calcul, destinées à ajuster les paramètres systèmes aux besoins du calcul. Ces solutions sont donc centrées autour du calcul et de son interaction avec le système exécutant. On y retrouve entre autres les modèles analytiques de codes d'applications permettant d'anticiper et d'accélérer la prise de décision en ligne, ainsi que les solutions dynamiques telles que le DVFS et DPM contrôlées généralement par le système d'exploitation. La seconde approche concerne la conception du système de calcul. Comme décrit par J.L. Hennessy et D.A. Patterson dans leur présentation à l'occasion de leur *Turing Award* [75], la tendance actuelle va vers une conception conjointe du système de calcul et des applications exécutables sur le système. Nous nous concentrons ici sur les solutions centrées autour de l'optimisation du support du calcul parallèle et regroupons l'ensemble des solutions matérielles permettant d'améliorer les performances et/ou la consommation énergétique des systèmes de calcul. On y retrouve entre autres les technologies de mémoires émergentes, l'optimisation de l'architecture du système de calcul et de son module d'interconnexion.

Dans le but de définir les problèmes adressés dans cette thèse, nous évoquerons dans ce chapitre les points qui suivent. Tout d'abord, nous parlerons de l'efficacité énergétique et des métriques utilisées pour la mesurer. Ensuite, nous nous intéresserons aux solutions d'optimisation logicielle du calcul parallèle avant de converger vers le contrôle dynamique du calcul. Nous verrons alors en quoi le ML peut apporter des solutions innovantes. Enfin, avant de définir précisément les problèmes adressés, nous évoquerons les challenges de la conception de systèmes optimisés. Nous nous concentrons plus particulièrement sur les réseaux sur puce, principale solution choisie pour concevoir le module d'interconnexion d'un système multi-cœurs, et présentant des problèmes de consommation énergétique auxquels nous tenterons d'apporter une solution via le ML.

2.2 Mesure de l'efficacité énergétique d'une application parallèle

L'efficacité énergétique a été caractérisée dans la littérature par différentes mesures (par exemple, FLOPS/W et MIPS/W) en fonction des domaines informatiques. Dans tous les cas, ces métriques suivent la définition générale :

$$\text{Efficacité énergétique} = \frac{\text{Quantité de travail par unité de temps}}{\text{Puissance consommée}}$$

2.2.1 Solutions existantes

Plusieurs techniques de mesure ont été proposées au cours des dernières décennies pour caractériser la consommation énergétique. A ce titre les études littéraires [121, 10] font un état des lieux des méthodes existantes. Elles peuvent être distinguées en fonction du niveau d'abstraction du système de calcul où elles opèrent.

Les auteurs de ces contributions ont fait état d'un certain nombre de techniques d'acquisition de données de consommation d'énergie par le biais de capteurs matériels, situés à différents endroits physiques des systèmes. Ces techniques ont leurs avantages et leurs inconvénients en termes de précision de mesure, résolution temporelle, coût de déploiement et de leur caractère intrusif. Elles comprennent des circuits de mesure intégrés dans des composants matériels tels que les GPU et les CPU ; des dispositifs d'instrumentation insérés dans les nœuds de calcul capables de sonder au niveau des composants ou des voies d'alimentation ; et des compteurs d'énergie qui peuvent recueillir la charge totale en dehors de l'alimentation électrique du nœud.

Cependant, ces solutions permettent de mesurer uniquement la consommation énergétique et/ou la puissance consommée, et ne suffisent donc pas à elles seules pour extraire une mesure d'efficacité énergétique, qui nécessite une donnée de performance. Habituellement, cette donnée de performance vient des compteurs de performances, permettant notamment d'extraire le nombre d'instructions exécutées (#inst., une quantité de travail) et d'en extrapoler le nombre d'instructions exécutées par seconde (IPS) ou encore le nombre d'opérations en virgule flottante par seconde (FLOPS), deux données de performance. Les mesures hybrides d'efficacité énergétique ainsi extrapolées sont le nombre d'instructions par seconde par watt (IPS/W) et FLOPS par watt. Cependant, ces mesures caractérisent l'efficacité énergétique du système de calcul globale (ou partiel selon les compteurs et capteurs disponibles) et sont donc bruitées par l'ensemble des processus exécutés et ne permettent pas de caractériser

l'efficacité énergétique d'une application précise. En effet, les compteurs de performances ne permettent pas d'identifier le processus à l'origine de l'évènement mesuré.

En complément des capteurs et compteurs matériels mentionnés ci-dessus, des approches logicielles sont également utilisées. L'API de performance (PAPI) [33] est une interface de bibliothèque bien connue (i.e. une couche logicielle) pour les compteurs de performance matériels qui facilite l'extraction de diverses statistiques d'exécution. D'autre part, d'autres outils de profilage comprennent les *Tuning and Analysis Utilities* (TAU) [157], Score-P [84], Scalasca [64] et PowMon [166]. Les outils de profilage de la puissance au niveau logiciel comprennent pTop [53], PowerAPI [93] et Jalen [120]. Le premier est similaire au programme "top" de GNU/Linux et fournit des données sur la consommation d'énergie des processus en cours d'exécution. L'interface de programmation d'applications PowerAPI et l'architecture de profilage au niveau logiciel Jalen permettent de surveiller la consommation d'énergie en temps réel. Les informations fournies par ces outils sont souvent obtenues a posteriori, c'est-à-dire après l'exécution d'une grande partie (voire de la totalité) d'un programme donné, ce qui ne laisse que peu de place aux optimisations précoces.

J. C. R. da Silva *et al.* [48] proposent une solution centrée sur l'environnement Android permettant une mesure précise de la consommation énergétique d'une application. En particulier, dans cette contribution les auteurs proposent une solution permettant de connaître la consommation énergétique de petites parties de code telles que des boucles ou des appels de fonction. Cette dimension de lien direct entre le code source d'une application et sa consommation se détache des solutions classiques reposant sur les compteurs de performance, ne permettant pas d'associer directement les mesures avec le code exécuté. Cependant, ce travail reste dédié à l'environnement Android, et son utilisation nécessite à la fois un module matériel supplémentaire ainsi que des modifications du code source, ce qui s'éloigne de notre objectif de solution multi-niveaux et facilement implémentable.

Enfin, des solutions exploitant la simulation sont proposées afin de déterminer l'efficacité énergétique de système exécutant une application, à partir de modèle. Par exemple, A. Butko *et al.* [35, 36] proposent une simulation complète de l'architecture multi-cœur hétérogène big.LITTLE afin d'explorer les configurations du système apportant les meilleurs résultats en termes de performance et efficacité énergétique à l'aide de l'environnement gem5 [26]. Les auteurs ont proposé des simulations orientées traces [37, 119], afin d'accélérer l'exploration de systèmes à grande échelle. D'autres solutions se concentrent sur l'analyse de codes sources afin de prédire la consommation énergétique d'une application en fonction d'un système de calcul cible. Entre autres, T. Béziers la Fosse *et al.* [92] proposent un modèle de caractérisation énergétique de code source, à partir de données mesurées, et E. Parisi *et al.* [129] présentent une solution à base de ML pour classifier un code source selon son

efficacité énergétique. Ces approches analytiques, basées sur des modèles d'estimation de haut niveau pour les performances et la consommation d'énergie, pourraient être utilisées pour une optimisation dynamique antérieure. Malheureusement, ces modèles ne sont pas nécessairement des approximations fiables pour toutes les plate-formes d'exécution et restent spécifiques à un système de calcul considéré.

L'approche proposée dans cette thèse fonctionne au niveau logiciel, via le runtime OpenMP. Les données de performance sont collectées en temps réel à partir de l'environnement d'exécution OpenMP, soit à la frontière entre la couche logicielle et la couche matérielle. Cela permet l'implémentation de cette méthode sur toute plate-forme supportant OpenMP. De plus, ces données de performances sont spécifiques à l'application considérée, contrairement aux compteurs de performance e.g. IPS. Les données sur la puissance et l'énergie consommée dépendent, quant à elles, du support.

2.2.2 Métrique spécifique via OpenMP

Nous recherchons donc une solution permettant de mesurer en temps réel l'efficacité énergétique d'une application. Nous nous intéressons plus particulièrement aux applications parallèles. Ainsi, notre intérêt se porte sur les applications OpenMP.

OpenMP est le modèle de programmation pour le calcul parallèle le plus populaire pour les systèmes à mémoire partagée. Son API supporte les langages de programmation C/C++ et Fortran. De plus, OpenMP est pris en charge par la majorité des plate-formes dont Windows et UNIX, ce qui justifie son usage très largement répandu.

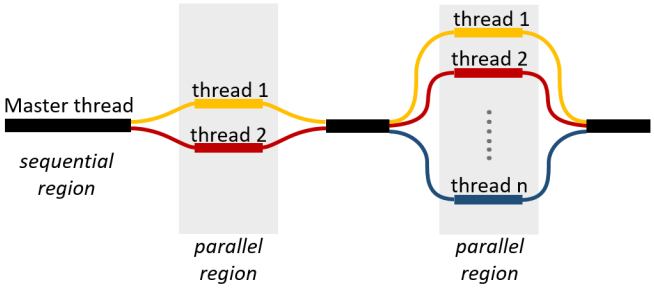


FIGURE 2.1 OpenMP : mécanismes *Fork* et *Join*.

Le parallélisme sous OpenMP est majoritairement basé sur l'utilisation de processus légers, aussi appelés threads. Il existe, depuis la version OpenMP 3.1, le mécanisme de gestion de tâches communicantes, mais ce dernier étant encore peu utilisé, nous ne nous y sommes pas intéressés. Les threads sont la plus petite unité de traitement qui soit gérée par un système d'exploitation. OpenMP propose un ensemble de fonctionnalités permettant de

contrôler la parallélisation et la synchronisation de threads. Elles sont toutes basées sur deux principes fondamentaux : *Fork* et *Join*. Comme illustré sur la figure 2.1, le mécanisme de *Fork* fait la jonction entre une région séquentielle (thread maître), et une région parallèle où les instructions du programme sont exécutées en parallèle sur un ensemble de threads travailleurs formant une équipe. Ensuite, la jonction inverse faisant la transition entre une région parallèle et une région séquentielle correspond au mécanisme de *Join*. C'est à ce moment-là que les threads travailleurs d'une même équipe se synchronisent et sont supprimés pour ne laisser s'exécuter que le thread maître.

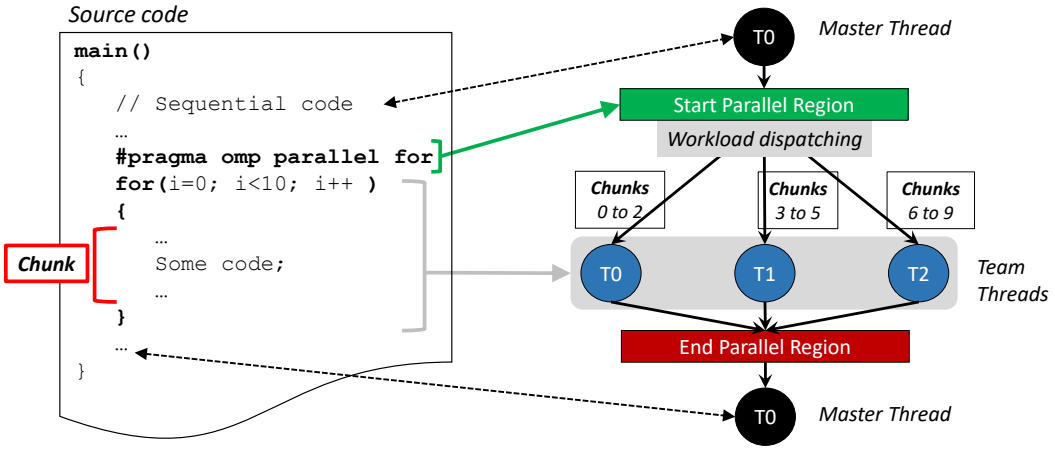


FIGURE 2.2 Schéma explicatif du concept de chunk.

Parmi les fonctionnalités de OpenMP, nous nous intéressons aux boucles parallélisées (e.g. boucle *for*). À l'intérieur de ces boucles, le workload est divisé en blocs d'instructions appelés chunks. Ces chunks correspondent aux itérations de la boucle parallélisée. Ils sont distribués pour exécution aux différents threads travailleurs créés au lancement de la région parallèle correspondante. Sur la figure 2.2 est illustré le concept de chunk. Sur ce schéma est représenté un code composé d'une boucle *for*. Cette boucle est parallélisé grâce à l'unique instruction "#pragma omp parallel for". Cette instruction indique au runtime OpenMP de démarrer une région parallèle contenant les threads travailleurs. La région parallèle se termine automatiquement avec la fin de l'exécution de la boucle. Il existe différentes stratégies de planification définissant la façon dont les chunks sont répartis parmi les threads. Elles peuvent être de deux types : statique ou dynamique. Dans le premier cas, la charge de travail est répartie équitablement parmi les threads, c'est l'exemple illustré sur la figure 2.2. Dans le second, les chunks sont attribués de façon dynamique, en fonction de la progression des threads. Cette dernière stratégie, bien qu'induisant un coût de planification, est prouvée efficace et est d'autant plus utilisée sur des systèmes hétérogènes tels que les architectures big.LITTLE.

Ainsi, nous proposons d'exploiter la notion de chunks décrite dans OpenMP comme une unité de mesure permettant de suivre en temps réel l'évolution de l'exécution d'une application OpenMP parallélisée. Les applications considérées seront donc celles contenant des boucles *for* parallélisables.

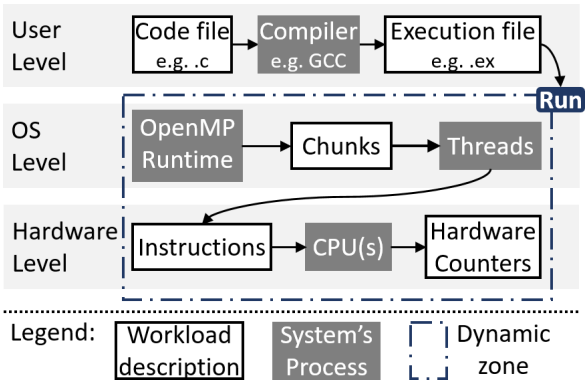


FIGURE 2.3 Étapes d'exécution d'un workload OpenMP.

Sur la figure 2.3 sont décrites les différentes étapes d'exécution d'un workload OpenMP constitué de boucles parallèles, du niveau utilisateur au niveau matériel. À chaque étape est précisée la granularité de la charge de travail. Cela démarre donc avec un fichier contenant le code source du programme et les instructions OpenMP pour la parallélisation, et cela termine par les compteurs matériels mis à jour durant l'exécution du programme. Le niveau de granularité supérieur des chunks est facilement visible puisqu'il correspond au premier niveau de description du workload durant l'exécution de l'application (c.f. cadre *Run* sur la figure 2.3). Cette description met en avant les avantages d'utiliser OpenMP qui est compatible avec la majorité des langages de programmation tels que C/C++, Fortran, etc, ainsi que la plupart des architectures, contrairement aux compteurs matériels qui sont spécifiques aux composants matériels (i.e. *hardware-specific*).

2.3 Optimisation de l'exécution du calcul parallèle

L'exécution du calcul parallèle peut être optimisée au cours de différentes activités du cycle de vie du calcul, et à différents niveaux d'abstraction. On distingue trois étapes au cours de ce cycle : la conception et l'implémentation, la compilation, et l'étape d'exécution à proprement parler. Pendant la conception et l'implémentation, des décisions telles que la sélection du langage/modèle de programmation et la sélection de la stratégie de parallélisation sont prises en compte. Les optimisations de compilation comprennent les décisions de sélection des drapeaux d'optimisation du compilateur et des transformations du code source

(telles que le déroulement des boucles, l'optimisation des nids de boucles, le pipelining et l'ordonnancement des instructions) de sorte que le programme exécutable soit optimisé pour atteindre certains objectifs (performance ou énergie) dans un contexte donné. Les activités d'exécution comprennent les décisions relatives à la sélection des données optimales et à l'ordonnancement des tâches sur les systèmes de calcul parallèles, ainsi que les décisions (telles que le DVFS) qui aident le système à s'adapter au cours de l'exécution du programme et à améliorer les performances globales et l'efficacité énergétique. Alors que les activités de conception et de mise en œuvre du logiciel sont réalisées par le programmeur, les activités logicielles au moment de la compilation et de l'exécution sont effectuées par des outils (tels que des compilateurs et des systèmes d'exécution).

2.3.1 Optimisation durant la conception et modèles haut-niveaux

Une grande partie de la littérature concerne les approches basées sur des modèles d'abstraction haut-niveaux permettant de simuler le comportement d'une application exécutée sur un système de calcul particulier. Avec la complexité croissante des architectures de calcul d'aujourd'hui, la simulation est devenue un outil indispensable pour explorer l'espace de conception des systèmes. Dans le cadre de l'optimisation de l'exécution d'un calcul, ces modèles permettent d'anticiper le comportement du calcul et de prévoir par exemple des ordonnancements de tâches particuliers.

Une approche basée SDF (*synchronous dataflow*) est proposée par M. Pelcat *et al.* [133]. Les auteurs proposent S-LAM, un modèle simple et précis de description d'architecture à haut-niveaux d'abstraction, prenant en considération les échanges de données au sein d'une architecture hétérogène. Ce modèle permet, une fois inclus dans le framework PREESM [132] de réaliser le prototypage rapide d'un déploiement de tâche – i.e. allocation (ou *mapping*) et ordonnancement (ou *scheduling*) sur un système multi-cœurs sur puce (MPSoC). Le déploiement ainsi généré peut être donné en entrée d'un simulateur basé en SystemC. De cet ensemble résulte un processus rapide et précis de prototypage d'algorithme parallélisé sur une architecture donnée.

Une approche analytique basée sur des horloges abstraites est présentée par X. An *et al* [11, 12]. L'outil CLASSY [11] proposé permet la description et l'analyse à base d'horloge de l'ordonnancement d'applications sur des systèmes composés de cœurs fonctionnant à différentes fréquences. Un algorithme est proposé pour réaliser l'ordonnancement automatiquement, en respectant des contraintes utilisateur. Ce travail a été approfondi [12] afin d'inclure, entre autres, une méthode d'exploration de l'espace de conception, proposant des solutions pareto-optimales. Une méthode pour l'exploration d'espace de conception est également proposée par K. Latif *et al.* [94] pour des applications automobiles. Une approche

orientée graphes et automates est également présentée par X. An *et al* [13] pour la conception de contrôleurs fiables pour la reconfiguration dynamique d'architecture.

Les méthodes décrites précédemment concernent essentiellement les systèmes embarqués [133, 13, 94] et MPSoC [11, 12]. Cependant, il existe aussi un intérêt pour la modélisation de systèmes HPC. En effet, comme décrit par K. Ahmed *et al.* [6], la modélisation et la simulation de systèmes HPC sont des outils cruciaux pour la conception de ces systèmes toujours plus sollicités. Par exemple, dans [168] G. Xu *et al.* proposent un modèle pouvant simuler, sur un PC privé, l'exécution d'un calcul exascale sur un système HPC. Des travaux plus spécifiques sont également conduits, comme la contribution de M. Mubarak *et al.* [116] sur la simulation de communication au sein de systèmes HPC.

L'ensemble de ces méthodes montre un intérêt certain pour la modélisation haut-niveau du calcul parallèle, qui permet de simplifier l'analyse de systèmes pourtant complexes. Cependant, dans cette thèse nous nous intéressons plus spécifiquement aux approches plus bas niveau, proches du matériel, minimisant les efforts "hors-ligne". En particulier, les méthodes de modélisation nécessitent un travail de description des programmes et des systèmes de calcul dans de nouveaux formats (e.g. SDF, évènements horloge), qui ne sont pas toujours compatibles pour une optimisation en temps réel.

2.3.2 Optimisation durant l'exécution

Nous nous intéressons plus particulièrement aux solutions adressant l'optimisation durant l'exécution du calcul parallélisé. En effet, la gestion au niveau du runtime ouvre des perspectives de solutions adaptatives pour un ajustement en temps réel. De plus, on constate un faible choix d'optimisation runtime sur les systèmes de calcul commercialisés, sujets pourtant au calcul parallèle. En effet, on retrouve sur les systèmes commercialisé différents pilotes (e.g. *intel_pstate*, *acpi-cpufreq*) agissant essentiellement sur la fréquence de fonctionnement des cœurs (i.e. *Dynamic Voltage-Frequency Scaling* (DVFS)) ou le mode de puissance (i.e. *Distributed Power Management* (DPM)). Entre autres, les gouverneurs Linux sont un ensemble de gestionnaires d'exécution qui permettent de contrôler la configuration du système selon les besoins logiciels et les contraintes de performance. Il existe par exemple le gouverneur *performance* avec pour objectif d'exploiter le maximum des capacités du système, le *powersave* qui à l'inverse cherche à diminuer la consommation instantanée au prix d'une perte en performance, et le gouverneur adaptatif *ondemand* [127] conçu pour adapter les performances du systèmes en fonction de ses besoins courants. Ces solutions n'adressent qu'une partie des leviers d'optimisation ne gérant que le DVFS et DPM. La gestion du mapping et scheduling est généralement prise en charge par l'OS, ou sous la responsabilité du programmeur.

Les solutions existantes dans les systèmes commercialisés adressent globalement l'optimisation de l'efficacité énergétique par le prisme du système matériel (performances et consommation des cœurs, états de fonctionnement des cœurs, etc.). Nous pensons donc qu'il y a des gains à générer via la conception de systèmes de contrôle ajustant les paramètres d'exécution du calcul par le prisme de l'application (i.e. suivi *application-specific*).

2.4 Vers le contrôle de l'efficacité énergétique des applications OpenMP

Dans le but d'exploiter le potentiel temps réel des chunks, nous concentrerons notre étude sur le contrôle d'application OpenMP. Nous adressons donc le problème du contrôle dynamique d'application OpenMP pour l'optimisation de l'efficacité énergétique. Contrairement aux méthodes incluses dans les systèmes commercialisés, nous souhaitons adresser à la fois le contrôle en fréquence (i.e. DVFS) et l'allocation des ressources (i.e. mapping). Plus de détails sur l'état de l'art sont donnés section 3.1.

2.4.1 Une multitude de paramètres

Le contrôle combiné de la fréquence de fonctionnement et de l'allocation des ressources peut rapidement s'avérer délicat avec l'augmentation des ressources disponibles. L'objectif de notre système de contrôle sera de déterminer en temps réel la meilleure configuration (nombre de cœurs attribués, fréquence des cœurs) du point de vue de l'efficacité énergétique. Or, dans le cas par exemple du serveur Intel Xeon utilisé dans nos expérimentations, ce dernier dispose de 20 cœurs physiques et chacun de ces cœurs peuvent être contrôlés pour fonctionner entre 1.2GHz et 2.2GHz. Cela représente un ensemble de 220 configurations différentes. Si on considère de plus un contrôle en fréquence individuel des cœurs, le nombre de configurations explose.

Face à ce nombre important de solutions possibles, il n'est pas concevable de tester chacune des solutions afin de déterminer laquelle optimise l'efficacité énergétique du calcul. De plus, le contrôleur doit pouvoir s'adapter à un changement de nature du calcul qui modifiera potentiellement ses caractéristiques d'efficacité énergétique, et il faudra donc de nouveau tester l'ensemble des configurations. Dans cet axe de recherche, nous proposons d'explorer les méthodes de ML et leurs capacités de généralisation et d'interpolation afin de correctement gérer ces grands ensembles de solutions et rapidement proposer un contrôleur efficace.

2.4.2 Prise de décision automatique et RL

Parmi les méthodes existantes, l'usage d'apprentissage par renforcement (RL) semble être la solution idéale pour notre contrôleur. En effet, l'apprentissage par renforcement (RL) est avéré efficace pour les problèmes de prise de décision. Il garantit théoriquement la convergence globale vers la solution la plus performante parmi les actions disponibles, bien que le temps de convergence soit souvent prohibitif. C'est donc une solution intéressante pour les problèmes de décision non intuitifs, c'est-à-dire les problèmes pour lesquels aucun modèle de système satisfaisant ne peut être construit.

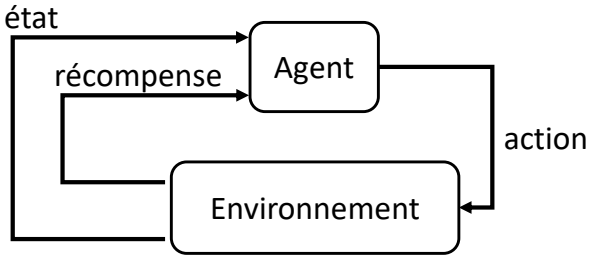


FIGURE 2.4 Diagramme de concept du RL.

Dans l'apprentissage par renforcement, on définit l'algorithme de contrôle (au sens du code et de ses variables) comme l'*agent*. Ce dernier interagit avec l'*environnement* pour trouver la solution optimale. L'environnement est décrit par son *état*. Les interactions de l'agent sur l'environnement sont appelées *actions*. La figure 2.4 illustre le principe de fonctionnement du RL. L'apprentissage du RL consiste à apprendre par l'expérience. Les actions menées sont évaluées via une fonction de récompense. On distingue généralement deux phases de fonctionnement dans l'utilisation de RL : la phase d'*exploration* durant laquelle l'agent produit des actions plus ou moins aléatoires afin d'explorer l'ensemble des solutions et de mémoriser un ensemble de données composé de paires état/action associées à la récompense produite. Puis un apprentissage est mené à partir de cette expérience d'exploration afin de laisser l'agent décider de la meilleure action à mener en fonction de l'état de l'environnement pour obtenir la meilleure récompense. Les problèmes de RL nécessitent donc une définition minutieuse non seulement de l'*état* et de la fonction de récompense, mais aussi des actions possibles pour concevoir un contrôleur satisfaisant, comme décrit dans [23] et [54] pour lesquels l'espace d'action se repose sur le DVFS et le DPM pour le premier et l'allocation de tâches pour le second.

Ainsi, nous proposons d'explorer l'usage de RL pour la construction d'un système de contrôle. L'objectif de ce système de contrôle sera d'apprendre automatiquement à contrôler de façon optimale une application OpenMP afin de garantir en permanence la meilleure

efficacité énergétique possible. Ce contrôle agira sur la configuration du système de calcul. Les détails de notre solution sont apportés dans le chapitre 4.

2.5 Conception de systèmes sur puce optimisés

Les sections précédentes ont permis d'évoquer les recherches sur l'optimisation en-ligne du calcul parallèle, et de converger vers le problème du contrôle dynamique d'application OpenMP. Cependant, l'efficacité énergétique d'un calcul repose également sur les caractéristiques du système qui l'exécute. Ainsi, nous discutons ici de l'optimisation du design des systèmes de calcul. Après une rapide vue d'ensemble des paramètres de conception, nous nous concentrerons en particulier sur les systèmes multi-cœurs sur puce (MPSoC), et le module d'interconnexion qui assure la communication inter-cœurs.

2.5.1 Paramètres de conception

2.5.1.1 Architecture et composants

Dans [117], R. Muralidhar *et al.* font état des tendances de conception architecturale guidées par la recherche de l'amélioration de l'efficacité énergétique, dont entre autres le problème de la gestion du *dark silicon* [59] et le concept du calcul énergie-proportionnel [17]. Les auteurs décrivent notamment les techniques micro-architecturales spécifiques aux composants présents sur un système de calcul (i.e. CPU, GPU, mémoire, etc.) permettant une optimisation de ces derniers. L'ensemble de ces techniques est vaste, comme déjà évoqué en introduction (chapitre 1) avec l'étude de Ryan Gary Kim *et al.* [87]. Nous pouvons évoquer en particulier les techniques dédiées à la mémoire, élément important d'un système de calcul lorsqu'il est question de la consommation énergétique. De récentes avancées technologiques sont largement explorées dans la littérature, comme par exemple l'intégration de mémoires non-volatiles faisant l'objet de divers travaux au sein de l'équipe ADAC [131, 135, 136, 134, 32, 153, 154, 46].

Nous nous concentrons ici sur la conception de MPSoC, qui sont des systèmes généralement soumis à de fortes contraintes énergétiques de par leur utilisation courante sur des systèmes embarqués. De plus, ces systèmes sont par définition conçus sur un unique support silicium, ce qui pose de nombreux challenges de conception.

2.5.1.2 MPSoC : Un ensemble de *IP blocks* interconnectés

Un MPSoC, et plus généralement un SoC, consiste en un ensemble de blocs de propriété intellectuelle (en anglais *IP block*, pour *intellectual property block*) interconnectés. Le support de communication joue donc un rôle clef dans les performances d'un système sur puce. Les enjeux de ce module sont d'autant plus importants sur les MPSoC étant donné leur nombre croissant d'éléments à interconnecter. La figure 2.5 présente une architecture simplifiée de système sur puce qui consiste en un ensemble de composants aux fonctions différentes (e.g. CPU pour l'exécution des principales tâches et calculs comme le système d'exploitation, et DSP pour l'exécution d'applications multimédias riches en calculs mathématiques), permettant de diversifier les types d'applications exécutables. Le module d'interconnexion est

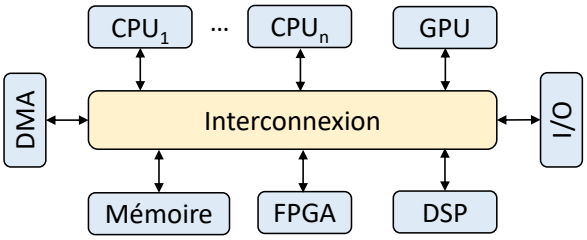


FIGURE 2.5 Exemple d'une architecture de SoC multi-cœurs.

l'élément permettant de connecter l'ensemble des composants présents dans le système et est considéré comme la clef de voûte des systèmes multi-cœurs. Des études ont d'ailleurs montré qu'à mesure que le nombre de cœurs augmente, l'interconnexion devient un facteur dominant, imposant des contraintes de performance et de puissance significatives sur la performance globale du système [31]. Il est donc essentiel de disposer d'une interconnexion sur puce optimisée capable de fournir une bande passante élevée et une faible latence pour le transfert de données entre les blocs IP.

2.5.2 Le NoC : un module d'interconnexion encore coûteux en énergie

Le réseau sur puce (NoC) [50] est devenu le principal composant utilisé pour l'interconnexion des SoC multi-cœurs, en raison de sa flexibilité et de sa facilité d'implémentation [3]. Avec la complexité croissante des systèmes multiprocesseurs sur puce (MPSoC) – i.e. l'augmentation du nombre de cœurs dans la puce, les architectures hétérogènes, etc. – trouver l'équilibre entre les performances et la puissance des systèmes est une tâche difficile. Les NoC représentent une part importante de la consommation d'énergie des puces (e.g. 28% de la puissance totale de la puce Intel Terascale 80-chip [80], 36% pour le MIT RAW [162] et 19% pour la puce *SCORPIO* [51]). Ainsi, la conception d'un NoC économe en énergie permettrait de réduire significativement la consommation des puces multi-cœurs [8].

2.5.2.1 Notions sur les NoC

Le NoC se compose de routeurs interconnectés par des liaisons de données. Les routeurs jouent un rôle clef dans l'acheminement des paquets de leur source vers leur destination, tandis que les liaisons sont des ensembles de connexions qui relient les routeurs entre eux et assurent le transfert des données entre les routeurs. La manière dont les routeurs sont disposés dans le réseau est régie par la topologie. Les topologies les plus courantes sont le *mesh*, le *torus* et le *ring*. Lorsqu'on décrit une topologie de NoC, on emploie généralement le vocabulaire des graphes, en désignant les routeurs comme les nœuds (ou sommets), et les connexions comme les liens (ou arêtes). Nous verrons dans le chapitre 4 que l'analogie aux graphes va plus loin.

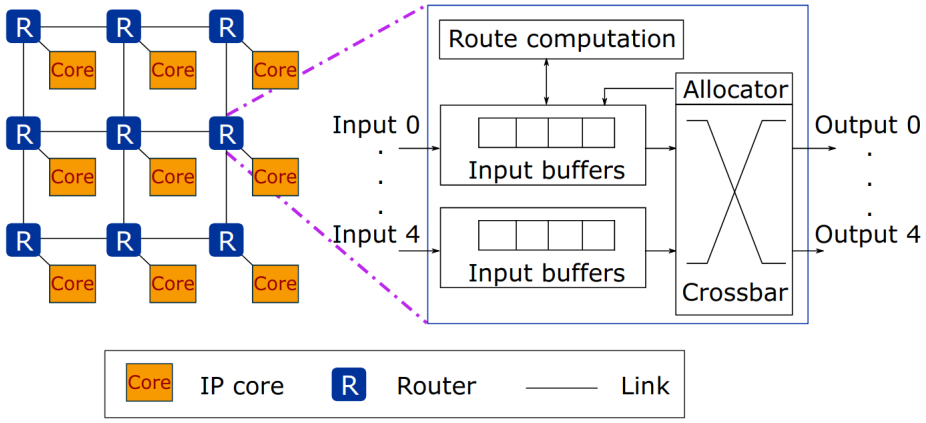


FIGURE 2.6 Exemple de topologie *mesh* avec l'architecture des routeurs à buffers d'entrée. *source :* [56]

La figure 2.6 extraite de la thèse de Charles Effiong [56] présente une topologie classique de réseau *mesh*, ainsi qu'une architecture typique de routeurs à buffers d'entrée similaire au routeur HERMES [114]. Dans le cadre de sa thèse, C. Effiong a exploré en détail l'architecture des routeurs pour proposer sa propre architecture R-NoC inspirée du fonctionnement des rond-points pour le trafic routier. Dans notre cas, l'architecture des routeurs ne sera pas explorée, et dépendra essentiellement des simulateurs utilisés (voire les chapitres de contribution 5 et 6). Il sera donc essentiellement question de routeurs à buffers d'entrée classiques tels que celui présenté figure 2.6. Le routeur comporte 5 ports entrées/sorties, désignés Nord, Sud, Est, Ouest, Local, où le port Local connecte le routeur au cœur physique sous-jacent, via une interface réseau. Les 4 autres ports sont dédiés aux connexions inter-routeur.

Topologie : Les topologies de NoC peuvent être régulières ou irrégulières [3]. Les topologies régulières sont connectées selon un modèle (ou motif) spécifique, comme les topologies

mesh, torus, star, ring ou encore *tree*. À l'inverse, une topologie irrégulière ne possède pas de motif particulier. La topologie du réseau affecte de manière significative les performances globales du réseau [49]. En effet, elle détermine la longueur du chemin (i.e. nombre de sauts entre deux routeurs) à parcourir par un message depuis sa source vers sa destination. Un chemin plus long se traduit par une latence et une consommation d'énergie plus élevées pour l'envoi d'un message. De plus, la fiabilité est aussi impactée par la topologie, puisqu'elle spécifie le nombre de chemins alternatifs permettant de pallier d'éventuels défauts et conflits. Enfin, la topologie spécifie également le nombre de routeurs présents dans le réseau, ce qui affecte directement la surface et la puissance consommée et donc le coût du réseau.

Routage : Le routage définit le chemin qu'empruntera un message pour rejoindre sa destination, à partir du routeur émetteur. L'objectif du routage est donc d'assurer la bonne transmission des données. Il empêche les situations d'impasse (*deadlock*), de blocage (*live-lock*) et de famine (*starvation*) [5]. Le deadlock est une dépendance cyclique entre les nœuds qui accèdent aux ressources et où aucun progrès ne peut être réalisé. Le live-lock est une situation où un paquet circule dans le réseau mais n'atteint pas sa destination. En cas de famine, un paquet dans un buffer demande l'accès à un canal de sortie, mais le canal de sortie est également alloué à un autre paquet.

Les algorithmes de routage peuvent être classés en deux types : déterministe et adaptatif [139]. Un routage déterministe signifie que pour les mêmes routeurs de départ et d'arrivée, le chemin emprunté par deux messages différents sera identique. Un exemple connu est le routage *XY* généralement utilisé dans les topologies de type mesh. Ce routage consiste à transmettre un message en premier lieu sur la dimension X (généralement l'axe Est-Ouest). Une fois que le message a atteint un routeur de même coordonnée X que la destination, il est transmis sur l'axe de la dimension Y (généralement l'axe Nord-Sud). Les routages déterministes sont simples à implémenter, cependant il sont sujets à des pertes de performances importantes lorsqu'on atteint un niveau de contention élevé sur le réseau, puisqu'ils ne permettent pas d'exploiter des chemins alternatifs. Les routages adaptatifs apportent plus de flexibilité en permettant d'adapter le chemin d'un message en fonction des différentes conditions de trafic sur le réseau. Alors que ces types de routage permettent d'améliorer la fiabilité du réseau, leur implémentation est beaucoup plus complexe puisqu'elle nécessite souvent des systèmes de contrôle additionnels pour éviter les situations d'impasse, de blocage et de famine.

Techniques de commutation : La technique de commutation fait référence au mécanisme de contrôle du flux des messages entre les routeurs. Les techniques de commutation de

base utilisées dans les NoC sont la commutation de circuits et la commutation de paquets. La commutation de paquets se divise en trois grandes catégories : le *wormhole*, le *store-and-forward* et le *virtual-cut-through*(VCT) [5]. Dans le wormhole, le paquet est divisé en flit (flit de tête, flit de corps et flit de queue). Le flit de tête contient les informations de source et de destination, le flit de corps contient les données qui sont transmises à la destination et le flit de queue contient les informations de fin de flit. En raison de la nature "pipelinée" du *wormhole*, cette technique réduit la latence des messages. Dans la technique *store-and-forward*, le paquet entier est stocké dans le routeur puis acheminé vers le routeur suivant. Dans la technique du VCT, le paquet est transmis au routeur suivant s'il s'assure que le paquet entier peut y être stocké. En raison de la nature du pipeline et de la faible latence, la technique de commutation par *wormhole* est préférable dans la conception du routage [138].

Trafics et métriques : Les trafics de NoC peuvent être classés en deux catégories : les trafics synthétiques et les trafics réels d'application. Les trafics réels sont des traces provenant de charges de travail d'applications réelles. On peut citer par exemple les télécommunications, les réseaux et les applications grand public de la suite de benchmarks E3S [52]. D'autre part, les trafics synthétiques sont des trafics expérimentaux utilisés pour évaluer l'architecture de communication et ils tentent d'imiter des comportements particuliers des trafics d'application du monde réel. Le trafic synthétique peut être régulier ou irrégulier. Un exemple de trafic régulier est le modèle de trafic aléatoire (ou trafic uniforme), où chaque nœud communique avec tous les autres nœuds avec une probabilité d'envoi égale. Le *hotspot* est un exemple de modèle de trafic irrégulier, où tous les nœuds communiquent avec un même nœud, i.e. le nœud *hotspot* du réseau. Un trafic irrégulier crée plus de contention dans le réseau par rapport à un trafic régulier, ce qui crée un goulot d'étranglement de communication dans le réseau, se rapprochant de trafics réels (e.g. accès mémoire). Cependant, ces trafics synthétiques peines à représenter fidèlement les trafics réels, menant à de mauvais dimensionnements des NoC conçus uniquement sur la base de ces trafics [15]. Afin d'évaluer les performances d'un NoC à supporter un trafic, on définit la métrique de latence qui désigne le temps moyen que met un message à arriver à destination après son envoie sur le réseau (i.e. injection sur le nœud source). La latence dépend de la contention du réseau et de la distance entre les nœuds de source et de destination du message. On affiche généralement la latence en fonction de la charge du trafic (souvent définie par le *taux d'injection*) afin de comparer les performances de différents NoC. Lorsque le taux d'injection est modifiable, comme notamment avec les trafics synthétiques, il est possible de tracer la courbe de saturation d'un NoC. Cette courbe permet de visualiser le seuil de saturation, i.e. le taux d'injection limite à partir duquel la latence augmente théoriquement à l'infini (réseau saturé). La figure 2.7 présente une courbe

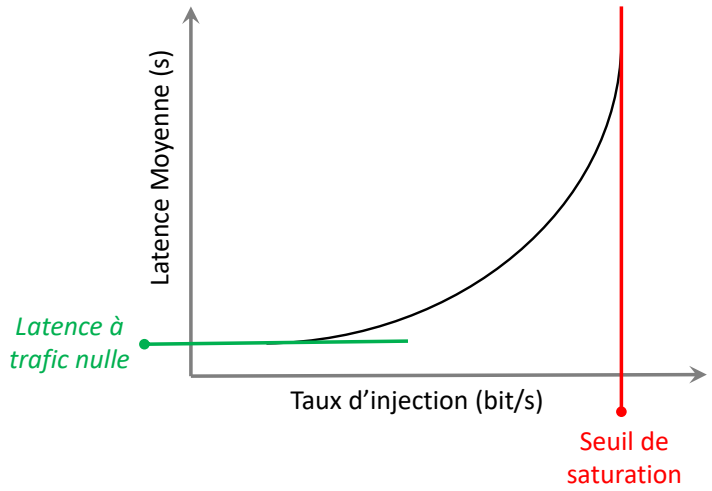


FIGURE 2.7 Courbe de saturation d'un réseau.

typique de saturation, la définition du seuil de saturation qui caractérise les performances maximales d'un NoC pour un trafic donnée, et la latence pour un trafic nul qui définit la latence minimale atteignable sur le réseau.

2.5.2.2 Optimisation des NoC

La charge du trafic traversant les NoC dans un environnement réaliste est généralement déséquilibrée [69]. Ces variations de trafic peuvent résulter : 1) de l'hétérogénéité du SoC, c'est-à-dire du type d'éléments interconnectés du SoC (System on Chip) (e.g. CPU, GPU, contrôleur de mémoire, etc.) [103], et 2) du workload de l'application elle-même s'exécutant sur un SoC hétérogène ou homogène, i.e. communications inter-tâches [169]. Par conséquent, une conception de NoC efficace doit tenir compte des caractéristiques du trafic afin d'éviter un surdimensionnement du NoC coûteux en énergie.

De nombreuses recherches ont été menées sur l'optimisation de l'efficacité énergétique des NoC. Elles peuvent être classées en deux groupes :

— Optimisation dynamique : comprend toutes les solutions adaptatives qui modifient les caractéristiques du NoC au moment de l'exécution. Cela inclut les algorithmes de routage adaptatifs, la gestion avancée de l'alimentation et le contrôle de flux [95, 144, 143].

— Optimisation statique : comprend toutes les optimisations hors ligne qui visent à adapter la conception du NoC aux spécificités du SoC ciblé. Cela englobe les algorithmes de routage déterministes et les méthodologies de conception [123, 4].

Les deux approches présentent des avantages et des inconvénients. L'optimisation dynamique permet souvent d'ajuster au mieux les ressources NoC utilisées à la charge de trafic courante.

Malheureusement, l'adaptation en-ligne nécessite des moyens de surveillance et d'exploitation complexes qui peuvent détériorer les performances du NoC en raison de la latence de réveil, par exemple. L'optimisation statique est, par définition, moins flexible. La complexité de l'optimisation n'est plus gérée au moment de l'exécution mais pendant la conception du NoC. Pour faire face à cette tâche complexe, les concepteurs de NoC doivent réduire la multiplicité des problèmes menant à des configurations de NoC hétérogènes non optimales.

Cette axe de recherche aborde la question de la conception de NoC hétérogènes optimisés avec des performances quasi optimales. En particulier, il est question d'exploiter les IA génératives afin de couvrir de grands espaces de conception où les méthodologies classiques de recherche exhaustive ont échoué en raison des exigences de temps de calcul inabordables.

2.5.3 CAO et IA générative

Il existe un grand nombre de techniques d'apprentissage et d'usage de ces techniques. De par le problème défini précédemment, nous nous intéressons aux techniques d'apprentissage pour la CAO (Conception Assistée par Ordinateur). Il est donc question de générer des designs de NoC optimisés, à l'aide de techniques d'IA permettant d'explorer un espace de conception très large. Pour cela, une famille de technique nous semble pertinente : les IA génératives.

Les IA génératives sont apparues durant la dernière décennie et ont montré des capacités impressionnantes pour construire des modèles précis via un apprentissage non-supervisé. À partir de cette famille de techniques issue de l'apprentissage profond, deux principales architectures sont à différencier : les auto-encoder variationnel (VAE pour *Variational Autoencoder*) [88] et les réseaux antagonistes génératifs (GAN pour *Generative Adversarial Networks*) [67]. La première architecture est généralement utilisée pour extraire des caractéristiques d'un ensemble de données. Quant aux GAN, ils sont utilisés pour générer de nouvelles données, permettant entre autres d'enrichir la base de données (*Data Augmentation*) et d'explorer un espace de données. L'architecture des GAN s'est montrée efficace dans différents domaines pour de la génération de données ayant des caractéristiques similaires à celles du dataset d'origine. En effet, de la génération de portraits photoréalistes [85] aux applications médicales [161], les GAN montrent toujours d'impressionnantes capacités d'apprentissage, comme en atteste la figure 1.4.

L'utilisation des GAN semble donc être une voie pertinente à explorer pour la construction d'un outil de CAO servant à la création de NoC optimisés.

Réseaux antagonistes génératifs : Un GAN est une architecture de réseau de neurones proposée pour la première fois en 2014 par Ian J. Goodfellow et al. [67]. Comme illustré sur

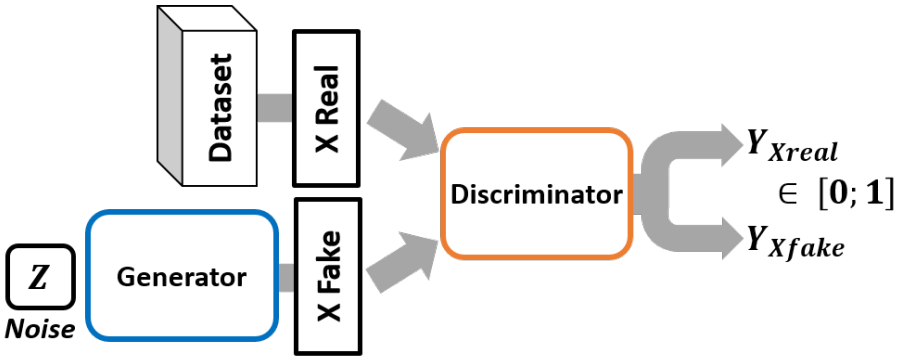


FIGURE 2.8 Schéma représentatif d'un GAN.

la figure 2.8, un GAN est constitué d'un *générateur* et d'un *discriminateur*, deux réseaux de neurones antagonistes. Le discriminateur est un réseau de neurones ayant pour objectif de distinguer correctement les données provenant de l'espace des données réelles, des données fausses générées par le générateur. Et donc le générateur est un réseau de neurones génératif qui apprend à générer des données dans le domaine des données réelles, de sorte que le discriminateur les classe comme réelles. Comme décrit sur la figure 2.8, le discriminateur prend en entrée les données réelles et les données générées (respectivement *X Real* et *X Fake*). Les données réelles sont extraites d'un dataset d'entraînement, et les fausses proviennent de la sortie du générateur. Étant équivalent à un classificateur binaire, il en sort une valeur de probabilité $Y \in [0;1]$. Plus Y est proche de 1, plus la donnée d'entrée est jugée réaliste par le discriminateur. Sur la figure 2.8, on distingue deux sorties : Y_{real} et Y_{fake}, respectivement les sorties du discriminateur pour les données réelles et les données générées.

Durant l'entraînement, la progression de l'un induira la progression de l'autre. Il y a donc une amélioration conjointe comparable à deux joueurs opposés l'un à l'autre. In fine, nous sommes intéressés par le générateur, qui est censé produire des données réalistes.

Les deux réseaux sont donc entraînés simultanément et de façon indépendante. Le discriminateur suit un entraînement supervisé, où les données provenant de la base de données sont étiquetées comme *réelles* et celles provenant de la sortie du générateur sont étiquetées comme *fausses*. En parallèle, le générateur suit un entraînement non-supervisé, où son unique but est que le discriminateur reconnaisse ses données générées comme *réelles*. Le générateur prend en entrée un vecteur aléatoire z issu d'un espace aléatoire Z, appelé *Noise*. À partir de ce "bruit", il apprend à produire de fausses données. Pour formaliser cela, notons G et D les modèles générateur et discriminateur. On note x la donnée réelle en entrée du discriminateur. Nos deux modèles ont chacun un objectif qui leur est propre, formant ainsi ce qui peut s'apparenter à jeux à deux joueurs appliquant la règle *minimax* décrite en Eq. 2.1

[67] :

$$\min_{G} \max_{D} \left(\mathop{\mathbb{E}}_{x \sim p_{data}(x)} [log(D(x))] + \mathop{\mathbb{E}}_{z \sim p_z(z)} [log(1 - D(G(z)))] \right) \tag{2.1}$$

où $\mathbb{E}[X]$ décrit l'espérance de X, p_{data} est la distribution de la base de données, p_z est la distribution du bruit d'entrée et la notation $y \sim p_y(y)$ décrit une variable y de densité de probabilité p_y.

Ainsi, le générateur apprend à générer des données pour tromper le discriminateur, et le discriminateur apprend à différencier correctement ses entrées. Au cours de l'entraînement, ce processus convergera vers un jeu à somme nulle. Chaque amélioration de l'un des réseaux provoquera une détérioration de l'apprentissage du second.

Un tel mécanisme d'apprentissage s'avère particulièrement sensible durant la période d'entraînement. De ce fait, la convergence de l'apprentissage est souvent considérée difficile à obtenir. Pour améliorer cela, Arjovsky *et al.* proposent le Wasserstein GAN (WGAN) [14] utilisant la distance de Wasserstein comme fonction de perte, présentée sous le nom de loss de Wasserstein (W-loss). Dans ce modèle, le discriminateur ne se comporte plus comme un classificateur binaire. En effet, en utilisant le W-loss, la sortie de ce bloc n'est plus comprise entre [0, 1] comme dans le GAN conventionnel, mais entre $[-\infty, +\infty]$. Cette nouvelle sortie peut s'interpréter comme une mesure du "réalisme" ou "irréalisme" des données évaluées. De part ce changement de concept, le réseau discriminateur est renommé réseau critique pour garder une dénomination cohérente, et sera noté C.

Cependant, pour garantir sa stabilité, le W-loss doit satisfaire la contrainte de Lipschitz [175]. Cette contrainte était initialement respectée grâce à une technique de bornement des pondérations (i.e. *weight clipping*) [14]. Malheureusement, cette technique brutale restreint les capacités d'apprentissage du modèle en limitant la plage de valeurs des poids du réseau de neurones. Ainsi, Gulrajani *et al.* ont proposé le *WGAN-GP* [70], améliorant la façon dont est garantie la contrainte de Lipschitz dans le WGAN. Leur méthode consiste à inclure une pénalité de gradient (*gradient penalty*, GP) dans l'entraînement. La fonction de perte du générateur ne change pas par rapport à celle du WGAN, et la fonction de perte du critique (L_C) est modifiée comme suit :

$$L_C(x_i, G(z_i)) = \underbrace{C(G(z_i)) - C(x_i)}_{\text{original critic loss}} + \underbrace{\alpha(\|\nabla_{\hat{x}_i} C(\hat{x}_i)\|_2 - 1)^2}_{\text{gradient penalty}} \tag{2.2}$$

où G et C sont les modèles générateur et critique, ∇ est l'opérateur de gradient usuel, x_i et z_i représentent respectivement un élément de la base de données et de l'espace aléatoire, $\hat{x}_i = \varepsilon x_i + (1 - \varepsilon) G(z_i)$ avec $\varepsilon \sim U[0,1]$ un nombre aléatoire. Le coefficient α a pour valeur 10, d'après le papier d'origine [70].

Cette architecture améliorée de GAN sera celle utilisée comme point de départ pour construire nos propres modèles.

2.6 Problèmes

Nous résumons ici les problèmes adressés dans cette thèse, suite au contexte développé précédemment.

2.6.1 Optimisation en temps réel d'applications OpenMP via le RL

Le développement de méthodes d'optimisation du runtime sur des systèmes de calcul parallèles expose les outils de gestion à un grand nombre de paramètres. Par exemples, pour un système multi-cœurs hétérogène exécutant un calcul parallélisé on retrouvera la liste suivante, non-exhaustive, de paramètres de configuration : nombre de CPU, type de CPU, vitesse de cœur de CPU, taux de parallélisme, niveau hiérarchique de l'accès mémoire, nombre de threads, etc. Trouver l'ensemble optimal de paramètres pour un contexte spécifique n'est pas une tâche triviale, c'est pourquoi l'essor des techniques d'apprentissage nous intéresse particulièrement.

Comme le démontre l'étude de S. Memeti *et al.* [102], les techniques d'apprentissage ouvrent des perspectives d'amélioration des méthodes d'optimisation du calcul parallélisé. De la conception à l'exécution du workload, l'IA permet de faciliter l'exploration et la sélection des paramètres d'optimisation. Dans cet axe, nous ciblons l'optimisation de l'exécution de calculs parallèles et nous nous intéresserons plus particulièrement aux techniques d'apprentissage par renforcement permettant d'implémenter des solutions adaptatives et automatiques.

Enfin, la portabilité des solutions d'optimisation étant un enjeu majeur de la conception de méthodes de contrôle, nous choisissons d'étudier les calculs parallélisés avec OpenMP. En effet, ce modèle de programmation est de loin le plus utilisé, supportant différents langages de programmation, et supporté par la majorité des plate-formes d'exécution. Ainsi, en agissant au niveau du runtime OpenMP, nous garantissons la portabilité des solutions proposées. De plus, cela nous permet de tirer profits du planificateur OpenMP, supportant différent mode de planification dont le mode dynamique conçu pour réduire le temps d'exécution d'un calcul parallèle, pour un coût supplémentaire de temps de planification négligeable.

Dans cet axe de recherche, nous adressons le problème du contrôle dynamique de l'exécution d'un workload OpenMP parallèle pour l'optimisation de l'efficacité énergétique du système. Nous choisissons d'explorer l'apprentissage par renforcement afin de proposer

une solution adaptative pour la prise de décision. Ainsi, nous allons tenter de répondre aux interrogations suivantes :

— *Comment exploiter les chunks pour suivre l'évolution de l'efficacité énergétique d'applications OpenMP ?*

— *Comment exploiter les techniques d'apprentissage par renforcement pour construire un contrôle adaptatif de calculs parallélisés avec OpenMP ?*

Nous précisons notre positionnement sur cet axe de recherche dans la section 3.1 du chapitre 3. Les solutions apportées sont développées dans le chapitre 4.

2.6.2 Conception de réseaux sur puce optimisés via les GAN

L'optimisation des designs de NoC est un enjeu majeur dans la conception de SoC optimisés. Entre autres, nous remarquons que l'optimisation de la consommation énergétique des systèmes sur puce doit impérativement passer par l'amélioration des NoC, à l'origine d'une partie non négligeable de l'énergie consommée par le système. Cependant, en considérant des topologies irrégulières et des NoC hétérogènes, l'espace de conception de ces réseaux de communication croît exponentiellement avec respectivement la taille du réseau (corrélée au nombre d'éléments du SoC) et le nombre de paramètres considérés. Aussi, nous devons nous confronter à la problématique précédente : comment rechercher une solution optimale lorsque l'espace de conception est bien trop grand pour être étudié de manière exhaustive, sachant que les simplifications liées à la recherche analytique introduisent un biais trop important pour l'obtention d'une solution fortement optimisée.

Ainsi, une solution permettant de réduire automatiquement l'espace de conception selon des critères d'optimisation des NoC est nécessaire pour permettre de concevoir des NoC optimisés.

Dans cet axe, nous tenterons de répondre aux interrogations suivantes :

— *Comment accélérer la recherche de solutions optimales en réduisant l'espace de conception ?*

— *Comment diversifier/multiplier les objectifs d'optimisation de cette réduction de l'espace de conception ?*

Nous précisons notre positionnement sur cet axe de recherche dans la section 3.2 du chapitre 3. Les solutions apportées sont développées dans les chapitres 5 et 6.

Chapitre 3

État de l'art

Sommaire

3.1 Efficacité énergétique du calcul parallèle

Les travaux existant ayant pour objectifs d'améliorer l'efficacité énergétique de l'exécution de calculs parallèles sont nombreux. En effet, différentes approches sont possibles, allant de l'optimisation en amont du code définissant le calcul, à l'adaptation dynamique des ressources de calculs durant l'exécution du calcul. Comme décrit dans le chapitre précédent, section 2.4, nos travaux s'orientent vers cette dernière approche dynamique par le biais de l'apprentissage par renforcement. Nous étudions en particulier les applications parallélisées avec OpenMP, précisant ainsi notre positionnement.

Les approches existantes visant à optimiser l'efficacité énergétique des systèmes de calcul s'appuient sur diverses techniques de conception déjà étudiées dans la littérature [111, 74, 125, 20].

3.1.1 Les leviers d'optimisation et leur contrôle

Les solutions industrielles se concentrent essentiellement sur l'amélioration des processeurs et de leurs pilotes. Les processeurs contemporains (i.e. multi-cœurs) supportent un ensemble de fonctionnalités tournées vers l'amélioration de l'efficacité énergétique. Parmi ces fonctionnalités, on retrouve chez Intel la gestion des *P-states* par cœur (PCP)[74], la modulation de la fréquence des composants hors des cœurs (UFS)[74], les fréquences "turbo" efficaces (EET) [21], les *C-state* des cœurs [152], le plafonnement de la puissance [146], ou encore les limitations thermiques (i.e. *thermal design power*(TDP)) qui permettent de limiter la consommation énergétique tout en assurant l'intégrité du système. Ces leviers d'optimisation se retrouvent tous, ou en partie, dans tout système de calcul. Plus généralement, on distingue deux principales fonctionnalités [20] : 1) la mise à l'échelle dynamique de la tension et de la fréquence – i.e. DVFS –, et 2) la gestion dynamique de l'alimentation – i.e. DPM.

Ces fonctionnalités sont deux techniques matérielles largement utilisées pour réduire la consommation d'énergie du CPU. Elles sont toutes deux contrôlées par le système d'exploitation (OS). En effet, la gestion des C-states et P-states pour Intel, et plus globalement le DVFS et DPM, se fait par le biais de pilotes (e.g. intel_pstate et acpi-cpufreq sur Linux) gérés par le système d'exploitation, et accessible à l'utilisateur par des interfaces dédiées. On parlera de gouverneur pour parler des différents modes de gestion disponibles sur ces pilotes (e.g. *ondemand* [127]). De récentes architectures tendent à ramener ce contrôle au niveau du processeur. Cependant, le contrôle au niveau du processeur montre d'importantes limitations, notamment pour la gestion du calcul parallèle, puisque seule les informations relatives aux cœurs considérés sont disponibles au contrôleur. Ainsi, les constructeurs misent sur un contrôle hybride où les processeurs agissent en fonction de données fournies par l'OS. Par exemple, comme mentionné par R. Schöne *et al.* [151], si on étudie l'évolution du *Hardware Power Management* (HWPM) de Intel on remarque qu'entre l'architecture Broadwell et Skylab-SP, le HWPM a perdu en autonomie pour plus de flexibilité et de collaboration avec l'OS. Ainsi, bien qu'un contrôle au niveau du processeur possède beaucoup d'avantages comme les délais de prise de décision réduits et l'interruption limitée des workloads, une intervention au niveau de l'OS est nécessaire pour améliorer la qualité du contrôle.

Enfin, un dernier levier d'optimisation est l'allocation des ressources à une tâche. Cette fonctionnalité est particulièrement utile pour la gestion de calcul parallèle puisqu'elle permet d'optimiser la façon dont le calcul est parallélisé. Cependant, les contrôles implémentés sur processeur par les constructeurs n'agissent pas encore sur ce levier (i.e. contrôle par cœur, donc aucune notion d'application parallélisée), et les gouverneurs n'exploitent pas ce potentiel. En effet, l'allocation des ressources reste à la charge de l'OS, et dépend

essentiellement du code source des applications spécifiant, entre autres, le nombre de threads à créer pour la parallélisation du calcul. Cependant, le nombre optimal de threads dépendra du système de calcul. De plus, selon le système de calcul, le type de ressources utilisées a un impact majeur sur l'efficacité énergétique (e.g. architecture hétérogène).

Ainsi, la conception de gouverneurs permettant d'optimiser l'efficacité énergétique de l'exécution d'un workload est une solution adoptée par beaucoup dans la littérature (voire section 3.1.2 pour optimiser les systèmes de calcul.

3.1.2 Les solutions de type "gouverneur"

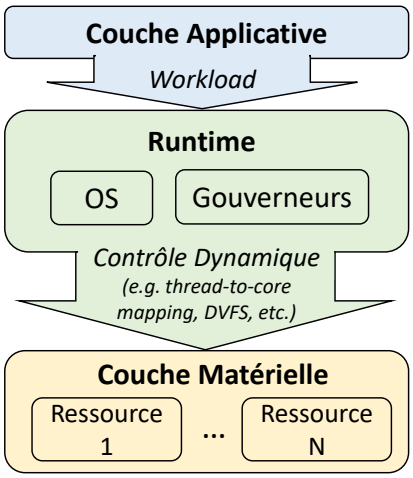


FIGURE 3.1 Modèle d'exécution simplifié d'une application.

Nous appelons "gouverneur" tout système de contrôle intervenant sur le système de calcul au niveau du runtime. La figure 3.1 décrit un modèle en trois couches illustrant les différents niveaux d'exécution d'une application. On y distingue la couche applicative comprenant la description haut-niveaux des tâches et de la charge de calcul, la couche du runtime où interviennent les contrôles temps réel du système d'exploitation et des gouverneurs, et la couche matérielle où le calcul est réparti pour être traité par les ressources disponibles. Ce sont donc les solutions opérant au niveau de la couche du runtime qui nous intéressent. Les actions des gouverneurs peuvent donc se distinguer selon quatre catégories : 1) DVFS, ou contrôle en fréquence, 2) DPM, ou contrôle en puissance, 3) Mapping, ou allocation des ressources, et 4) Scheduling, ou ordonnancement des tâches. La littérature comporte de nombreuses solutions de contrôle s'apparentant à des gouverneurs. Nous distinguerons donc les gouverneurs selon les critères suivants :

— **Type d'action,** ou comment le contrôle proposé agit sur le système. e.g. DVFS, mapping.

— **Méthode *online*,** ou quelle méthode de contrôle est employée durant l'exécution d'un workload. e.g. RL.

— **Activité *offline*,** ou quelle charge de travail est nécessaire en amont du contrôle. e.g. entraînement d'un modèle.

— **Besoins,** ou quelles sont les conditions indispensables à l'implémentation des solutions.

— **Portabilité,** ou quels sont les systèmes ciblés par cette méthode.

Un tableau comparatifs 3.1 des solutions existantes dans la littérature est proposé. Ce tableau propose de mettre en évidence les différentes lacunes existantes dans l'état de l'art. La liste des contributions n'est pas exhaustive, cependant les contributions sélectionnées l'ont été de façon qualitative, afin de correctement reproduire les tendances de l'état de l'art. En effet, ce sujet est un *hot topic*, ce qui résulte en un grand nombre de contributions.

Après une description de ces différentes méthodes, nous développerons notre analyse autour de trois caractéristiques essentielles qui nous intéressent : la portabilité des solutions, les actions menées et l'usage de l'apprentissage par renforcement.

3.1.2.1 Vue d'ensemble des méthodes existantes

Dès 2012, R. Ye *et al.* [170] se penchent sur le potentiel de l'apprentissage par renforcement pour réaliser un contrôle en-ligne ne nécessitant pas d'analyse hors-ligne. Leur contrôle consiste en un DPM afin de gérer de façon optimale les périodes d'inactivité des cœurs. Adressant ce problème pour les systèmes multi-cœurs, leur technique d'apprentissage se base sur un réseau de neurones (Deep Q-Learning) afin de supporter le large espace d'états et d'actions. Leur travaux montrent des résultats prometteurs, cependant ils ne se concentrent que sur le DPM. Or, nous pensons qu'un contrôle hybride adressant différents leviers tels que le DVFS et le mapping est nécessaire pour un contrôle optimal. De plus, leur contrôle se concentre sur l'aspect matériel (état des cœurs), sans exploiter de corrélations avec l'application en cours de traitement.

PoGo [100] est proposé par L.A. Maeda-Nunez *et al.* pour réaliser un contrôle DVFS *application-specific*. Ce travail met le suivi d'exécution du calcul parallèle au centre du contrôleur, permettant l'amélioration de l'efficacité énergétique d'une application particulière. Un apprentissage par renforcement est utilisé pour décider des modifications du DVFS. Cependant, leur application ne concerne pas un système multi-cœurs, ce qui rend leur champ d'actions limité à 4 possibilités. Or, nous ciblons des systèmes multi-cœurs au nombre d'actions plus vastes [170]. De plus, leur système de contrôle récupère l'information sur les performances à partir de *flags* disponibles après modification du code de l'application.

Référence	***Date***	***Actions***	***Méthode online***	***Activité offline***	***Besoins***	***Support d'application***
[170]	2012	DPM	Deep Q-learning (RL)	-	Simulateur	Simulateur multi-cœurs (e.g. Intel Atom 4 et 8 cœurs)
PoGo [100]	2015	DVFS	Q-Learning (RL)	Modification du code d'application	Compteurs matériels	SE (e.g. beagleboard avec Arm Cortex A8)
Sparta [54]	2016	Mapping	Binning-based (*predictor*) Heuristique (*SPARTA Allocator*)	Entraînement du prédicateur (régression linéaire)	Système Hétérogène	Architecture hétérogène (e.g. Arm big.LITTLE et simulation HMP)
[113]	2016	DVFS	Q-learning (RL)	Réglage empirique de la fonction de reward Initialisation des valeurs Q	Taux d'utilisation des buffers	MpSoC (e.g. Arm avec 16 cœurs)
DyPO [73]	2017	DVFS DPM	Classification du workload Sélection de la configuration (*Pareto-optimal*)	Instrumentation des Application Caractérisation de benchmarks Entraînement du classificateur (*régression logistique*)	Compteurs matériels LLVM PAPI	MpSoC hétérogène (e.g. Arm big.LITTLE)
[40]	2017	Mapping	RL	-	Compteurs matériels	Système multi-cœurs (e.g. Intel Xeon E5, 20 cœurs)
[142]	2018	DVFS	EWMA (prédiction du workload) Binning-based (DVFS)	Apprentissage des *Bins*	Compteurs matériels	Processeur multi-cœurs (e.g. Intel Xeon E5/Phi)
AdaMD [18]	2019	DVFS Mapping	Prédiction des performances (régression additive) Binning-based (DVFS), EWMA (prédiction du workload)	Entraînement du prédicateur de performances Apprentissage du classificateur (DVFS)	Compteurs matériels	MpSoC hétérogène (e.g. Arm big.LITTLE)
[72]	2019	DVFS DPM	DQL (RL)	Entraînement de l'oracle Instrumentation des Applications Pré-entraînement offline du DQL	Compteurs matériels	MpSoC (e.g. Arm big.LITTLE)
[101]	2019	DVFS DPM	Régression linéaire (*imitation learning*)	Conception d'oracle (Q-Learning) Approximation de l'oracle (régression) Instrumentation des applications	Compteurs matériels	MpSoC hétérogène (e.g. Arm big.LITTLE)
RLBMCS [145]	2020	Scheduling	RL	-	Free-RTOS	Simulation multi-cœurs
[158]	2021	Scheduling	Deep RL	-	Performance des VM	Simulation IoT
My Work	2020	DVFS Mapping	Deep RL	Entrainement du AE training (si besoin)	Support OpenMP	Système multi-cœurs (e.g. Arm big.LITTLE, Intel Xeon E5)

TABLE 3.1 Comparaison de méthodes récentes de contrôle dynamique du calcul parallèle.

Or, pour faciliter l'implémentation de notre contrôle, nous proposons une solution moins invasive grâce aux chunks.

Sparta [54] présenté par B. Donyanavard *et al.* est un système de contrôle du mapping de tâches sur systèmes multi-cœurs hétérogènes pour améliorer l'efficacité énergétique du calcul. Alors que leur méthode montre des gains supérieurs aux solutions existantes pour le support big.LITTLE testé, leur méthode repose sur des apprentissages hors-ligne et donc non adaptatifs en temps réel. Cette méthode demande donc une conception minutieuse de la politique de contrôle avant exécution, et ne garantit pas une adaptation à toute application exécutée.

Dans [113], A. Molnos *et al.* proposent une solution basée sur du RL pour un contrôle DVFS du système durant l'exécution d'une application impliquant des interfaces externes (I/O), afin de garantir les contraintes de qualité d'exécution (e.g. nombre d'images par seconde traité pour un traitement vidéo continu). Leurs travaux se concentrent sur le réglage optimal de l'apprentissage afin de garantir une convergence rapide du contrôleur DVFS. Cependant, le mapping des tâches n'est pas pris en compte, ce qui pourrait permettre d'élargir le champ d'actions afin de garantir les contraintes de qualité à un moindre coût énergétique.

DyPO [73] proposé par U. Gupta *et al.* est un système de contrôle hybride DVFS et DPM destiné à adapter la configuration du système de calcul en temps réel en fonction des phases d'exécution de l'application traitée. Ce modèle se base sur l'entraînement d'un modèle de classification de workload à partir de données collectées pour un ensemble d'applications. Une classification en-ligne du workload exécuté permet ensuite d'adapter la configuration du système. Contrairement à notre solution, leur système nécessite une instrumentation de l'application à optimiser pour extraire les informations de l'avancement de son exécution nécessaires à la classification. Les auteurs reprennent leur système d'instrumentation dans [72], et proposent de réutiliser leur modèle de classification comme un oracle permettant de pré-entraîner un apprentissage par renforcement. Ainsi, le modèle final bénéficie des connaissances de l'oracle, tout en apprenant en temps réel à s'adapter à de nouvelles applications. Enfin, une dernière amélioration est apportée par les auteurs dans [101] où une technique de transfert d'apprentissage est implémentée pour améliorer l'apprentissage du contrôleur pour le contrôle d'applications inconnues. Cependant, chaque nouvelle application nécessite une instrumentation hors-ligne, contrairement à notre méthode.

Dans [40], G. Chasparis *et al.* proposent un apprentissage par renforcement pour le mapping intelligent de chaque thread d'une l'application exécutée sur un système multi-cœurs. Ce système repose sur les données de compteurs de performance, et ne considère pas le contrôle en fréquence des cœurs du système.

K. R. Basireddy *et al.* [142] proposent une méthode basée sur l'apprentissage hors-ligne de l'attribution de paramètres tension/fréquence en fonction de la charge de calcul courante. Cette charge de calcul est déterminée à partir des compteurs de performance. Leur méthode

est appliquée sur un système multi-cœurs et montre des gains importants en comparaison avec des méthodes existantes. Cependant, leur méthode repose sur un apprentissage hors-ligne de la loi de contrôle à partir d'un benchmark limité. Cela ne garantit pas un contrôle optimal pour tout type de workload. De plus, leur méthode de mesure du workload se base sur les compteurs de performances, et n'est donc pas spécifique à l'application contrôlée comme ce que nous proposons. Ce travail est prolongé dans AdaMD [18], où la gestion du mapping d'applications concurrentes vient compléter leur système de contrôle.

Enfin, D.R. Rinku *et al.* [145] et S. Sheng *et al.* [158] sont deux contributions récentes exploitant l'apprentissage par renforcement respectivement pour des systèmes multi-cœurs exécutant des applications parallélisées avec Free-RTOS, et pour des applications IoT exécutées sur des machines virtuelles. Ces deux contributions traitent uniquement de la planification des tâches (scheduling) et ne considère pas les contrôles DVFS et DPM plus proches du matériel.

Résumé : Cet ensemble de contributions montre la variété des solutions explorées dans la littérature pour l'optimisation dynamique du calcul parallèle. Notre contribution se détache principalement par l'exploitation des chunks qui sont une métrique haut-niveau et *application-specific* disponible sur OpenMP. Cela permet en particulier de profiter de grande diversité de supports et applications utilisant OpenMP, et, avec un minimum d'instrumentation, de mesurer en temps réel la quantité de calcul exécutée. Nous proposons d'exploiter cette métrique pour un contrôle dynamique du mapping et DVFS via un apprentissage par renforcement en-ligne.

3.1.2.2 Portabilité

Nous avons constaté dans un premier temps que les solutions proposées sont généralement conçues pour un seul système (e.g. Odroid XU3 et architecture big.LITTLE [73, 54, 18, 101]), et demandent donc des efforts significatifs pour être adaptées à un nouveau système. Il faut noter le nombre important de contributions dédiées aux systèmes embarqués et plus généralement aux MPSoC, e.g. [160, 173, 100, 73, 18, 72, 54, 101]. Cependant, le problème de l'efficacité énergétique concerne tout type de système de calcul.

C'est donc ce premier point que nous avons décidé d'adresser. Pour cela, nous proposons une méthode de contrôle basée sur le runtime OpenMP qui est supporté par la majorité des OS et architectures. Ainsi, notre solution peut s'appliquer sur tout système supportant OpenMP, allant du système embarqué au serveur HPC, du multi-cœurs au many-cœurs.

Dans les travaux suivants de l'équipe ADAC [122, 47], il est question d'optimiser l'allocation des ressources sur des architectures hétérogènes à partir d'informations fournies

directement depuis la compilation d'applications. Cela à l'avantage de bénéficier des informations sur les caractéristiques des applications, ainsi que d'automatiquement instrumenter le programme. Agir au moment de la compilation favorise la portabilité de la solution. A noter l'utilisation de RL dans [122] pour optimiser l'allocation dynamique de tâches. Cependant, les solutions proposées n'adressent que le mapping de tâches, sans considérer des actions dynamiques telles que les variations de la fréquence de fonctionnement ou encore des phénomènes dynamiques tels que la concurrence d'application.

Une approche qu'il convient aussi de mentionner dans cet état de l'art est le travail de Alessi *et al.* [7]. Les auteurs ont proposé une approche spécifique à OpenMP qui consiste à étendre l'API OpenMP existante avec une API spécialisée pour l'économie d'énergie. Elle permet de prendre des décisions directement au moment de l'exécution pour minimiser l'énergie. Cependant une modification du code des applications est nécessaire pour utiliser cette méthode, et cet outil ne semble pas être maintenu à jour.

3.1.2.3 Actions et activités menées

Une majorité des techniques de contrôle visant à améliorer l'efficacité énergétique proposées dans la littérature adresse le DVFS et le DPM [101, 72, 18, 142, 73, 113, 100, 170, 23, 141, 16]. D'autres considèrent l'allocation des ressources de calcul aux tâches et l'optimisation de leur ordonnancement [145, 158, 18, 40, 65, 54, 63, 61]. Enfin, il va de soi que le contrôle de la configuration du système (i.e. DVFS, DPM) et de la gestion des ressources (i.e. mapping, scheduling) sont complémentaires. Ainsi, on trouve dans la littérature des contributions adressant plusieurs de ces approches avec un même contrôleur, en particulier AdaMD [18] récemment proposé par K. R. Basireddy *et al.*. Dans ce travail, différentes techniques d'apprentissage automatique sont exploitées afin de proposer un contrôle du DVFS ainsi qu'un contrôle de l'allocation thread-cœur. Les résultats exposés montrent des gains importants en efficacité énergétique, puisqu'une amélioration de 28% est obtenue sur la consommation énergétique, tout en respectant les contraintes de performances des applications étudiées. Finalement, dans [62], A. Gamatié *et al.* adressent conjointement la conception de systèmes hétérogènes et l'allocation des workloads pour l'amélioration de l'efficacité énergétique des systèmes de type *edge computing*, ce qui rejoint la tendance actuelle du *domain specific computing* évoquée en introduction.

Nous proposons donc d'adresser le contrôle complémentaire du DVFS et de l'allocation des ressources, similaire à AdaMD [18]. En effet, bien qu'efficace, la solution proposée par K. R. Basireddy *et al.* requiert un travail important en amont du contrôle dynamique pour créer les modules de prédiction et de classification des workloads. En effet, la collecte des données et l'entraînement des modules sont autant de tâches nécessitant du temps CPU. De

plus, l'ensemble des données collectées afin de caractériser les workloads ne garantit pas d'être représentatif de tout type de workload (base de données non exhaustive). Enfin, leur méthode dépend d'un accès aux compteurs matériels du système considéré. Cela demande donc un travail supplémentaire pour intégrer cette solution sur différents systèmes. Ainsi, nous tâcherons de nous différencier de cette méthode en proposant une meilleure portabilité de notre solution (voire section 3.1.2.2), mais aussi des contraintes plus faibles sur l'activité offline nécessaire à la mise en place du contrôle.

3.1.2.4 Intérêt pour le RL

On constate ces dernières années un intérêt croissant pour l'utilisation de l'apprentissage automatique [102, 142, 43], et plus particulièrement de l'apprentissage par renforcement [100, 155, 170, 156].

L'utilisation de l'apprentissage par renforcement (RL) s'avère efficace dans le cadre de problèmes de prise de décision. L.A. Maeda-Nunez *et al.* proposent PoGo [100], une approche adaptative de minimisation de l'énergie utilisée comme gouverneur Linux et testée sur un système embarqué. Un algorithme Q-Learning est implémenté comme unité de décision, et la technique globale démontre de significatives économies d'énergie, par rapport au gouverneur Linux Ondemand existant [127].

Ce travail a été étendu dans [155] où la méthode proposée considère à la fois les changements de workload intra-application et les changements inter-application impliquant des techniques de transfert d'apprentissage. Leur méthode continue à montrer de grandes perspectives même sur les systèmes multi-cœurs avec l'hypothèse d'exécuter une tâche par cœur. Dans [170], une approche basée sur le Q-Learning est proposée pour la gestion des périodes d'inactivité des processeurs multi-cœurs, montrant la polyvalence des applications RL. Enfin, Basireddy *et al.* ont proposé dans [142] une méthode de gestion du temps d'exécution tenant compte de la charge de travail pour économiser l'énergie des systèmes HPC. Ils ont implémenté un algorithme d'apprentissage basé sur le "binning" [98] pour concevoir l'unité de décision, et ont utilisé des compteurs de performance matériels pour la caractérisation du workload.

Récemment, A. K. Singh *et al.* [160] ont proposé une étude complète de l'état de l'art des techniques de gestion dynamique de l'énergie des systèmes embarqués multi-cœurs. Cette étude, plus récente que nos travaux aux moments de leur publication, met en avant, entre autres, la difficulté de prendre en considération un nombre croissant de paramètres. En particulier, les méthodes basées sur du RL et qui reposent sur le Q-learning, et sont donc particulièrement sensibles à ce problème. Or, nous constatons un intérêt particulier pour ces méthodes dans notre étude de la littérature (c.f. table 3.1). En effet, un des avantages majeurs

de ces méthodes est qu'elles permettent la construction d'un contrôleur précis, nécessitant très peu d'effort offline puisque l'apprentissage se fait online, rendant ces méthodes flexibles. On peut constater cela sur la table 3.1, où les contributions utilisant du RL nécessitent peu de travail offline. En effet, hormis [72] qui pré-entraîne son modèle avant de l'implémenter en temps réel, il n'est question que d'initialisation [113] ou d'instrumentation [100] spécifiques aux méthodes proposées, voire aucun travail offline [170, 40, 145, 158]. Cependant, les méthodes classiques de RL se basant sur une table de consultation (i.e. *look-up table*) telles que le Q-Learning [100] se voient limitées par le nombre de paramètres pouvant être pris en compte. Cette limitation est mentionnée également par S. K. Mandal *et al.* [101], expliquant que la taille de la table-Q croît exponentiellement avec l'augmentation des paramètres d'entrée ce qui in fine rend la solution infaisable. Pour profiter des compétences du RL pour la construction de modèles de prise de décisions complexes tout en palliant le problème de la taille des tables de consultation, nous proposons d'exploiter une méthode de deep Q-learning (DQL) [170], qui consiste à remplacer la table-Q par un réseau de neurones capable d'approximer une table-Q complexe.

Un défaut de cet apprentissage online est qu'il va être adapté au contrôle du workload d'entraînement, et sera donc peu flexible pour s'adapter à un nouveau workload. Cependant, des travaux encourageant montrent qu'il est possible de pallier ce défaut à moindre effort, par le biais de techniques telles que le transfert d'apprentissage [155]. Ainsi, le RL reste une solution particulièrement intéressante pour implémenter un contrôle complexe.

3.1.3 Conclusion

Les travaux mentionnés ci-dessus montrent comment l'énergie peut être économisée à l'exécution en réalisant une adaptation en fonction du workload. On constate également un intérêt certain pour l'usage des techniques d'apprentissage, et plus spécifiquement pour l'utilisation d'apprentissage par renforcement. Cette méthode permet de construire un système de prise de décision de façon automatique, pour le contrôle de l'exécution de workloads. Cependant, il en ressort une absence de solutions adressant, par le biais d'un même système de contrôle, les différents leviers que sont le DVFS, le DPM et le mapping de l'allocation des ressources. De plus, les solutions basées sur le RL n'adressent pas spécifiquement les workloads OpenMP, mais se positionnent à un plus bas niveau, dépendant de ce fait des compteurs matériels disponibles sur le système de calcul.

Une partie de nos travaux consistera à explorer cette opportunité en montrant un ensemble de résultats prometteurs sur l'amélioration de l'efficacité énergétique par une allocation des ressources appropriée à l'exécution basée sur l'apprentissage par renforcement.

Nous proposons donc de construire un contrôleur à partir d'apprentissage par renforcement. Notre système repose sur des données extraites du runtime OpenMP afin d'avoir une solution multi-niveau et portable. Les actions du contrôleur comprennent le DVFS et l'allocation de ressources.

3.2 Conception de NoC optimisés au niveau matériel

Nous nous intéressons ici aux défis de la conception matérielle de NoC optimisés, que nous distinguons des défis de contrôle et de gestion [66] concernant le niveau logiciel des NoC.

La performance et l'efficacité du NoC dépendent fortement de la conception matérielle de l'interconnexion. Les routeurs sont les composants actifs qui ont un impact significatif sur la latence et le débit de la communication sur le réseau sur puce. De même, les liens supportant le trafic et transmettant les données entre les routeurs sont en partie responsables des caractéristiques de performance (e.g. bande passante) des NoC. Ainsi, une allocation efficace des ressources de routage et une optimisation de l'interconnexion peuvent améliorer les performances des applications SoC. Par conséquent, l'optimisation de la topologie du réseau, ainsi que la personnalisation des composants des NoC (i.e. routeurs et connexions) via des designs de réseaux sur puce hétérogènes sont deux leviers qui doivent être considérés lors de la conception de NoC optimisés.

Dans la suite de cette section, nous citons tout d'abord les approches dites "classiques" (i.e. sans IA) visant à améliorer la topologie et la composition des NoC. Ensuite, comme évoqué dans la section 2.5.2.1, les NoC peuvent être décrits comme des graphes. Nous nous intéressons donc aux approches de conception de graphes optimisés en exploitant cette analogie. Enfin, avant de conclure sur cet état de l'art, nous présentons les approches existantes de conception de NoC exploitant les techniques d'apprentissage.

3.2.1 Méthodologies de conception classiques

3.2.1.1 Optimisation de la Topologie

Topologies régulières : Les topologies régulières de NoC ont déjà été étudiées dans différentes contributions [27, 49, 128]. En particulier, I. A. Alimi *et al.* décrivent dans [9] les avantages et inconvénients d'une dizaine de topologies régulières différentes. Ces topologies comprennent entre autres des topologies 2D telles que le mesh, le torus, le ring, le star et le binary tree, mais aussi des topologies 3D telles que le cube et l'hypercube. Cette variété de topologies différentes montre la complexité de la conception de NoC. En effet, il n'existe pas

de topologie unique surpassant en tout point les autres topologies existantes. De ce fait, selon l'utilisation et l'architecture du SoC sous-jacent, le choix de la topologie aura un impact important.

Ainsi, la conception d'un NoC demande une attention particulière sur la topologie du réseau. De ce fait, la personnalisation de la topologie peut être étendue jusqu'à la considération de topologies irrégulières, plus flexibles.

Topologies irrégulières : Les topologies irrégulières sont basées sur l'intégration de diverses formes, généralement des structures régulières, selon différentes modalités. Ainsi, une approche hybride, hiérarchique ou asymétrique peut être adoptée. Les topologies irrégulières visent à augmenter la bande passante disponible par rapport aux topologies régulières, en s'adaptant aux trafics considérés. En effet, comme décrit dans [163], l'utilisation de topologies irrégulière permet de réaliser des NoC adaptés aux contraintes du SoC (i.e. applications et architecture) en optimisant, entre autres, la distance entre les routeurs (e.g. [124]). Cependant, le routage de topologies irrégulières est un défis pour le concepteur du NoC, puisque les algorithmes déterministes telles que le routage XY s'avèrent inutilisables [124], demandant alors un routage dynamique complexe pour garantir le bon fonctionnement du NoC (e.g. sans inter-blocage) .

Résumé : La topologie du réseau détermine la manière dont les nœuds sont connectés dans le réseau. Bien que plusieurs topologies NoC existent dans la littérature, seules quelques-unes ont été implémentées dans des puces industrielles [171]. Par exemple, les processeurs de la série Intel Xeon Phi [137] utilisent la topologie ring. La topologie du NoC affecte les performances [42], c'est pourquoi une attention particulière doit être accordée à la sélection d'une topologie appropriée en fonction de l'application. Des recherches supplémentaires sont nécessaires pour déterminer la topologie NoC appropriée pour les systèmes hétérogènes capables d'exécuter différentes applications, comme étudié dans [126]. Ainsi, un outil de CAO approprié permettrait de faciliter cette étape de conception des NoC.

3.2.1.2 Optimisation de la composition des NoC : NoC hétérogènes

L'optimisation des NoC passe par une implémentation optimisée des composants du NoC. Ces composants sont les routeurs et les connexions. Il existe différentes architectures de routeurs (e.g. routeurs *bufferless* [115] , R-NoC [55, 57, 58]) ainsi que différents types de connexions [86] (i.e. différentes bandes-passantes, nombre de canaux virtuels, etc.), ayant des caractéristiques de performance et de consommation énergétique différentes.

Afin d'optimiser au mieux l'architecture d'un NoC en fonction des contraintes auxquelles il est soumis (e.g. type de trafic), le concept de NoC hétérogènes se révèlent comme la meilleure solution. En effet, le NoC hétérogène est un concept flexible où chacun des composants du NoC est déterminé individuellement, afin d'obtenir un NoC global aux performances optimales. Ainsi, alors que la charge du trafic est souvent déséquilibrée [69], un NoC hétérogène pourra être optimisé pour utiliser un minimum de ressources tout en garantissant un niveau de performances du réseau. Dans l'article [110], Mishra *et al.* proposent une

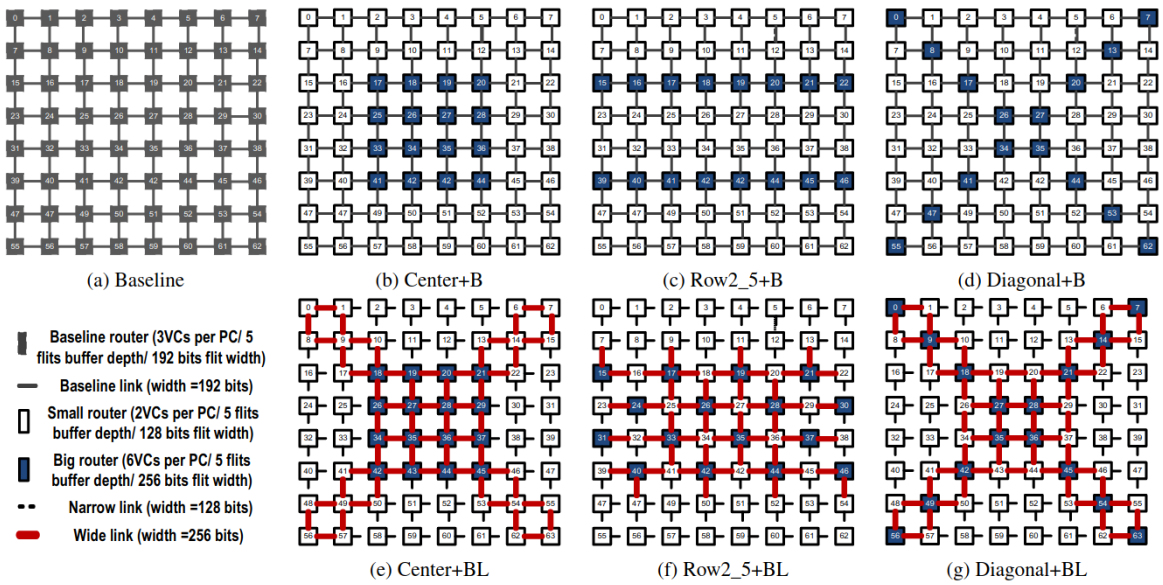


FIGURE 3.2 Proposition de NoC hétérogènes par *HeteroNoC* [110]. source : [110]

conception *HeteroNoC* qui incorpore deux types de routeurs : de gros routeurs avec plus de canaux virtuels (VC, pour *virtual channels*) et des liens à large bande passante, et de petits routeurs avec moins de VC et des liens à faible bande passante. La conception globale est nettement plus performante qu'un réseau homogène équivalent, tout en consommant moins d'énergie. Les NoC hétérogènes proposés sont visibles sur la figure 3.2. Cependant, la proportion de routeurs de chaque type ainsi que leur position sur la grille maillée sont déterminées par une exploration non exhaustive de l'espace de conception, et rien ne garantit que les configurations NoC proposées soient optimales. De plus, la topologie du NoC n'est pas explorée et est fixée à un mesh. Zhao *et al.* [174] ont envisagé la mise en œuvre de routeurs avec et sans mémoire tampon et ont comparé différents placements de routeurs. Tout comme pour *HeteroNoC*, les auteurs ont réduit l'espace de conception de la recherche en limitant les types de routeurs disponibles à seulement deux routeurs différents. L'utilisation du réseau *Clos* au lieu du réseau crossbar habituel dans les routeurs, ainsi que le mélange de routeurs avec et sans buffers sont également proposés par Naik *et al.* dans [118] pour construire un

NoC à commutation de circuits plus efficace. Enfin, dans le document [30], Bokhari *et al.* ont proposé la mise en œuvre de plusieurs architectures de routeur avec diverses propriétés au sein d'un même nœud NoC. Les caractéristiques du nœud peuvent ensuite être sélectionnées au moment de l'exécution en fonction du profil de la charge de travail. Cette méthode permet une conception adaptative du NoC et donc une amélioration de l'efficacité énergétique, mais au prix d'une complexité de contrôle supplémentaire et de matériel additionnel.

Ces travaux soulignent les avantages attendus en termes d'efficacité énergétique de l'utilisation de NoCs hétérogènes. Cependant, ils soulèvent également la question de la méthodologie à appliquer pour concevoir des NoCs hétérogènes optimaux.

3.2.2 Génération de graphes optimisés

Comme évoqué précédemment, la création de NoC est similaire à la création de graphes. Nous faisons donc un premier état de l'art concernant les outils de conception de graphes reposant sur les techniques avancées d'apprentissage.

La génération de graphes via des techniques d'apprentissage profond fait l'objet de précédentes recherches, généralement pour de l'apprentissage de motifs de graphes. Les auteurs de [28] se sont concentrés sur l'identification de motifs dans de très grandes structures de graphes comme les réseaux sociaux. Pour cela, un GAN est proposé, basé sur les réseaux de neurones *Long Short-Term Memory* (LSTM).

Dans [172], les auteurs réalisent de la génération de graphes à partir de la représentation en matrice d'adjacence. Ils définissent un réseau de neurones qui apprend à produire les connexions d'un sommet à ses sommets voisins, appartenant à un même graphe. Dans [60], les auteurs utilisent un WGAN pour produire des graphes labellisés. Ils ont obtenu des résultats prometteurs, au vu de la complexité des données générées. En effet, leur GAN est capable de générer à la fois la matrice d'adjacence et la matrice des labels. Leur travail est inspiré du projet MolGAN [38] où des graphes labellisés sont produits. Dans MolGAN, les graphes représentent des molécules, et les labels indiquent le type des atomes et des connexions présents dans les molécules. L'approche présentée dans MolGAN propose une architecture de GAN possédant un troisième réseau de neurones appelé réseau Reward. Le schéma du GAN final est visible figure 3.3. Ce troisième bloc est implémenté pour guider l'apprentissage du générateur pour converger vers la génération de données appartenant à un sous-espace de solutions correspondantes aux contraintes utilisateurs. L'apprentissage du générateur en fonction du Reward est similaire à un apprentissage par renforcement (RL), et l'entraînement du Reward repose sur l'appel à un logiciel externe durant le processus d'entraînement général. En ce qui concerne la conception de GAN "guidés", on peut aussi citer le travail de Lee et Seoh dans [96, 97]. Ils ont tout d'abord proposé un GAN contrôlable

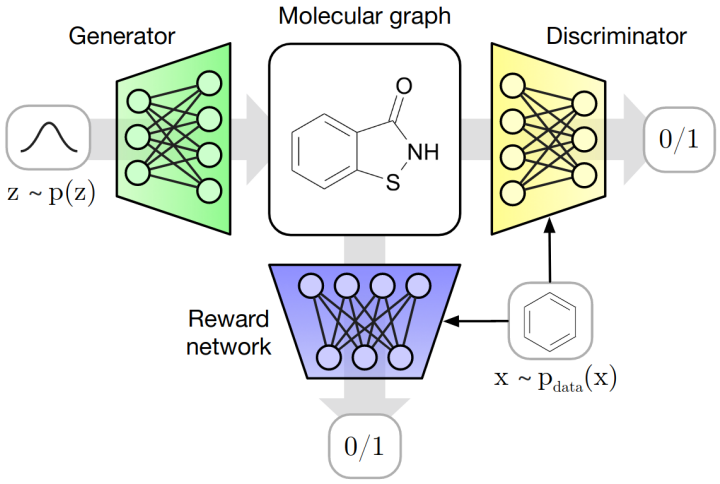


FIGURE 3.3 Architecture de MolGAN pour la génération de molécules. source : [38]

[96] inspiré des GAN conditionnels [109], en utilisant un troisième réseau de neurones comme classificateur. Ils ont ensuite étendu leurs travaux en ajoutant un quatrième réseau [97] qui permet au générateur du GAN de produire des données plus diverses et de meilleure qualité, en se basant sur le *inception score* [149].

3.2.3 Méthodologies de conception des NoC via l'apprentissage automatique

Pour ce qui est de la conception propre aux NoC, via les techniques d'apprentissage, on trouve l'approche MLNoC proposée par Rao *et al.* in [140]. Les auteurs montrent que les techniques d'apprentissage peuvent se révéler très efficace dans la prédiction de descriptions générales de NoC, telles que le type de topologie (i.e. mesh, torus, etc.) ou la méthode de routage, en fonction des propriétés et caractéristiques du SoC. Cependant, MLNoC n'explore que les techniques d'apprentissage supervisé, et aucun travaux existant ne propose d'étudier la génération de topologies de NoC à base d'IA génératives.

Dans [143], Reza *et al.* adressent le problème de conception de NoC hétérogènes efficace énergétiquement en explorant les solutions de contrôle dynamique via de l'apprentissage en-ligne afin d'adapter en temps réel la configuration du NoC.

On retrouve dans la littérature différents travaux exploitant les techniques d'apprentissage pour aider à la conception de NoC. Alhubail *et al.* proposent une méthodologie qui aborde le problème de l'optimisation multi-objectifs de la conception de réseau sur puce, pour des systèmes hétérogènes (CPU et GPU confondus) [8]. Leur solution repose sur l'utilisation d'un algorithme génétique [79] (GA) et un algorithme évolutif de recherche de Pareto-

optimaux (i.e. SPEA2 [176]), chacun exploité à une étape différente du processus de design, pour finalement sortir une architecture de NoC optimisée. Bien que cette contribution soit probablement la plus complète à ce jour, la méthode proposée ne produit qu'une seule solution sans garantir que ce soit la meilleure i.e. utilisation d'approches heuristiques. De plus, les auteurs implémentent un GA pour réaliser de l'optimisation à objectif unique i.e. pour minimiser la latence du réseau, et l'optimisation en terme de puissance consommée (i.e. utilisation du SPEA2) arrive en second plan. Ainsi les différents objectifs d'optimisation sont adressés de manière séquentielle, et n'ont donc pas la même priorité sur la conception du NoC final.

L'utilisation d'apprentissage profond est récurrente dans l'état de l'art. Les capacités de modélisation des réseaux de neurones sont exploitées pour prédire certaines métriques de NoC (e.g. latence moyenne) et aident donc les concepteurs dans leur recherche d'optimisation. En effet, ces modèles permettent des prédictions rapides et précises, rendant possible l'exploration de larges espaces de designs, impossible à réaliser via les simulations classiques beaucoup plus lentes. Pour en citer quelques-unes, K. Rusek *et al.* proposent *RouteNet* [148], un modèle d'apprentissage profond entraîné à prédire les performances d'un réseau, dans le domaine global des réseaux de communication. Ce modèle est utilisé pour de la modélisation et de l'optimisation de réseaux. Une contribution similaire est proposée par R. Kirby *et al.* [90], où le concept de *graph neural network* (GNN) est exploité pour inférer la congestion de design de puce physique à très bas niveau (portes logiques).

3.2.4 Conclusion

L'étude de l'état de l'art de ce second axe de recherche révèle premièrement d'encourageantes perspectives quant à l'utilisation de GAN pour la génération de NoC, aux vues des solutions existantes pour d'autre types de graphes. Ensuite, il s'avère qu'aucune contribution n'ait encore considérée cette piste particulière. Ainsi, il y a ici l'opportunité d'explorer une nouvelle voie pour la CAO de NoC. Enfin, les différents travaux existants dans la littérature témoignent de la difficulté de créer des NoC optimisés, relevant essentiellement de l'ampleur de l'espace de conception à considérer. Ainsi, nous tenterons d'apporter une solution à cette problématique en proposant un outil de réduction de l'espace de conception, vers un sous-ensemble optimisé. Notre solution s'inspire grandement de MolGAN, mais contrairement à la solution proposée dans [38], notre Reward est entraîné en amont de l'apprentissage du GAN, ce qui permet un apprentissage global plus rapide puisqu'il n'est plus nécessaire d'appeler un logiciel (e.g. simulateur de NoC) externe durant l'entraînement. Bien évidemment, les graphes étudiés font référence à des topologies de NoC. À notre connaissance, c'est la première fois que les GAN sont utilisés pour la réduction d'espace de conception de NoC.

Nous proposons ensuite d'étendre ce concept de GAN guidé vers une architecture avec un nombre illimité de Rewards.

Chapitre 4

Efficacité énergétique des calculs parallélisés OpenMP

Sommaire

Ce chapitre présente les travaux effectués sur le premier axe de recherche, tentant de répondre aux questions posées section 2.6.1. Ces travaux sont accompagnés des publications suivantes : [107, 104, 108].

Il est donc question d'optimiser un système de calcul exécutant une application OpenMP parallèle. Cette optimisation se veut dynamique, pour s'adapter au mieux aux besoins de l'application exécutée, et se fait via la modification de la configuration du système. Par configuration, nous parlons d'un ensemble de ressources de calcul attribuées à la tâche, ainsi que de leur fréquence de fonctionnement.

Dans un premier temps, nous proposons une "boîte à outils" permettant de récupérer en temps réel différentes informations sur l'efficacité énergétique d'un système exécutant une application OpenMP, sans requérir aux compteurs matériels de performance qui dépendent du système utilisé. Dans un second temps, nous développons une méthode de contrôle reposant sur une technique d'apprentissage par renforcement. Finalement nous terminons ce chapitre par quelques résultats démontrant l'efficacité de notre méthode, et discutons des perspectives.

4.1 Suivi de l'efficacité énergétique des applications OpenMP

Afin d'adapter le système en fonction de son efficacité énergétique, il est nécessaire d'avoir accès en temps réel à la valeur de cette efficacité énergétique. Pour cela, nous proposons une nouvelle approche, au niveau logiciel, exploitant l'environnement d'exécution (i.e. *runtime*) OpenMP. Dans le but d'étudier différentes architectures de calcul, nous nous sommes intéressés à deux systèmes de natures différentes : une plate-forme Odroid hétérogène de type big.LITTLE, ainsi qu'un serveur Intel Xeon multi-cœurs.

4.1.1 Chunks et métriques associées

Nous proposons ici de définir, à partir des chunks, deux nouvelles métriques permettant de caractériser les performances et l'efficacité énergétique d'une application exécutée sur un système multi-cœurs.

Définition 1 (Chunks par Seconde - CpS) *Le nombre de chunks exécutés en une seconde, où un chunk est un bloc d'instructions attribué à un thread pour exécution. Défini une vitesse de travail, i.e., c'est une métrique de performance.*

Définition 2 (Chunks par Joule - CpJ) *Le nombre de chunks exécutés pour un Joule, où les Joules désignent la quantité d'énergie utilisée par le système de calcul. Peut aussi être défini comme des CpS par Watt ou CpS/W, le nombre de chunks par seconde exécutés par Watt. Défini la quantité de travail par quantité d'énergie, i.e., c'est une métrique d'efficacité énergétique.*

Exemple 1 On considère un code OpenMP simple décrit sur la Figure 4.1. Il consiste à changer la valeur du $i^{\text{ème}}$ élément d'un vecteur B, en lui additionnant le $i^{\text{ème}}$ élément d'un vecteur A. C'est un code simple, parcourant un espace mémoire, faisant un calcul élémentaire sur chacune de ces cellules mémoires.

Les calculs sont donc effectués à l'intérieur d'une boucle *for*, qui correspond au workload à paralléliser. Pour comprendre comment la parallélisation est effectuée, on se réfère à la

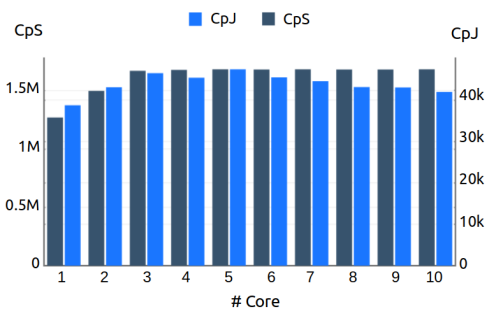
```
00    #include <omp.h>
01  // Initialisation
02    n = 1e9; nthreads = 10;
03    double *A, *B;
04    posix_memalign((void**)&A, 64,
                           n*sizeof(double));
05    posix_memalign((void**)&B, 64,
                           n*sizeof(double));
06    for (i = 0; i < n; ++i) {
07        A[i] = 0.1;
08        B[i] = 0.0;
09    }
10  // Parallélisation
11    omp_set_num_threads(nthreads);
12    #pragma omp parallel for
            schedule(dynamic, 1) // directive OpenMP
13    for (i=0; i<n; i++){
14       B[i] = A[i] + B[i];
15    }
16    return 0;
```

FIGURE 4.1 Exemple d'un code C OpenMP simple.

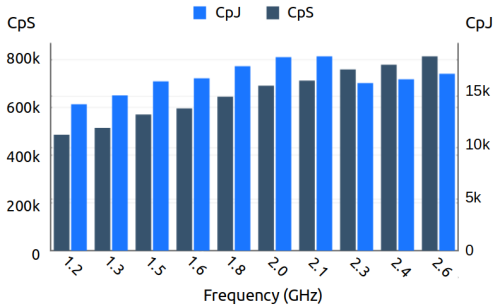
Figure 2.1. La commande OpenMP ligne 12 initialise une région parallèle, où les chunks sont distribués dynamiquement, et un par un (option "schedule(dynamic, 1)"), parmi les threads travailleurs. L'équipe de threads travailleurs est configurée ligne 11, avec ici 10 threads de créés.

Pour l'exemple, nous nous servons ici d'un processeur Intel Xeon disposant de 10 cœurs de calcul. Lorsque nous exécutons ce code sur différentes configurations d'architecture, en termes de nombre de cœurs ou de fréquence de fonctionnement, nous obtenons différentes valeurs de performances et d'efficacité énergétique, reflétées par nos métriques CpS et CpJ, comme montré sur la Figure 4.2.

Sur la Figure 4.2a, la performance (i.e. CpS) augmente avant d'atteindre un plateau à partir de 3 cœurs. Cela s'explique par une saturation de l'accès mémoire, liée directement à la nature exigeante en mémoire de l'application. Au-delà de 3 cœurs de calculs, on constate une dégradation de l'efficacité énergétique. En effet, à partir de ce seuil, la majorité des cœurs se retrouve en attente de données pour exécution, tout en consommant de la puissance. Lorsque c'est la fréquence de fonctionnement qui varie, pour un nombre de ressources de calcul fixe, on observe une amélioration linéaire de la performance, figure 4.2b. L'efficacité énergétique, quant à elle, chute à partir d'une certaine fréquence. Cela correspond au passage vers des fréquences supérieures à la fréquence de base du processeur Intel Xeon (i.e. 2.2GHz), donnant lieu à une augmentation de la puissance consommée.



(a) Configurations ayant différents nombres de cœurs. Fréquence = 1.2GHz.



(b) Configurations ayant différentes fréquences de cœur. #cœur = 1.

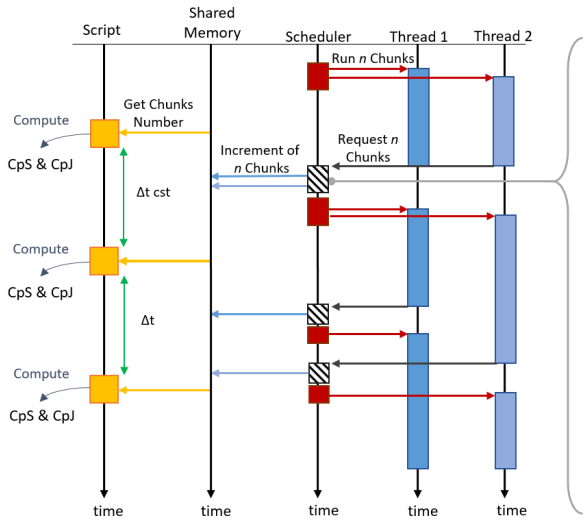
FIGURE 4.2 CpS et CpJ pour différentes configurations du système.

Discussion et hypothèses d'utilisation : La métrique des chunks est définie comme une métrique relative. Selon la terminologie employée dans OpenMP, un chunk correspond à une itération d'une boucle parallèle. Ainsi, nous considérerons uniquement des applications composées essentiellement de boucles *for* parallélisées avec OpenMP. De plus, cette caractéristique (i.e. un chunk est une itération de boucle *for*) fait que la taille d'un chunk varie en fonction de la quantité de travail contenue dans une itération. Nous reviendrons là-dessus par la suite, notamment dans la section 4.2.2 où il est question de différents types de calculs.

Il faut également noter que notre méthode de calcul du CpJ est basée sur la consommation énergétique du système entier. Cela limitera nos expérimentations au contrôle de l'exécution d'une seule application à la fois, de sorte que les chunks représentent la majorité du calcul utile du système. Ainsi, dans nos expérimentations, nous pourrons faire l'hypothèse que la consommation énergétique du système est majoritairement impactée par les chunks de l'application considérée.

4.1.1.1 Méthode d'extraction des chunks

L'implémentation de l'API OpenMP sur GNU est intégrée dans la bibliothèque *libgomp* de GCC. En particulier, le mécanisme de planification responsable de la répartition des chunks parmi les threads y est décrit. Cette bibliothèque est liée à l'environnement d'exécution après compilation. Le suivi des chunks se fait à ce niveau là. La bibliothèque *libgomp* est modifiée pour inclure le mécanisme de collecte automatique des chunks décrit dans la figure 4.3. Ainsi, à chaque allocation de chunks à un thread, un compteur est mis à jour. Ce compteur est enregistré sur une mémoire partagée, créée via l'API IPC (Inter-Process Communication) disponible sur le système d'exploitation utilisé. Les fichiers binaires d'applications liés avec cette version modifiée de GCC/*libgomp* peuvent ainsi réaliser cette instrumentation




```
Update the counter's value, saved in a shared memory

Initialization:
Create a shared memory with the identification number "shmid".
Initialize nb_chunk_done[n]: table of long with n cells, where n
is the number of thread.
Initialize nb_chunk_updated: total number of chunks done

01 Get number of chunk done from the current thread
02 int tid = omp_get_thread_num();// OpenMP directive
03 nb_chunk_done[tid] = Number of chunk done by the tid^th thread.

04 Get time of the capture:
05 act_time = get_time();

06 Update the chunk-counter value:
07 for i in NumberOfThread do
08  nb_chunk_updated += nb_chunk_done[i];

09 Prepare data to write in the Shared Memory:
10 StringToWrite = "nb_chunk_updated" & "act_time";

11 Attached shared memory and write data (ipc/shm
   directives):
12 shared_memory = shmat(shmid, NULL, 0))
13 memcpy(shared_memory, &StringToWrite,
                                  strlen(StringToWrite)+1);
14 Housekeeping...
```

FIGURE 4.3 Diagramme séquentiel de la collecte de chunks (gauche) et méthode de mise-à-jour du compteur de chunks (droite, pseudo-code).

automatique. Il est ensuite possible de suivre en temps réel l'évolution des métriques CpS et CpJ durant l'exécution d'application, en allant lire dans la mémoire partagée depuis un autre processus.

Nous avons donc implémenté une bibliothèque qui permet de créer des scripts ayant accès aux valeurs de CpS et CpJ, et pouvant réaliser un traitement arbitraire sur ces données. En collectant périodiquement le nombre de chunks, le CpS est directement mesurable, et on en dérive le CpJ en accédant à la puissance consommée. Cette donnée de consommation de puissance est accessible de façons diverses selon les systèmes de calcul utilisés. Ainsi, pour la plate-forme Odroid, les données de consommation sont obtenues en lisant directement les sorties des capteurs de courant, alors que pour le serveur Intel, des compteurs dédiés sont disponibles (i.e. compteurs RAPL). Ces derniers sont accessibles soit via l'outil Intel PCM, soit en lisant directement les registres concernés. Ces scripts de collecte peuvent ensuite servir à contrôler dynamiquement différents paramètres du système, tel que la fréquence de fonctionnement ou l'attribution des threads aux ressources de calcul du système (i.e. planification "thread-to-core").

Le concept global est illustré sur la figure 4.3. Plus précisément, elle décrit la lecture périodique de la mémoire partagée par le script de collecte, pendant que l'outil de planification OpenMP (i.e. scheduler) gère, de façon asynchrone, à la fois l'attribution des chunks aux threads et la mise à jour du compteur dans la mémoire partagée. Pour des raisons de robustesse, seul le scheduler est autorisé à écrire dans la mémoire partagée. De ce fait, il n'y a pas à gérer de concurrence entre les threads qui, quant à eux, se contentent de lire dans la

mémoire. Enfin, aucun mécanisme de sécurité de la mémoire, tel que les "mutex", n'est implémenté car la probabilité d'un accès simultané à la mémoire partagée par le script et le scheduler est très faible. En effet, nous avons observé qu'un tel évènement est négligeable, car automatiquement annulé durant l'analyse des données (i.e. donnée aberrante), et leur traitement, e.g. entraînement de réseau de neurones.

Ces métriques sont donc facilement implémentables et du fait qu'elles soient extraites directement au niveau de l'environnement d'exécution d'OpenMP, elles sont exploitables sur tout système et architecture disposant de la librairy *libgomp*, i.e. la majorité des systèmes. De plus, elles peuvent être utilisées durant l'exécution d'une application, ce qui ouvre des perspectives d'analyse en temps réel de l'efficacité d'un système de calcul.

4.1.1.2 Métrique des chunks : Formalisation

Nous venons de présenter la métrique des chunks nous permettant de suivre en temps réel depuis le runtime l'évolution de l'exécution d'un workload. Nous proposons ici de formaliser cette métrique.

Cette formalisation se concentre sur les workloads parallèles, les régions séquentielles sont donc laissées de côté puisqu'elles ne sont pas sujettes à une exécution en parallèle. Un workload parallèle peut être décrit comme un ensemble fini de chunks, pouvant être exécutés aussi bien en série qu'en parallèle. On note $\mathscr{I}$ l'ensemble des instructions supportées par le système d'exécution considéré.

Workloads parallèles : Un workload parallèle $Par = \{C\}$ est un ensemble de chunks $C = \{i_k\}$. Chaque i_k représente une instance d'une instruction appartenant à $\mathscr{I}$.

Architecture : On définit une architecture d'exécution $Arc = (Proc, Mem)$ d'un système de calcul comme une combinaison d'éléments de traitement *Proc* et de ressources mémoires associées *Mem*. On note $\alpha = \{Arc_k\}$ l'ensemble de toutes les architectures possibles.

Une architecture peut être monitorée à l'aide de compteurs matériels et de capteurs de consommation énergétique. On considère $V = \langle v_1, v_2, ..., v_n \rangle$ comme un vecteur dont chaque valeur v_k correspond à une mesure des compteurs matériels et de la consommation énergétique. Par exemple, ces valeurs peuvent être le nombre de *cache misses*, *hits*, d'accès mémoire tels que *read* et *write*, etc. L'ensemble de tous ces vecteurs est noté $\mathscr{V}$.

Définition de l'exécution : On définit la fonction d'exécution $Ex : \mathscr{I} \times \alpha \rightarrow \mathscr{V}$, qui sort un vecteurs de métriques, en fonction de l'exécution d'une instruction sur une architecture donnée.

À partir de cette définition, l'exécution d'un chunk $C = \{i_k\}$ sur une architecture $a \in \alpha$ produit un vecteur $V = \langle v_1, v_2, ..., v_n \rangle$ de la façon suivante : $\forall i_k \in C, Ex(i_k, a) = \langle v_1^k, v_2^k, ..., v_n^k \rangle$ tel que

$$v_1 = \sum_{\forall i_k} v_1^k, v_2 = \sum_{\forall i_k} v_2^k, ..., v_n = \sum_{\forall i_k} v_n^k$$

De ce fait, un chunk est une représentation de la charge de calcul à un degré de granularité plus grossier que celui des instructions. En effet, le résultat de l'exécution d'un chunk correspond à la somme des exécutions de toutes les instructions comprises dans le chunk. Ainsi, mesurer les chunks permet de suivre l'exécution d'une charge de calcul dans son ensemble, alors que le suivi global reposant sur les compteurs matériels nécessite un traitement des données de type "fusion de données", car ces compteurs donnent seulement des informations spécifiques à des évènements indépendants.

Exécution parallèle d'un workload : Au cours de son exécution, une charge de calcul parallélisé (précédemment noté *Par*) sera dispersée parmi un certain nombre de threads d'exécution, dépendant de l'architecture ainsi que des choix utilisateur (e.g. choix de configuration). On définit $\mathscr{T} = \{th_1, th_2, ...th_n\}$ l'ensemble des threads $th_{k,(k\in 1..n)}$ utilisés pour exécuter *Par*. On note $q_{k,(k\in 1..n)} \in \mathscr{P}(Par)$ l'ensemble des chunks attribués pour exécution à un thread th_k, tel que : $q_1 \cup q_2 \cup ... \cup q_n = Par$

Performance et efficacité énergétique : Pour un workload parallèle *Par*, le temps d'exécution δ_{Par} est donné par la valeur maximale de $\delta_{th_{k,(k\in 1..n)}}$, où $\delta_{th_k} = \sum_{C_i \in q_k} \delta_{C_i}$ représente la durée d'exécution de tous les chunks contenus dans q_k attribué au thread th_k.

De la même façon, la consommation énergétique ε_{Par} d'une charge de travail parallèle est définie par la somme des $\varepsilon_{th_{k,(k\in 1..n)}}$, où $\varepsilon_{th_k} = \sum_{C_i \in q_k} \varepsilon_{C_i}$ représente l'énergie nécessaire à l'exécution de tous les chunks de q_k attribué au thread th_k.

La quantité de chunks exécutés par seconde peut être facilement calculée : ce sont nos *Chunks par Seconde* (CpS). De même, la quantité de chunks exécutés par Joule consommé peut être dérivée : ce sont nos *Chunks par Joules* (CpJ). Ces métriques nous permettent de décrire respectivement la performance et l'efficacité énergétique de charges de travail parallèles OpenMP. Ainsi, on définit les deux métriques suivantes :

$$Perf(Par) = \frac{1}{\delta_{Par}} * \sum_{q_k \in \mathscr{P}(Par)} |q_k| \tag{4.1}$$

et

$$EnergyEff(Par) = \frac{1}{\varepsilon_{Par}} * \sum_{q_k \in \mathscr{P}(Par)} |q_k| \tag{4.2}$$

La formule (4.1) définit notre métrique de performance basée sur les chunks (i.e. CpS), pour l'exécution d'une charge de travail parallèle, alors que la formule (4.2) décrit sont efficacité énergétique (i.e. CpJ).

Discussion : Nous formulons donc les chunks comme une métrique pertinente pour suivre en temps réel l'exécution de workloads parallèles. De plus, pour le suivi particulier de la performance et de l'efficacité énergétique, on définit les métriques CpS et CpJ basées sur les chunks. Il est important de rappeler que les caractéristiques des chunks varient en fonction de la boucle *for* parallélisée. Ainsi, les métriques hybrides - i.e. CpS et CpJ - sont des métriques spécifiques à la boucle parallélisée, et ne peuvent donc pas être utilisées comme moyens de comparaison entre différents workloads.

Dans ces travaux, nous nous intéressons uniquement aux workloads parallèles. Cependant, une charge de travail séquentielle peut être interprétée comme une charge de travail parallèle ne s'exécutant que sur un seul thread i.e. $\mathscr{T} =$ singleton. De ce fait, en suivant cette formulation, il est facile d'appliquer les formules (4.1) et (4.2) en remplaçant simplement les termes $|q_k|$ par 1. En effet, une région séquentielle est analogue à une région parallèle constituée d'un unique chunk.

4.1.2 Vers l'analyse de l'efficacité énergétique

La possibilité de suivre l'exécution d'une application et son efficacité peut être exploitée pour déterminer la configuration système la plus avantageuse, du point de vue de l'efficacité énergétique. Les codes d'application montrent souvent différentes phases d'exécution qui diffèrent dans leur comportement, au regard de leur utilisation des ressources matérielles, e.g. tâche riche en calcul (*compute-intensive*) versus tâche riche en accès mémoires (*memory-intensive*). Prenons l'exemple tendance d'une application s'exécutant sur le cloud pour illustrer notre propos. Comme très bien décrit par A.Bhattacharyya *et al.* [22], une application cloud traverse habituellement une multitude de types de tâches, i.e. workload, allant du stockage en mémoire avec des périodes de chargement, aux périodes de traitement gourmands en puissance calcul. Elle peut aussi avoir à réaliser des tâches de communication, et les problèmes de latence qui en découlent. Globalement, il y a autant de types de phases que de fonctionnalités présentes sur l'application. Pour rester dans cette idée, un autre exemple classique sont les applications pour smartphone. Cette famille d'applications joue le rôle d'interface entre l'utilisateur et le système d'exploitation. De ce fait, elles possèdent

différentes phases de fonctionnement, comme la gestion des appels ou l'accès à la galerie d'image, ou simplement rester en veille (*sleep mode*).

Ainsi, un changement de phase se traduit par un changement de type de workload. Plus particulièrement, dans notre cas d'applications parallélisées, ce changement signifie une modification des caractéristiques des chunks. En effet, la métrique des chunks est relative à la charge de travail contenue dans une itération de boucle. Ces changements causent des variations dans la consommation énergétique, d'où le besoin d'adapter la configuration du système en fonction de la phase courante de l'application, pour optimiser cette consommation énergétique.

4.1.2.1 Exemple d'application multi-phase

On montre le besoin d'adapter la configuration du système à la phase d'exécution courante en illustrant la différence entre les applications *memory-intensive* et *compute-intensive*. D'après la définition des chunks, les valeurs de CpS et CpJ s'interprètent de la manière suivante : plus hautes sont les valeurs, meilleures sont la performance et l'efficacité énergétique.

Exemple 2 On considère le code OpenMP détaillé figure 4.4. Ce programme est conçu pour avoir deux phases d'exécution, possédant différentes caractéristiques.

Ce programme consiste en une seule boucle *for* principale, exécutant consécutivement deux autres boucles *for*. Ces deux boucles intriquées sont les deux différentes phases (i.e. deux types différents de chunks) du programme principal. La première boucle réalise de simples additions, et effectue donc majoritairement des accès en mémoire. C'est une région de type *memory-intensive*. La seconde boucle quant à elle est une région *compute-intensive*, où fct() est une fonction nécessitant beaucoup de calculs.

Sur la figure 4.5 est tracée l'évolution temporelle des CpJ et CpS, collectés durant l'exécution de ce programme sur une configuration fixée. La consommation énergétique accompagne ces tracés sur un troisième graphique. Deux phases peuvent être observées sur les deux tracés des CpS et CpJ, avec des comportements similaires. De plus, malgré son caractère bruité, on peut considérer la consommation énergétique comme constante, puisque qu'aucun motif particulier ne ressort. À partir de ces tracés, on peut donc affirmer que les deux phases visibles sur les courbes correspondent aux deux boucles décrites précédemment. Ces résultats montrent que nos métriques de CpS et CpJ permettent de faire de l'analyse de phase. *Note :* Ce résultat contre-intuitif (i.e. on peut s'attendre à voir une variation de la consommation énergétique en fonction du type de workload) s'explique en partie par le fait que les mesures sont faites pour le système entier, et qu'elles ont été relevées pour une

```
00    #include <omp.h>
01  // Initialization
02    nthreads = 10;
03    double *A, *B, *C, *D;
04    posix_memalign((void**)&A, 64,
                           n*sizeof(double));
05    posix_memalign((void**)&B, 64,
                           n*sizeof(double));
06    posix_memalign((void**)&C, 64,
                           n*sizeof(double));
07    posix_memalign((void**)&D, 64,
                           n*sizeof(double));
08    for (i = 0; i < n1; ++i) {
09        A[i] = 0.1;
10        B[i] = 0.0;
11    }
12    for (i = 0; i < n2; ++i) {
13        C[i] = 0.1;
14        D[i] = 0.0;
15    }
16    omp_set_num_threads(nthreads);
17    for (j = 0; j < n; ++j) {
18      // Parallel region
19      #pragma omp parallel for
            schedule(dynamic, 1) // directive OpenMP
20      for (i=0; i<n1; i++){
21          B[i] = A[i] + B[i];
22      }
23      // Parallel region
24      #pragma omp parallel for
            schedule(dynamic, 1) // directive OpenMP
25      for (i=0; i<n2; i++){
26          D[i] = fct(C[i], D[i]);
27      }
28    return 0;
```

FIGURE 4.4 Un programme OpenMP simple en C possédant des phases alternantes de type compute-intensive et memory-bound.

configuration du système induisant une forte consommation énergétique - i.e. 19 cœurs à 2.1GHz - masquant ainsi les variations liées à l'application.

Lorsque ce code est exécuté sur différentes configurations d'architecture, on obtient pour l'efficacité énergétique les résultats visibles figure 4.6.

La figure 4.6 montre en (a) la performance (CpS) et en (b) l'efficacité énergétique (CpJ) de l'application dans sa globalité, mais aussi pour chacune des deux phases d'exécution, pour trois configurations particulières. La répartition (en %) du temps d'exécution entre les deux phases est décrite en (c). On définit tout d'abord la configuration optimale comme celle

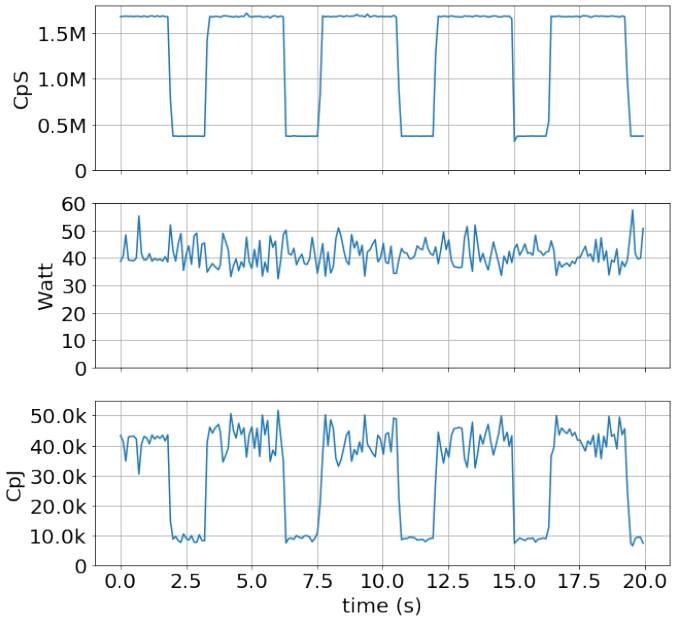
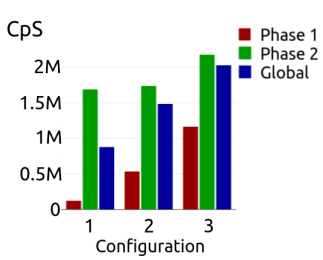
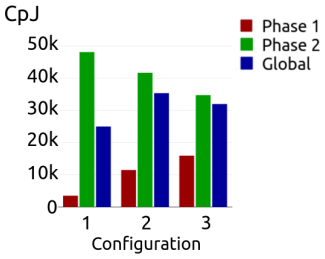


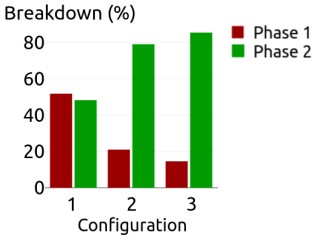
FIGURE 4.5 Profile de l'application synthétique exécutée sur un serveur Intel. De haut en bas : CpS, puissance consommée (Watt), CpJ.



(a) Moyenne du CpS globale et par phase



(b) Moyenne du CpJ globale et par phase



(c) Répartition du temps d'exécution des phases

FIGURE 4.6 Comparaison des métriques pour 3 configurations sur un serveur Intel. 1 : 2 cœurs et f= 1.5GHz ; 2 : 9 cœurs et f= 1.5GHz ; 3 : 17 cœurs et f= 2.1GHz.(a) : Décrit les valeurs moyennes du CpS, pour l'exécution globale et pour chaque phase d'exécution, (b) : Identique à (a) pour le CpJ, (c) : Répartition des durées des phases

donnant la meilleure efficacité énergétique. On peut voire que chaque phase possède une configuration optimale distincte. En effet, suivant la notation utilisée sur la figure 4.6, les configurations 1 et 3 sont optimales respectivement pour la phase 1 et la phase 2. De plus, la meilleure efficacité énergétique obtenue en considérant l'application dans sa globalité est pour la configuration 2, différentes des optimums des phases respectives. À noter que la possibilité d'alterner entre les configurations 1 et 3 mènerait évidemment à de meilleurs

résultats globaux, comparés à ceux de la configuration 2, cette dernière étant manifestement un compromis.

Dans cet exemple en particulier, une alternance entre les configurations 1 et 3 suivant les changements de phases d'exécution, plutôt que d'exécuter entièrement l'application sur la configuration 2, permettrait d'obtenir une augmentation du CpJ de près de 15% (en considérant un système parfait, sans coût lié aux changements de configuration).

À partir de l'exemple ci-dessus, on voit que les CpS et CpJ sont des métriques permettant de capturer aisément l'efficacité énergétique d'un système durant l'exécution d'une tâche, et en fonction de ses phases. En effet, on observe différents comportements des CpS et CpJ selon les différentes phases d'exécution. Cela traduit le fait que ces phases (i.e. types de chunk) ont leur propres caractéristiques de CpS et CpJ, nous permettant de conclure que ces métriques permettent de capturer les phases d'application.

Ces phases d'exécution ont été observées, dans le cadre de l'exemple précédent, a posteriori de l'exécution du programme. Un système en temps réel est donc nécessaire pour automatiser ce traitement i.e. déterminer le nombre de phases existantes dans l'application, les énumérer et finalement être capable d'identifier quelle phase est en cours d'exécution. Cela permettra par la suite d'identifier la configuration optimale d'une phase, et la sélectionner en temps réel.

4.1.3 Auto-encodeur pour la détection de phase d'exécution

Nous avons donc montré que nos métriques de CpS et CpJ permettent de voir en temps réel les phases d'exécution d'une application. Nous avons aussi observé que, du point de vue de l'efficacité énergétique, ces phases peuvent nécessiter différentes configurations. Ainsi, détecter les phases et identifier leur configuration optimale sont les clefs d'un contrôle dynamique optimal.

Ici, nous proposons une solution pour détecter automatiquement en temps réel les phases d'exécution d'applications. Notre solution exploite l'architecture des auto-encodeurs [77], une forme particulière de réseaux de neurones permettant entre autres de faire de l'extraction de caractéristiques. Ainsi, à l'aide d'un auto-encodeur entraîné, nous réalisons de la détection de phases avec d'excellents résultats.

Les auto-encodeurs sont donc des topologies particulières de réseaux de neurones profonds qui deviennent de plus en plus populaires. Ils sont utilisés dans différents domaines d'application, comme le traitement d'images avec la suppression de bruit [167]. L'objectif d'un auto-encodeur est de réduire la dimensionnalité de ses données d'entrée, e.g. la taille des images dans le cas du traitement d'images évoqué précédemment. Cette compression

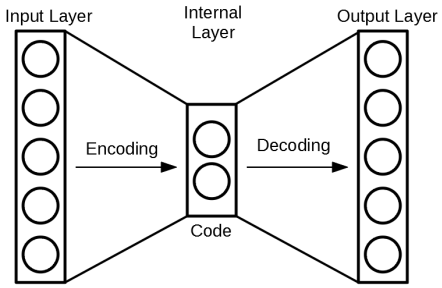


FIGURE 4.7 Illustration du concept d'auto-encodeur

dimensionnelle est réalisée par la forme en "goulot d'étranglement" de l'architecture du réseau de neurones, comme illustré sur la figure 4.7. En effet, les auto-encodeurs ont une forme symétrique, avec la couche interne de neurones comme axe de symétrie, de dimension inférieure à la couche d'entrée. Ainsi, deux parties distinctes peuvent être identifiées : l'encodeur et le décodeur. Le premier définit la partie allant de la couche d'entrée à la couche interne, tandis que le second désigne la partie allant de la couche interne à la couche de sortie du réseau de neurones. Plusieurs couches intermédiaires peuvent être implémentées à l'intérieur de ces deux parties, pour augmenter la profondeur du réseau.

Un auto-encodeur est entraîné pour reproduire en sortie ce qui lui a été donné en entrée. De cette façon, une représentation compacte de son entrée est disponible à la frontière entre l'encodeur et le décodeur. C'est la couche interne évoquée plus tôt, et décrite figure 4.7.

Dans notre cas, notre auto-encodeur est entraîné pour reproduire les valeurs de CpS et les données de configuration système (i.e. *Confiuration data* sur la figure 4.8 : nombre de cœurs alloués et fréquence de fonctionnement), en passant par la couche interne où sera récupérée l'information sur la phase. L'architecture implémentée pour répondre à cette tâche de détection de phase est illustrée figure 4.8, et décrite section 4.1.3.1.

4.1.3.1 Auto-encodeur proposé

Dans le but d'extraire une information sur la phase, une couche discrète (rectangle bleu, numéroté 1), i.e. sorties des neurones binaires, est utilisée comme couche interne. L'information sur la configuration du système (fréquence et nombre de cœurs de calcul alloués), représentée par le rectangle jaune numéroté 2, est donnée directement en entré du décodeur (rectangle vert numéroté 3) en la concaténant aux informations contenues dans la couche interne. De ce fait, l'auto-encodeur est contraint de se construire une représentation discrète des CpS, en se basant sur les informations de configuration. In fine, cette représentation correspondra à l'information sur la phase. Cela correspond donc à un entraînement non-supervisé d'un classificateur. En effet, le nombre de classes correspondant aux nombre de

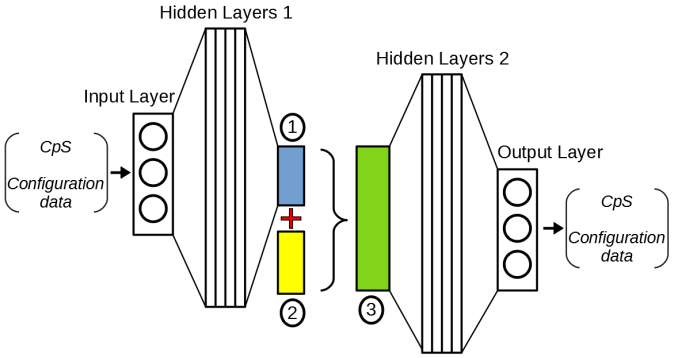


FIGURE 4.8 Auto-encodeur conçu. 1 : couche interne, 2 : données de configuration (#cœurs, fréquence), 3 : concaténation de 1 et 2

phases différentes n'est pas donné a priori à l'auto-encodeur. Chaque classe déterminée par l'entraînement de l'auto-encodeur correspond à une phase d'exécution identifiée par le réseau. L'entraînement nécessite par contre une première étape de collecte de données, pour balayer les différentes configuration systèmes et collecter les valeurs de CpS et CpJ correspondantes.

Après l'entraînement, le modèle a seulement besoin des données de CpS et de la configuration système pour déterminer la phase d'exécution courante du programme en cours. Ainsi, cela rend possible une détection de phase en temps réel.

4.1.3.2 Preuve de concept sur SRAD (benchmark Rodinia)

Nous avons donc un framework pour du suivi en temps réel de l'efficacité énergétique d'applications OpenMP, avec une méthode de détection de phase pour les applications ayant différentes phases d'exécution avec différentes configurations optimales. Nous proposons d'illustrer l'utilisation de ce framework avec le benchmark SRAD, issu de la suite de benchmark Rodinia [41]. Les tests sont effectués sur deux systèmes de calculs : un serveur Intel-Xeon à deux processeurs (*socket*) multi-cœurs (20 cœurs, 10 par socket), et une plateforme Odroid XU3 basée sur l'architecture Armv7 big.LITTLE.

Dans la suite, le terme de configuration système désigne un ensemble de deux caractéristiques : le nombre de cœurs de calcul attribués à l'exécution du programme, et la fréquence de fonctionnement de ces cœurs. Comme illustré sur la figure 4.9, le benchmark SRAD est un choix pertinent pour tester l'ensemble du framework. En effet, sur cette figure sont tracées les courbes de CpS et CpJ pour l'exécution de SRAD sur une configuration fixe, et on observe deux phases d'exécution distinctes.

Une caractérisation de cette application est lancée sur les deux systèmes de calculs, en explorant toutes les configurations possibles, i.e. toutes les fréquences et ensembles de cœurs

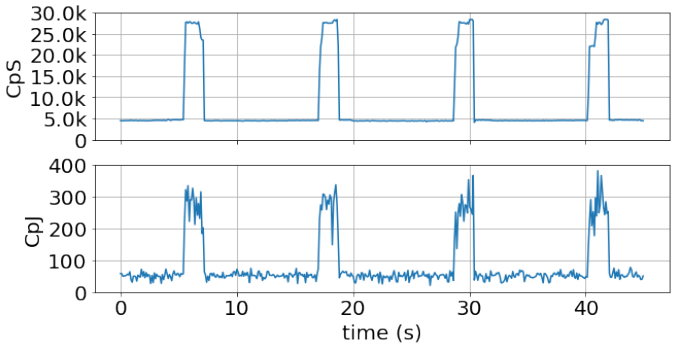
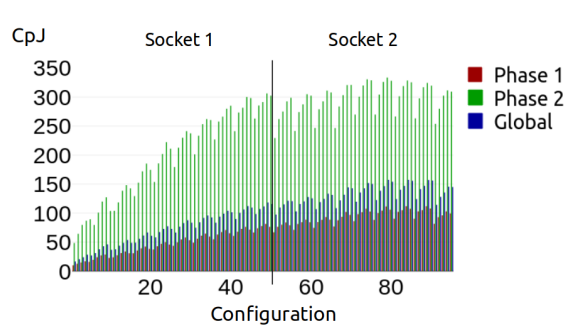


FIGURE 4.9 Échantillon de l'exécution de l'application SRAD. Profils selon le CpS et le CpJ.

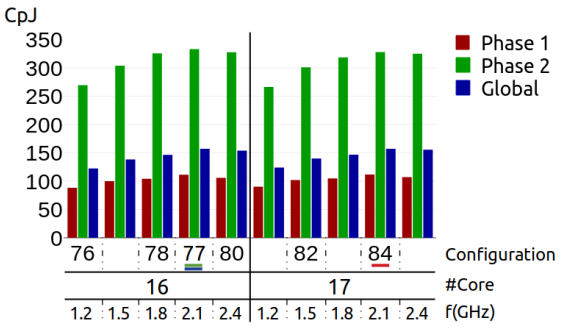
possibles. Les figures 4.10a et 4.10b montrent les résultats pour la caractérisation sur le serveur Intel. Le panel des configurations démarre à un seul cœur et augmente progressivement jusqu'à 19, car le cœur dédié à la collecte des données a été exclu. Pour chaque nombre de cœurs, un ensemble de fréquences est exploré, allant de 40 à 80% de la fréquence maximale, par pas de 10%. Les fréquences plus élevées ne sont pas utilisées pour notre analyse car sujettes à une régulation (*CPU throttling*) causée par "l'enveloppe thermique" (TDP) du processeur. Cela nous donne un total de 95 configurations différentes.

La même expérimentation est menée sur la carte Odroid, et décrite figures 4.10c et 4.10d, avec des résultats similaires. À noter que pour chacun des systèmes les configurations menant aux meilleures performances sont différentes et identifiées sur les figures 4.10b et 4.10d. Ce simple exemple démontre une nouvelle fois la pertinence des métriques proposées (CpS et CpJ) pour obtenir des informations sur des applications s'exécutant aussi bien sur des systèmes embarqués tels que la carte Odroid, que sur des systèmes de calculs haute-performance (HPC) tels que le serveur Intel.

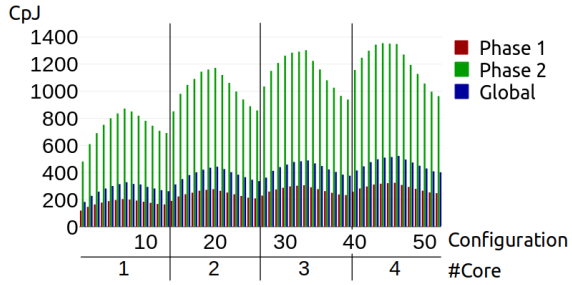
Sur la figure 4.11 sont tracés l'évolution des CpS avec les deux phases différentes détectées en temps réel par l'auto-encodeur. Cela illustre les très bons résultats dans l'ensemble de l'auto-encodeur, après une durée d'entraînement relativement courte. En effet, l'entraînement de l'auto-encodeur prend entre 1min et 5min pour chaque jeu de données (serveur Intel et carte Odroid), et converge vers son erreur finale (*loss*) après quelques dizaines de secondes. Ces entraînements ont tous été menés sur le serveur Intel Xeon, disposant de CPUs Xeon E3-1225v3. Comme attendu, une fois entraîné, notre encodeur produit un code correspondant à chaque phase. La valeur du code est arbitraire et peut varier d'un entraînement à l'autre, et doit donc être interprétée comme un type énuméré.



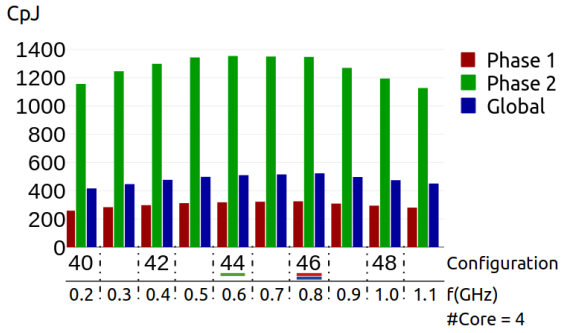
(a) CpJ pour le serveur Intel, 95 configurations



(b) CpJ pour le serveur Intel, zoom sur les configurations optimales (soulignées des couleurs des phases correspondantes)



(c) CpJ pour la carte Odroid, 52 configurations



(d) CpJ pour la carte Odroid, zoom sur les configurations optimales (soulignées des couleurs des phases correspondantes)

FIGURE 4.10 Caractérisation de l'application SRAD, sur deux architectures : un serveur Intel possédant 20 cœurs et une plate-forme Arm avec 4 cœurs hétérogènes.

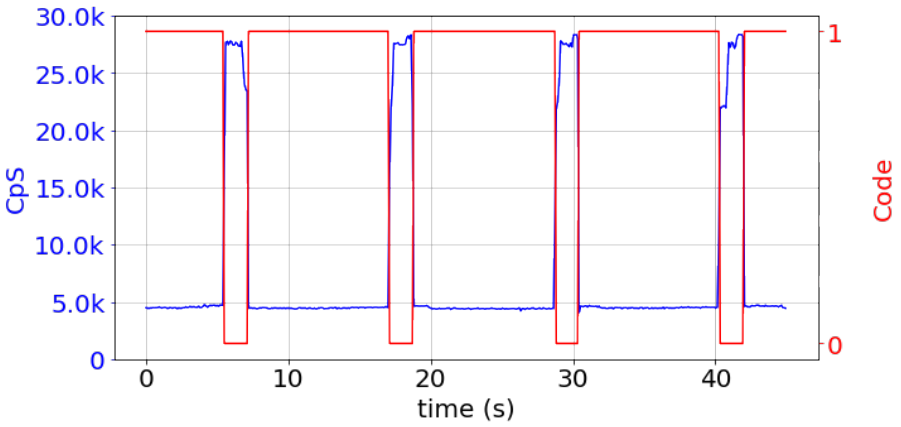


FIGURE 4.11 Exemple de détection de phase pour l'application SRAD, sur le profil du CpS.

4.2 Optimisation des CpJ : méthodologie et solution

Nous avons donc à notre disposition une "boîte à outils" composée de métriques pour suivre en temps réel les performances et l'efficacité énergétique (respectivement CpS et CpJ) d'une application OpenMP parallélisée, ainsi que d'un outil de détection automatique de phases via l'exploitation des auto-encodeurs. Afin de garantir les hypothèses d'utilisation des CpJ définies section 4.1.1, les applications considérées seront composées essentiellement de boucles *for* parallélisées de sorte que les chunks représentent la majorité du calcul utile du système de calcul. Ainsi, pour la mesure du CpJ, nous pourrons raisonnablement supposer que la consommation énergétique du système est impactée majoritairement par les chunks. Ensuite, les chunks étant spécifiques au workload parallélisé, nous considérerons le contrôle d'une seule application à la fois.

Dans cette partie, nous proposons une solution d'optimisation de l'efficacité énergétique, via l'optimisation des CpJ. Cette solution est basée sur l'apprentissage par renforcement, qui a l'avantage de ne pas requérir de données collectées en amont de son entraînement.

Note : Pour la suite de cette partie, seul le serveur Intel Xeon sera utilisé pour les expérimentations. En effet, aux vue de l'état de l'art récent ciblant majoritairement les systèmes hétérogènes (e.g. carte Odroid bib.LITTLE), il semblait plus intéressant de se focaliser sur un système de type serveur SMP (*Symmetric Multiprocessing*), à l'architecture homogène.

4.2.1 Méthodologie

Tout d'abord, dans cette section est présentée la méthode suivie pour créer, tester et valider notre solution de contrôle dynamique pour l'optimisation des CpJ.

Étape 1 : Création d'un benchmark synthétique Nous avons montré précédemment que les chunks sont une métrique relative au type de travail effectué, et que la configuration optimale (i.e. menant la meilleure efficacité énergétique) varie selon le type de chunks. Nous proposons donc dans un premier temps d'illustrer plus en détails cet impact du type de chunks sur la configuration optimale en s'appuyant sur la dualité "compute-intensive vs. memory-intensive" évoquée en 4.1.1. Pour cela nous proposons un benchmark synthétique, développé section 4.2.2, composé d'applications OpenMP statiques (i.e. un seul type de chunks et une phase d'exécution). Ce benchmark nous servira par la suite à démontrer l'efficacité de notre système de contrôle dynamique, détaillé en 4.2.3.

Étape 2 : Implémentation d'un apprentissage profond par renforcement adapté Pour assurer les prises de décision en temps réel (i.e. le cœur du système de contrôle), nous proposons d'implémenter une IA basée sur l'apprentissage profond par renforcement. Comme évoquée en section 2.4.2, l'apprentissage par renforcement est privilégiée pour les solutions de prises de décisions, car il garantit en théorie de converger vers les meilleures solutions possibles. De plus, le fait ici d'utiliser une méthode d'apprentissage profond (i.e. réseau de neurones) permet d'avoir une solution prenant en compte un champ de possibilité très grand. Nous développons ce point section 4.2.3.

Étape 3 : Évaluation du système de contrôle Enfin, nous évaluerons notre système sur différentes applications. Dans un premier temps, nous comparerons les performances de notre système pour le benchmark DGEMM [99] avec les différents gouverneurs Linux. Ensuite, nous nous intéresserons aux applications multi-phases avec une application synthétique composite de deux applications statiques, et le benchmark SRAD. Les résultats font l'objet d'une nouvelle partie, en 4.3.

Note : de futurs travaux devront valider notre système sur un plus large ensemble d'applications multi-phases, notamment en se basant sur des suites de benchmarks connues comme Rodinia et PARSEC.

4.2.2 Création d'un benchmark synthétique

4.2.2.1 Benchmark synthétique : Modèle

Pour extraire les gains potentiels en efficacité énergétique du système de calcul choisi (i.e. serveur Intel), nous avons conçu un modèle paramétrable de benchmark synthétique à partir duquel on peut dériver des benchmarks aux profils différents. Le modèle est construit autour de deux blocs de code consécutifs et paramétrés, permettant de couvrir les opérations de type *memory-intensive* et de type *compute-intensive*. De ce fait, les applications ainsi créées possèdent différents comportements, selon leur intensité en termes d'accès mémoire et de besoin en calculs. Le modèle est décrit sur la figure 4.12.

Le segment de code de type *memory-intensive* exécute un ensemble d'opérations sur de grands vecteurs telles que des additions, copies et permutations. L'intensité du recrutement mémoire dépend donc de la taille des vecteurs, de leurs dimensions, et du comportement aléatoire des différents accès aux vecteurs (i.e. augmente la probabilité des *cache-misses*). Le segment *compute-intensive* quant à lui exécute un ensemble de calculs mathématiques relevant de l'algèbre linéaire et de combinaisons de fonction arithmétiques et trigonométriques, faisant appel à des méthodes calculatoires intenses telles que l'analyse en série de Fourier

```
00  #include <omp.h>
01  // Initialization
02  Some environment definitions
03  // Parallelization
04  omp_set_num_threads(nthreads);
05  // OpenMP directive
06  #pragma omp parallel for schedule(dynamic, 1)
07  for (i=0; i<n; i++){
08      for (j=0; j<100; j++){
09          if (j < coef){
10              MEM-intensive code fragment
11          }
12          else{
13              CPU-intensive code fragment
14          }
15  }
16  return 0;
```

FIGURE 4.12 Modèle du Benchmark.

et différentes opérations en virgule flottante. La charge de stress appliquée au(x) CPU(s) dépend alors du nombre d'appels à ces fonctions, et de leurs caractéristiques. De ce fait, ces deux blocs sont des codes basiques représentatifs des deux caractéristiques que l'on souhaite mettre en relief. Enfin, pour dériver un benchmark, le ratio entre l'intensité CPU et l'intensité Mémoire est fixé par la variable "coef".

4.2.2.2 Caractérisation des applications synthétiques

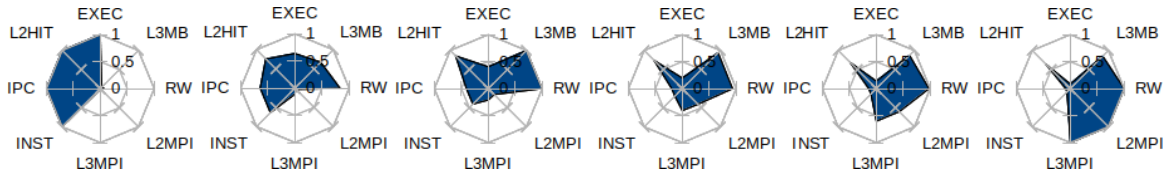
À partir du modèle décrit plus haut, 6 benchmarks sont produits : *C100M0*, *C98M2*, *C96M4*, *C90M10*, *C80M20*, *C0M100*. La notation *CxMy* traduit les proportions entre les deux modes de contraintes que l'on souhaite cibler : *x* est le pourcentage d'intensité CPU et *y* celui de l'intensité Mémoire. Ainsi, les benchmarks *C0M100* et *C100M0* sont respectivement exclusivement *memory-intensive* et *compute-intensive*. On attribue jusqu'à 19 cœurs de calcul aux threads d'exécution, parmis les 20 disponibles sur notre serveur Intel (c.f. section 4.1.3.2).

Toutes les valeurs décrites dans la suite de cette partie sont collectées à partir des compteurs matériels de performance Intel, via l'outil Intel PCM [1], et décrites à travers les graphiques radars présentés sur la figure 4.13. Le tableau 4.1 liste toutes les métriques considérées (i.e. compteurs de performance), triées en fonction de leur nature, selon si elles relèvent de comportements calculatoires, ou d'utilisations mémoires. La figure 4.13 montre que chaque application possède son propre profil, et stimule différemment le CPU et la mémoire, avec différents impacts sur les compteurs. En effet, le benchmark "CPU-intensive"

CPU intensity oriented	Memory intensity oriented
IPC = instructions per CPU cycle	RW = MEM Read and Write
EXEC = instructions per nominal CPU cycle	L3MB = L3 cache external memory bandwidth
L3HIT = L3 (read) cache hit ratio	L2MPI = number of L2 (read) cache misses per instruction
INST = Instructions retired	L3MPI = number of L3 (read) cache misses per instruction

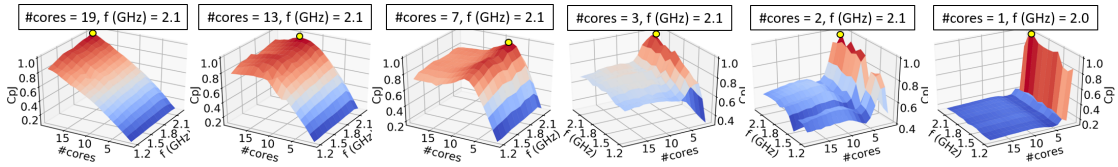
TABLE 4.1 Description des compteurs Intel PCM.

C100M0 possède tous les compteurs concernés à leur valeur maximale. À l'inverse, le benchmark "memory-intensive" *C0M100* possède tous les compteurs orientés mémoire à leur maximum.



(a) C100M0 (b) C98M2 (c) C96M4 (d) C90M10 (e) C80M20 (f) C0M100

FIGURE 4.13 Profil des applications selon les compteurs PCM.



(a) C100M0 (b) C98M2 (c) C96M4 (d) C90M10 (e) C80M20 (f) C0M100

FIGURE 4.14 Caractérisation de l'efficacité énergétique des applications (i.e. CpJ) et configuations optimales.

4.2.2.3 Efficacité énergétique

Pour chacun des 6 benchmarks proposés, nous réalisons une caractérisation exhaustive, i.e. nous exécutons l'application pour chaque configuration possible (fréquence et nombre de cœurs) et reportons le CpJ moyen sur l'ensemble de la durée d'exécution. *Note :* La mesure de la consommation énergétique utilisée pour calculer les CpJ est directement extraite des registres spécifiques au modèle (MSR) RAPL d'Intel [74].

Les résultats sont présentés figure 4.14, où la meilleure configuration est étiquetée. On remarque la diversité des configurations optimales parmi les applications, qui souligne la

Benchmark		C100M0	C98M2	C96M4	C90M10	C80M20	C0M100
Best Conf.	CpJ	3866	2505	1581	818	517	336
i.e. Reference	CpS	303k	170k	90k	40k	25k	12k
vs.	CpJ	10%	25%	29%	95%	60%	442%
Performance	CpS	-12%	-11%	-19%	21%	1%	151%
vs.	CpJ	16%	24%	33%	44%	75%	469%
Powersave	CpS	75%	59%	49%	56%	92%	355%
vs.	CpJ	10%	20%	32%	18%	75%	433%
Ondemand	CpS	-12%	-14%	-17%	-1%	50%	193%
vs.	CpJ	10%	20%	29%	32%	56%	469%
Conservative	CpS	-12%	-14%	-19%	-16%	1%	160%

TABLE 4.2 Gains en efficacité énergétique (CpJ) et performance (CpS) des configurations optimales des benchmarks, en comparaison avec les gouverneurs Linux : Powersave, Performance, Ondemand et Conservative.

dualité entre l'intensité des contraintes CPU et l'intensité des contraintes Mémoire au sein des applications. En effet, les deux applications extrêmes (*C100M0* et *C0M100*) possèdent deux configurations optimales (en terme de CpJ) opposées, au regard du nombre de ressources de calcul attribuées, i.e. 1 cœur pour l'application *memory-intensive* contre 19 cœurs pour l'application *CPU-intensive* (nombre maximum de cœur attribué, avec 1 cœur par thread).

Pour observer l'impact que peut avoir l'attribution de la meilleure configuration d'une application, durant son exécution sur le système de calcul concerné, on compare les valeurs de CpJ et CpS moyennes pour des exécutions soumises aux gouverneurs Linux suivant : Powersave, Performance, Ondemand et Conservative. Les résultats sont reportés dans le tableau 4.2.

Nous obtenons des gains potentiels en efficacité énergétique allant de 10% à 469%, en comparaison avec les gouverneurs Linux. Des pertes de performance sont observées sur la majorité des applications de type *CPU-intensive*, mais sont toujours inférieures à 20%, ce qui reste raisonnable comparé aux considérables améliorations en efficacité énergétique. Le principal problème avec les gouverneurs Linux est l'absence de gestion des ressources, en termes de contrôle *thread-to-core*. Cela met en valeur les avantages de pouvoir contrôler l'attribution des ressources de calcul aux applications parallélisées. D'où l'utilisation d'apprentissage "en-ligne" décrite plus tard pour adapter de façon dynamique la configuration du système.

4.2.3 Apprentissage par renforcement pour de la reconfiguration dynamique

Comme déjà évoqué, la solution proposée pour réaliser le contrôle dynamique de la configuration du système pour maximiser son efficacité énergétique se base sur l'apprentissage par renforcement. Plus précisément, nous nous inspirons du Deep Q-learning (DQL) [112], une technique d'apprentissage profond développée par DeepMind en 2015.

Ainsi, le système de contrôle que nous proposons réalise à la fois l'entraînement de son réseau de neurones et le contrôle des configurations en temps réel, durant l'exécution de l'application contrôlée.

Ce système est décrit figure 4.15.

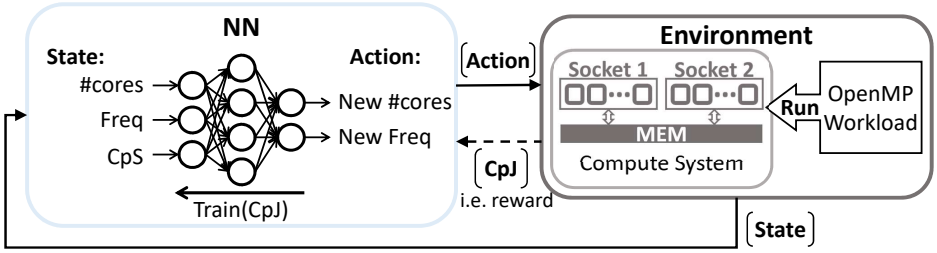


FIGURE 4.15 Système de contrôle.

Il est basé sur le principe de récompense utilisé dans l'apprentissage par renforcement (RL pour *Reinforcement Learning*). Ici, le réseau est seulement entraîné pour réaliser de l'inférence combinatoire, ce qui signifie que ses prises de décision sur les actions à réaliser dépendent uniquement de l'état courant du système, i.e. ce n'est pas fonction des états précédents comme originalement dans le DQL. Notre modèle a tout de même été développé de façon générique pour supporter le DQL d'origine, mais ce n'est pas l'objet de cette contribution. L'environnement représente le système de calcul multi-cœurs exécutant les applications OpenMP. Pour chaque état possible, la qualité de chaque action (i.e. configuration système) est évaluée via la fonction de récompense (i.e. reward) qui est directement fonction du CpJ. Ainsi, notre approche pour obtenir un contrôle efficace repose sur une définition compacte de l'état de l'environnement, permettant d'assurer une certaine rapidité d'exploration et de convergence. De ce fait, l'état est défini à partir de la configuration du système de calcul (i.e. fréquence courante et nombre de cœurs actifs) et de la valeur du CpS.

4.2.3.1 Processus de prise de décision

La prise de décision "en-ligne" repose sur l'apprentissage par l'expérience i.e. *learning-by-doing*. En suivant la terminologie du RL, les décisions sont appelées *actions* et l'envi-

ronnement est défini par son *état*. Le système commence à prendre des décisions à la fin de son apprentissage, appelé *phase d'exploration*. Durant cette phase d'exploration, des actions aléatoires sont prises afin de déterminer la meilleure action pour chaque état. Les actions sont récompensées en fonction des bénéfices qu'elles apportent (ici, en fonction de la valeur du CpJ). L'avantage d'utiliser ici un réseau de neurones (NN pour *Neural Network*) repose sur ses capacités d'interpolation i.e. la capacité de prédire une action en fonction d'un nouvel état inconnu, en se basant sur le savoir accumulé durant la phase d'exploration.

Dans notre contexte, l'état est défini par la configuration du système couplée avec la valeur du CpS, i.e. l'ensemble des états n'est pas un ensemble discret. Une action est une nouvelle configuration à appliquer au système, et la récompense est la valeur du CpJ résultante de ce changement. L'API Tensorflow [2] est utilisée pour construire le réseau de neurones, via l'interface Keras [44].

Globalement, ce système est efficace pour le contrôle d'applications statiques (i.e. à une seule phase d'exécution). Nous illustrons ses performances dans la section 4.3.1 sur le benchmark DGEMM [99]. Cependant, ce système a plus de difficultés pour contrôler l'exécution d'applications possédant différentes phases d'entraînement. En effet, même en utilisant toutes les ressources du DQL (avec la prise en compte des états passés), le système n'apprend pas correctement à identifier les phases de fonctionnement. Ainsi, nous proposons d'inclure dans notre boucle de contrôle l'outil de détection de phases présenté plus haut.

4.2.3.2 Inclusion de l'auto-encodeur pour les applications multi-phases

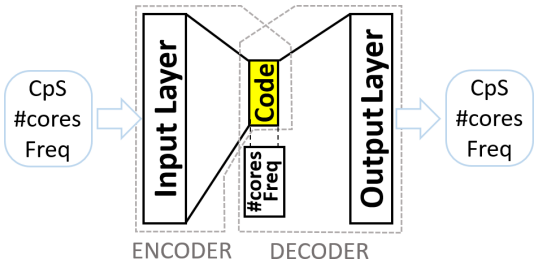


FIGURE 4.16 Auto-encodeur proposé.

Nous proposons donc d'inclure au système de contrôle l'auto-encodeur présenté section 4.1.3.1 en figure 4.8 et résumé ici figure 4.16. Il est entraîné hors-ligne à reproduire l'état de l'environnement (i.e. valeur de CpS et configuration du système). Ainsi, on peut extraire de sa couche interne des informations relatives à la phase de l'application en cours d'exécution, désignées par le terme *code* sur la figure 4.16.

Ainsi, le système de contrôle final est décrit figure 4.17. L'auto-encodeur permet de récupérer directement l'information de la phase. De ce fait, la valeur du CpS devient obsolète

pour l'apprentissage, et est directement remplacée par l'information sur la phase. Nous illustrons le bon fonctionnement du système en section 4.3.2, où une application synthétique à deux phases nous sert de preuve de concept.

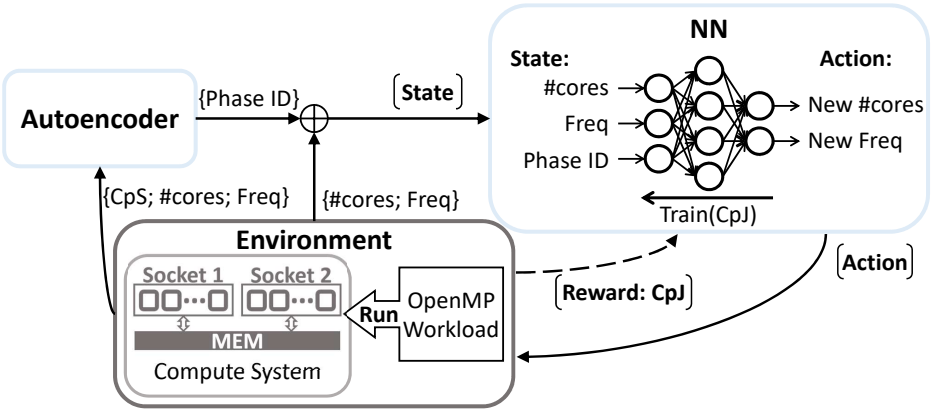


FIGURE 4.17 Système de contrôle avec l'auto-encodeur.

4.2.3.3 Détails d'implémentation des réseaux de neurones

Tous nos réseaux de neurones sont implémentés en Python. Nous utilisons plus précisément l'API Tensorflow [2], avec Keras [44] comme frontend. Les apprentissages utilisent l'optimiseur *Adam* de Keras, paramétré par défaut. Les dimensionnements des réseaux n'ont pas fait l'objet d'une optimisation particulière. Ils sont basés sur les tendances relevées dans l'état de l'art, et nos résultats expérimentaux.

Agent ("NN" figure 4.15) : Notre agent est décrit dans le tableau 4.3. Il est constitué de 3 couches cachées (i.e. L1, L2 et L3) de type *Dense*. La dimension correspond au nombre de neurones. La couche d'entrée est de dimension 3, i.e. la dimension de l'état de l'environnement, et la dimension de la couche de sortie est de 209, i.e. le nombre d'actions possibles. La période d'exploration prend 2048 itérations. Ce nombre d'itérations a été déterminé expérimentalement pour garantir un apprentissage correct de notre agent tout en minimisant la quantité de données collectées (i.e. temps d'exploration réduit). L'apprentissage

Couches :	Entrée	L1	L2	L3	Sortie
Type	Input	Dense			
Dimensions	3	8	64	256	209
Fonction d'activation	–	*Linear*			

TABLE 4.3 Dimensionnement du réseau de neurones de l'Agent.

prend moins de 100ms par itération. Afin de permettre un l'apprentissage en temps réel, nous choisissons une période d'échantillonnage du contrôleur de 500ms par itération. Enfin, une inférence est réalisée en moins de 2ms.

Auto-encodeur : L'auto-encodeur est décrit dans le tableau 4.4. Il est composé de 7 couches cachées successives, réparties comme suit : 3 pour la partie encodage (i.e. L1, L2, L3), 3 pour la partie décodage (i.e. L4, L5, L6), et une pour le code interne. Le code interne est une couche dense binaire de dimension 2, de sorte que le code de phase peut prendre 4 valeurs. La binarisation est assurée par l'utilisation de la fonction d'activation *binary tanh*. L'apprentissage se fait en 50 époques, et prend moins de 5s par époque. Par conséquent, il est réalisé hors-ligne. Enfin, il faut moins de 5ms pour prédire l'ID de la phase, ce qui permet de réaliser l'inférence durant l'exécution.

Modules :	Encodeur				|	Décodeur			
Couches :	Entrée	L1	L2	L3	Interne	L4	L5	L6	Sortie
Type	Input	Dense							
Dimensions	3	100	100	100	2	100	100	100	3
Fonction d'activation	–	*Linear*			*binary tanh*	*Linear*			

TABLE 4.4 Dimensionnement de l'auto-encodeur.

4.3 Résultats et analyses

Dans cette dernière partie sont exposées les différentes expérimentations et leurs résultats. Elles sont menées sur le serveur Intel Xeon présenté précédemment. Ici, le driver utilisé est le CPUFreq par défaut sur le noyau Linux, à la place du driver d'Intel, le *Intel P-State*. Il permet entre autres d'accéder aux gouverneurs Linux tel que *ondemand* [127].

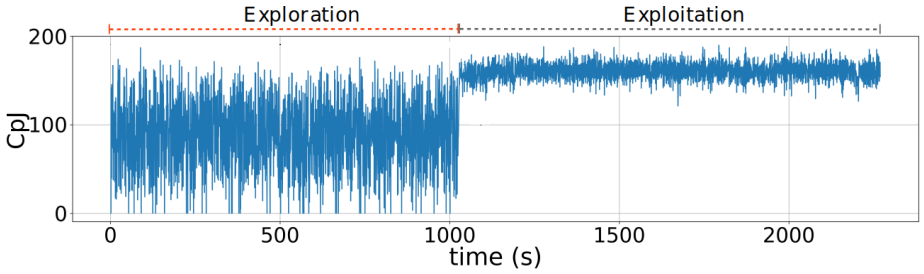
Les configurations possibles vont donc de 1 à 19 cœurs, répartis équitablement sur les deux sockets, pour une fréquence de fonctionnement allant de 1.2GHz à 2.2GHz, par pas de 100MHz. Nous n'explorons pas les fréquences supérieures à 2.2GHz, car ces dernières sont soumises à "l'étranglement thermique" (TDP) et il n'est donc pas possible de garantir leur constance.

4.3.1 Applications OMP statique

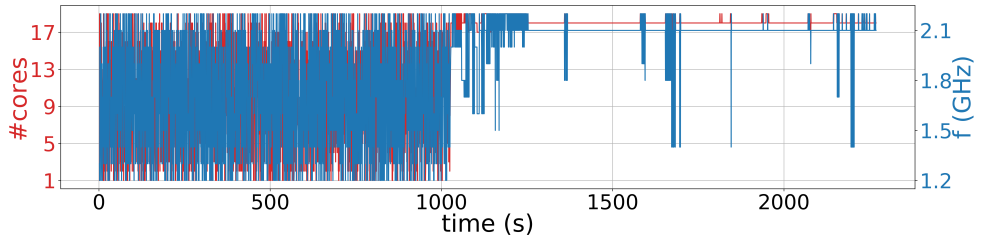
Dans cette première partie des résultats, nous nous intéressons au contrôle d'applications statiques, c'est-à-dire ne possédant qu'une seule phase d'exécution équivalant à un seul type

de chunks. Nous utilisons donc la version du système décrite figure 4.15, ne possédant pas l'auto-encodeur, détecteur de phases.

Ce design est évalué avec le benchmark DGEMM [99]. Une recherche exhaustive ad-hoc a permis de déterminer que la configuration optimale de cette application est de 18 cœurs à 2.1GHz, pour un CpJ moyen de 166. La période d'échantillonnage est fixée à 500ms, pour la collecte des données ainsi que la prise de décision.



(a) Évolution de l'efficacité énergétique.



(b) Actions du contrôleur i.e. configuration.

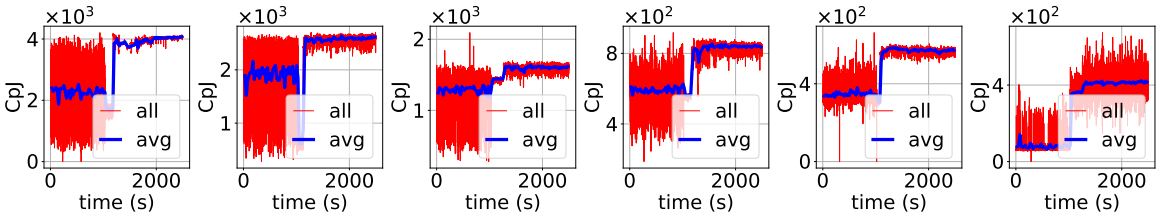
FIGURE 4.18 Évolution des variables du systèmes durant l'apprentissage en-ligne, pour le contrôle du benchmark DGEMM.

Pour notre expérimentation, la phase d'exploration est fixée arbitrairement à 2048 itérations (i.e. 1024s) ce qui est relativement court en comparaison aux quantités habituelles de données utilisées en général pour entraîner un réseau de neurones. De plus, aucune technique d'augmentation du jeu de données n'a été utilisée pour améliorer la qualité du dataset. On observe néanmoins sur la figure 4.18 que cela suffit à notre système pour atteindre des performances quasi-optimales. Figure 4.18a montre le tracé des CpJ durant l'exécution. On note la valeur globalement stable du CpJ, restant dans les 3% de la valeur connue pour la configuration optimale. On remarque quelques fluctuations, dont des modifications sporadiques transitoires de la configuration sur la figure 4.18b. Cependant, la configuration moyenne imposée par notre système de contrôle après entraînement est de 18.05 cœurs à 2.09GHz, que l'on peut raisonnablement considérée comme égale à la configuration optimale pré-déterminée.

Gains : En comparant avec les gouverneurs Linux disponibles sur notre serveur Intel Xeon, notre méthode produit les gains suivants : 10%, 17%, 11% et 10%, respectivement

comparés aux gouverneurs *Performance, Powersave, Ondemand, Conservative*. Ces résultats sont similaires à ceux obtenus pour le benchmark *C100M0*, ce qui est cohérent avec le caractère *compute-intensive* de DGEMM.

Validations complémentaires : En complément, nous proposons de valider ce contrôleur sur l'ensemble des applications du benchmark synthétique.



(a) C100M0 (b) C98M2 (c) C96M4 (d) C90M10 (e) C80M20 (f) C0M100

FIGURE 4.19 Efficacité énergétique pour chacune des applications du benchmark synthétique, durant l'utilisation du contrôleur.

Figure 4.19 montre l'évolution de l'efficacité énergétique pour chacune des six expérimentations (une par application). La remarque globale que nous pouvons faire est que, pour chaque expérience, l'efficacité énergétique en fin d'entraînement (après les 1024s d'exploration pure) est maximisée i.e. supérieure ou égale au maximum vue durant l'exploration.

Le tableau 4.5 résume ces résultats, en comparant les configurations déterminées par le contrôleur avec les configurations optimales mesurées en amont des simulations. Les configurations moyennes finales du système correspondent globalement aux configurations optimales des applications respectives. Des gains sont même obtenus lorsque nous comparons les valeurs de CpJ en fin d'entraînement avec celles obtenues pour les configurations optimales.

Benchmark		C100M0	C98M2	C96M4	C90M10	C80M20	C0M100
Best Conf.	CpJ	3866	2505	1581	818	517	336
	#cores, freq(GHz)	19, 2.1	13, 2.1	7, 2.1	3, 2.1	2, 2.1	1, 1.9
vs. Results	CpJ	4051	2580	1603	839	575	405
	Gains (CpJ)	4.8%	3.0%	1.4%	2.6%	11.2%	20.5%
	#cores, freq(GHz)	19, 2.1	13, 1.9	8, 1.9	4, 2.1	3, 2.1	1, 1.9

TABLE 4.5 Résultats du contrôleur pour chacune des applications du benchmark synthétique.

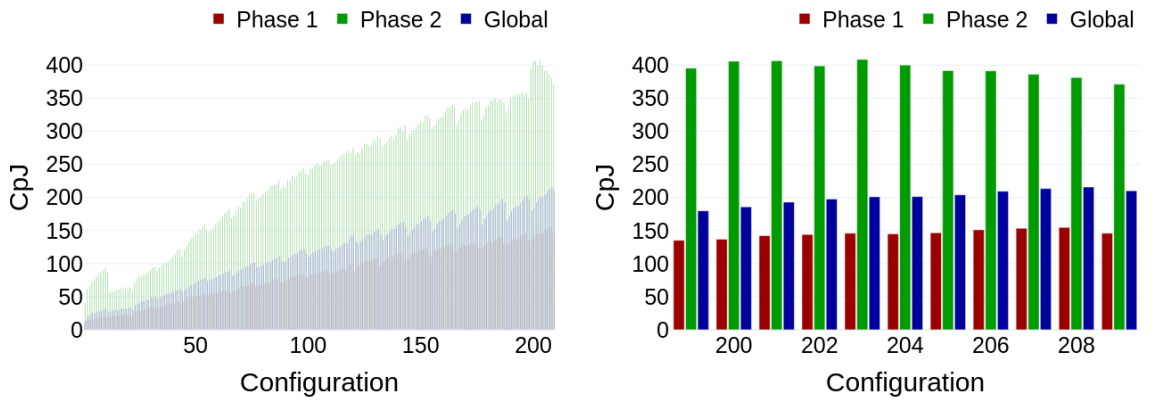
Il faut cependant être vigilant concernant ces gains. En effet, nous obtenons ces résultats très positifs allant jusqu'à 20% en comparaison avec les configurations optimales. Or, par définition, les configurations optimales sont celles délivrant les meilleures valeurs d'efficacité

énergétique. Ce résultat contre-intuitif demande des investigations supplémentaires pour expliquer son origine. Il est probable que des variations expérimentales entre la caractérisation des applications et les tests de notre contrôleur soient venues altérer les performances du serveur. Néanmoins, l'impact positif du contrôleur ne fait aucun doute sur la figure 4.19 et permet de définitivement confirmer son bon fonctionnement pour les applications statiques.

4.3.2 Applications OMP variables

Nous nous intéressons maintenant au contrôle d'applications variables, c'est-à-dire possédant plusieurs phases d'exécution. Nous utilisons donc la version du système décrite figure 4.15, possédant la détection de phases. Le reste du set-up expérimental est le même que pour la section 4.3.1.

4.3.2.1 Évaluation sur SRAD

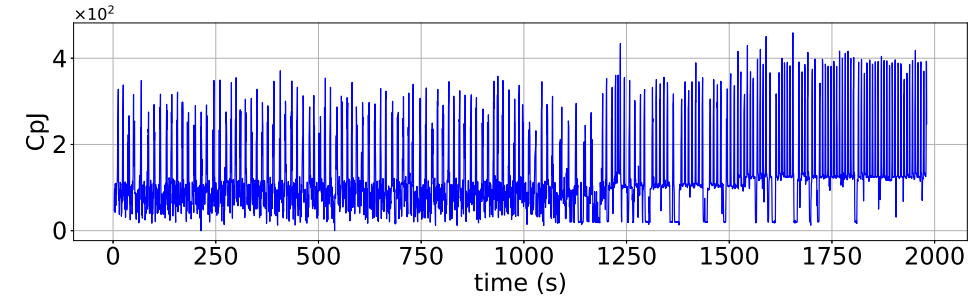


(a) CpJ pour les 209 configurations du serveur

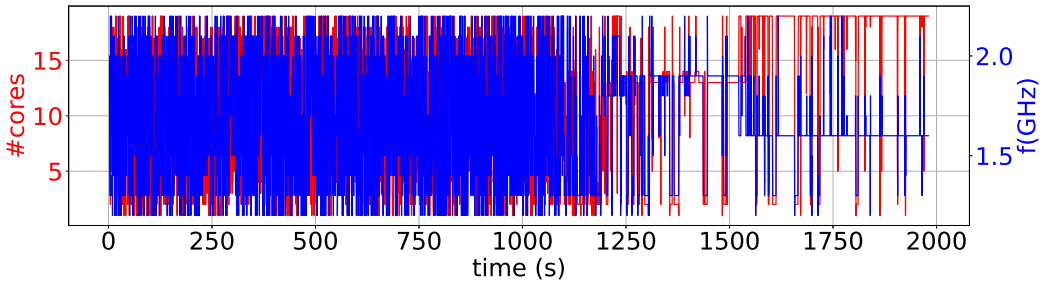
(b) Zoom sur les configurations optimales : 208 et 203 respectivement pour la phase 1 et la phase 2.

FIGURE 4.20 Caractérisation de l'application SRAD, sur le serveur Intel, en répartissant les ressources équitablement parmi les sockets.

Nous évaluons notre système de contrôle sur une application multi-phase connue : SRAD, issue de la suite de benchmark Rodinia [41]. Cette application possèdent deux phases, et nous proposons une nouvelle caractérisation de cette application (i.e. différente de celle proposée figure 4.10), prenant en compte la répartition homogène des ressources attribuées parmi les deux sockets du serveur Intel. Les résultats de la caractérisation sont décrits sur la figure 4.20. Les configurations optimales sont {19 cœurs, f=1.6GHz} et {19 cœurs, f=2.1GHz}, respectivement pour la phase haute et la phase basse.



(a) Évolution de l'efficacité énergétique.



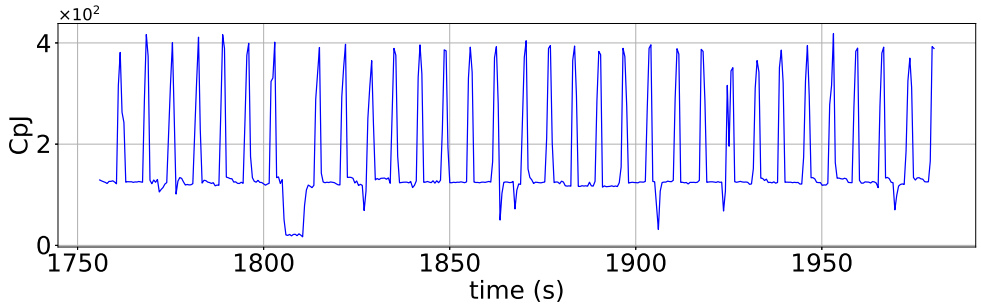
(b) Actions du contrôleur i.e. configuration.

FIGURE 4.21 Traces de fonctionnement du contrôleur, pour SRAD.

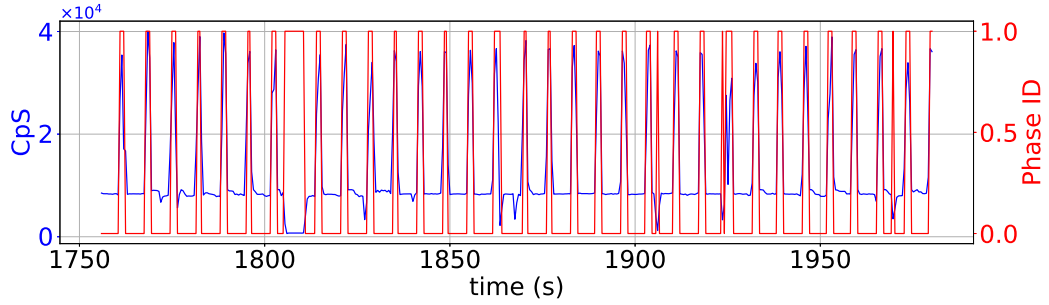
Les résultats obtenus avec notre système de contrôle sont exposés sur les figures 4.21 et 4.22. Tout d'abord, une vue d'ensemble de l'entraînement est proposée figure 4.21, où sont clairement visibles les périodes d'exploration et d'exploitation de l'apprentissage, respectivement de 0 à 1024s pour l'exploration pure, et de 1024s à environ 2000s pour l'exploitation. On peut également observer la transition entre ces deux périodes, entre 1024s et 1700s environ, durant laquelle le taux d'actions aléatoires décroît progressivement de 100% à 5% au profit des actions décidées par l'agent. Les 5% restant d'actions aléatoires permettent de maintenir un certain niveau d'apprentissage afin d'ajuster les comportements appris durant la courte période d'exploration. Ce pourcentage peut sur le long terme être mis à 0.

La première remarque que nous pouvons faire est que le système converge vers une configuration particulière : {19 cœurs, f=1.6GHz}, la configuration optimale de la phase haute. Bien que ce résultat montre une certaine qualité d'apprentissage, on note l'absence de changement de configuration en fonction de la phase d'exécution de l'application. Pourtant, on peut noter sur la figure 4.22b que l'information sur la phase est correctement extraite par l'auto-encodeur. Plusieurs raisons peuvent expliquer ce défaut de fonctionnement, et nécessitent de futures investigations :

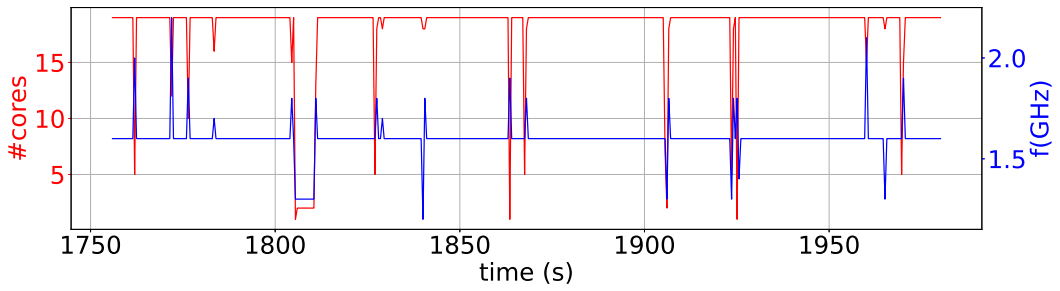
— L'apprentissage n'est pas suffisamment long pour différencier les deux phases et converger vers les deux configurations optimales.



(a) Évolution de l'efficacité énergétique.



(b) Évolution des performances, et sortie de l'auto-encodeur i.e. ID de phase.



(c) Actions du contrôleur i.e. configuration.

FIGURE 4.22 Zoom post-entraînement _ Traces de fonctionnement du contrôleur, pour SRAD.

— La valeur du CpJ de la phase haute étant largement supérieure à celle de la phase basse, l'apprentissage est biaisé par cet écart. En effet, la fonction de récompense étant directement proportionnelle au CpJ, l'apprentissage sera plus important pour la phase haute. Une nouvelle police de reward pourrait améliorer cela.

Gains : En comparant avec les gouverneurs Linux disponibles sur notre serveur Intel Xeon, notre méthode produit les résultats suivants : -12.3%, 7%, -12.9% et -12.8%, respectivement comparés aux gouverneurs *Performance, Powersave, Ondemand, Conservative*. Ces résultats sont inférieurs à nos attentes, et s'expliquent par le fait que l'IA n'a pas appris correctement à assigner la configuration optimale de la phase basse. Lorsque nous comparons les résultats pour chacune de ces phases dans le tableau 4.6, on constate ce manque à

gagner. En effet, à part en comparaison avec le gouverneurs *Conservative* où nous sommes légèrement inférieur (-3.6%), nous obtenons des gains en efficacité énergétique pour la phase haute (i.e. phase 2), pour laquelle la configuration sélectionnée par notre contrôleur est sa configuration optimale. Ensuite, excepté pour le gouverneur *Powersave* pour lequel nos gains vont jusqu'à 16.6%, nous obtenons des résultats négatifs sur la phase basse (i.e. phase 1). Or, cette phase basse a une durée d'exécution plus longue que la phase haute (visible sur la figure 4.22b), ce qui réduit logiquement nos gains moyens.

Phases		Phase 1	Phase 2	Global
Résultats	CpJ	121.3	386.0	169.6
vs. Performance	δ	-11.5%	6.0%	-12.3%
vs. Powersave	δ	3.7%	16.6%	7%
vs. Ondemand	δ	-11.8%	1.5%	-12.9%
vs. Conservative	δ	-10.6%	-3.7%	-12.8%

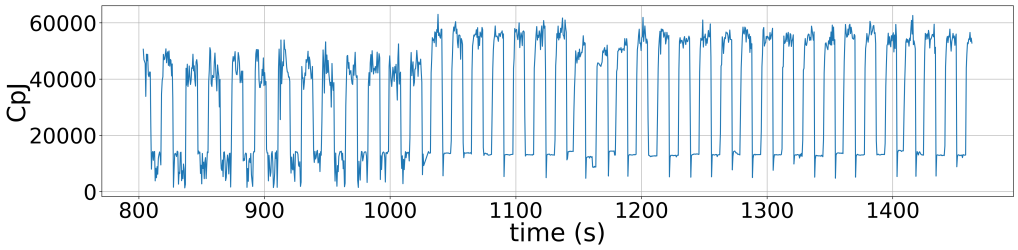
TABLE 4.6 Différences (δ) en efficacité énergétique (CpJ) de notre contrôleur, en comparaison avec les gouverneurs Linux : Powersave, Performance, Ondemand et Conservative.

Afin d'apporter des éléments de réponse sur le problème de l'apprentissage des configurations optimales des phases, nous proposons de tester notre modèle sur une application synthétique aux caractéristiques différentes de SRAD. Cette application synthétique possède également deux phases, une phase haute et une phase basse, aux durées d'exécutions similaires, mais ayant des configurations optimales éloignées.

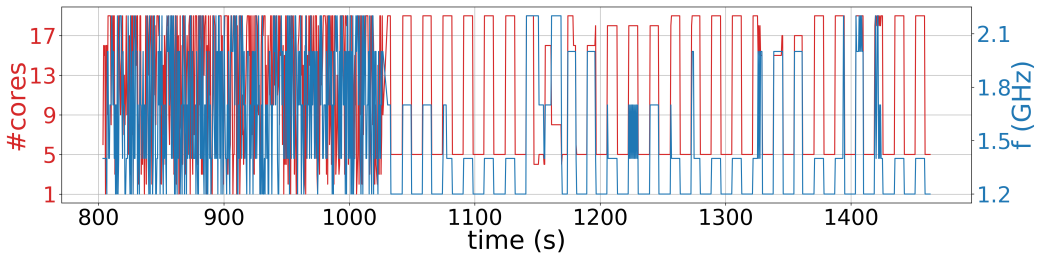
4.3.2.2 Évaluation sur une application synthétique

Nous évaluons maintenant notre design avec une application synthétique à deux phases. Elle est dérivée de deux applications statiques synthétiques. L'une des phases est de type *compute-intensive*, similaire à *C100M0*, et sa configuration optimale est {19 cœurs, f=2.2GHz}. La seconde phase est de type *memory-intensive*, et possède la configuration optimale suivante : {4 cœurs, f=1.3GHz}. Ainsi, les 2 phases possèdent des caractéristiques très différentes, contrairement aux phases de SRAD dans la section précédente.

Comme visible sur la figure 4.23a, les deux phases sont clairement identifiables à travers le tracé des CpJ, reflétant les différents types de charges de travail exécutées. Bien que non représenté sur ces graphiques, l'auto-encodeur génère les informations relatives aux phases dès le début de l'exécution. Ainsi, le réseau de neurones du contrôleur en-ligne est entraîné directement avec cette donnée. Comme précédemment, cet entraînement est réalisé durant les 1024 premières secondes.



(a) Évolution de l'efficacité énergétique.



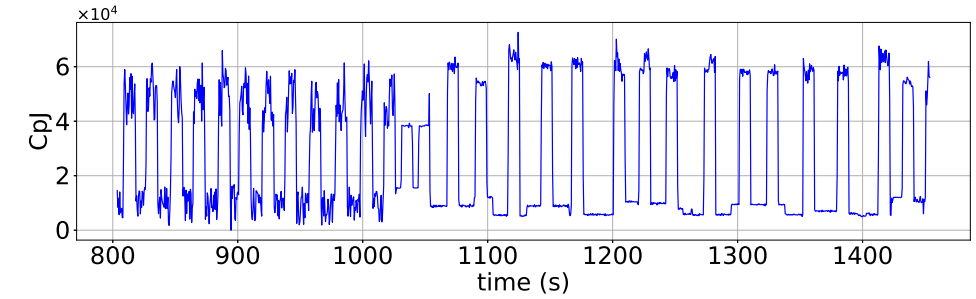
(b) Actions du contrôleur i.e. configuration.

FIGURE 4.23 Traces de fonctionnement du contrôleur, pour le benchmark à 2 phases.

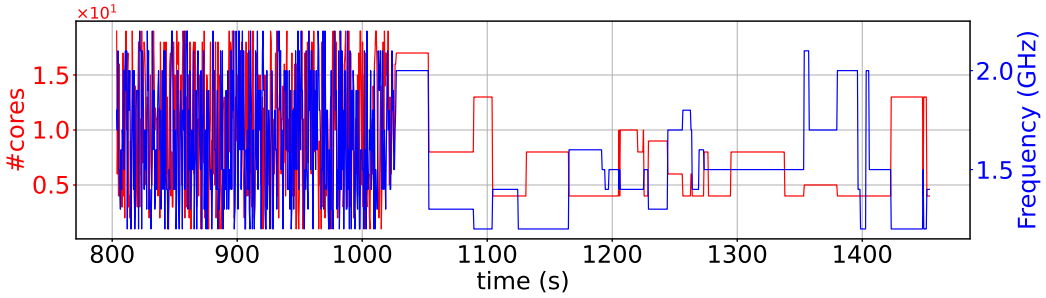
Dès le début de la phase d'exploitation (i.e. à partir de 1024s), on observe que le contrôleur identifie seulement 2 phases, et adapte la configuration en fonction, validant le fonctionnement de l'auto-encodeur. Le contrôle en-ligne sélectionne 19 et 5 cœurs respectivement pour les deux phases, ce qui correspond aux configurations optimales connues. Quant à la fréquence, cette dernière oscille entre 1.2GHz et 1.4GHz ce qui n'est pas suffisamment proche des configurations connues, laissant suggérer qu'une durée d'entraînement plus longue permettrait d'améliorer l'efficacité énergétique.

Gains : En comparant avec les gouverneurs Linux disponibles, notre méthode produit les gains suivants : 51%, 7%, 51% et 50%, respectivement comparés aux gouverneurs *Performance, Powersave, Ondemand, Conservative*. Les gains potentiels de chaque phase ont été mesurés individuellement via une caractérisation hors-ligne, et une amélioration de l'entraînement pourrait à terme produire des gains allant de 34% à 136%, toujours en comparant aux gouverneurs Linux.

Vérification : Enfin, nous proposons ici de valider l'intérêt de l'auto-encodeur en renouvelant la dernière expérience mais en implémentant le modèle du contrôleur sans l'auto-encodeur. Comme visible sur la figure 4.24, contrairement à la figure 4.23, rien ne met en évidence une corrélation entre les actions menées et les changements de phase. De plus, nous obtenons une valeur moyenne du CpJ après entraînement inférieur de 34% comparé à l'expérience précédente (i.e. avec auto-encodeur). Ainsi, l'auto-encodeur et plus généralement



(a) Évolution de l'efficacité énergétique.



(b) Actions du contrôleur i.e. configuration.

FIGURE 4.24 Traces de fonctionnement du contrôleur, pour le benchmark à 2 phases.

l'information sur la phase d'exécution permet comme attendu d'améliorer l'apprentissage du contrôleur.

4.3.2.3 Discussion

Globalement, nous pouvons donc confirmer que notre système de contrôle enrichi d'un module de détection de phase est capable d'apprendre à reconnaître les phases de fonctionnement d'une application, et d'ajuster la configuration du système en fonction de cette information afin d'optimiser l'efficacité énergétique du calcul.

L'auto-encodeur proposé se montre particulièrement efficace pour réaliser cette détection de phase en temps réel. Cependant, dans la solution actuelle, l'auto-encodeur est entraîné pour une seule application et avant d'être inclus dans le système de contrôle. Cela va donc à l'encontre de l'avantage premier de l'apprentissage temps réel amené par le RL. Plusieurs solutions à cette limitation peuvent faire l'objet de futurs travaux, telles que la conception d'un auto-encodeur généraliste, ou l'inclusion de la détection de phase dans le RL.

Ensuite, des limitations apparaissent quant aux degrés de précision que peut tolérer notre modèle. En effet, on remarque que pour l'application SRAD dont les phases possèdent des configurations optimales similaires, notre modèle ne fait pas de distinctions entre ces deux configurations, menant à des pertes en efficacité énergétique. Au contraire, lorsque

les configurations optimales sont éloignées, comme pour notre application synthétique, l'apprentissage converge correctement et permet de gagner en efficacité énergétique. De futurs travaux consisteront à définir précisément ces limitations, et modifier en fonction les paramètres et la politique d'apprentissage du contrôleur.

Enfin, aux hypothèses d'utilisation de départ s'ajoute la contrainte de la période d'échantillonnage devant permettre à la fois de réaliser les différents traitements du contrôleur (i.e. entraînement du RL, inférences des réseaux de neurones, etc.), ainsi que de suivre l'évolution d'une application (i.e. période supérieure à la durée d'exécution d'un chunk). Ces différentes contraintes posent des difficultés pour sélectionner des applications réelles éligibles pour notre contrôle, expliquant le nombre limité de résultats. De futurs travaux consisteront à repousser ces limitations, afin de pouvoir appliquer notre méthode à un plus grand ensemble d'applications.

4.4 Résumé

Après avoir montré le potentiel des chunks comme métrique fiable pour le suivi des performances et de l'efficacité énergétique, nous nous sommes intéressés à l'exploitation des solutions de type réseau de neurones pour proposer une méthodologie de contrôle des applications OpenMP.

Ainsi, un outil basé sur les auto-encodeurs est présenté pour détecter et identifier, si existantes, les phases d'exécution d'une application. Cet outil se montre particulièrement efficace sur des applications à deux phases, mais de futures études devront être menées pour valider ce fonctionnement sur des applications possédant un plus grand nombre de phases.

Ensuite, une méthode de contrôle de configuration système est proposée pour optimiser à la fois le nombre de ressources attribuées au workload et leur fréquence de fonctionnement. Ce contrôle est construit autour d'un apprentissage par renforcement inspiré du DQN, et permet une optimisation automatique de la configuration durant l'exécution de l'application à optimiser sans entraînement préalable (hormis pour l'auto-encodeur si utilisé). Cette méthode est validée sur un ensemble d'applications synthétiques conçues pour représenter la dualité *memory-intensive vs. compute-intensive*. Des gains significatifs en efficacité énergétique ont été obtenus en comparant notre méthode aux gouverneurs classiques de Linux. De futurs travaux devront confirmer ces résultats sur des benchmarks d'applications réelles. En effet, bien que probantes, nos validation reposent principalement sur des applications synthétiques. Il sera donc nécessaire de montrer la robustesse de notre solution sur des benchmarks reconnus. Enfin, au vu de la littérature récente (e.g. AdaMD [18]), un développement de notre solution vers la gestion multi-tâches sera intéressant à explorer. En effet, notre solution

est par essence *application-specific* du fait de l'utilisation des chunks, or la tendance se veut plus généraliste, avec un contrôle dynamique qui soit efficace pour toute nouvelle application exécutée sur le système. Des techniques telles que le transfert d'apprentissage [155] pourraient permettre de pallier ce défaut.

Enfin, le contrôle multi-phase a été testé sur une application réelle ainsi qu'une application synthétique. Les résultats confirment que le RL peut permettre un contrôle adaptatif du calcul parallèle. En particulier, l'ajout d'un module d'identification de phases permet au système d'apprendre les différentes actions à mener pour optimiser l'efficacité énergétique du système de calcul. De plus, cet apprentissage ne nécessite que 2048 données, ce qui est peu au regard des bases de données connues (ex. 70000 données pour MNIST). Cependant, ces résultats ne sont pas optimaux et nécessitent de futurs travaux afin de rendre le système plus précis et plus robuste. En effet, on a remarqué un défaut d'apprentissage du contrôleur pour l'application SRAD qui possède deux configurations optimales similaires. Ainsi, des recherches plus approfondies permettront d'ajuster la politique d'apprentissage du contrôleur, ainsi que le système de contrôle global. Un point d'amélioration important est la fréquence de fonctionnement du contrôleur. En effet, ce dernier fonctionne à 2Hz, soit une action toutes les 0.5s. Cette période d'échantillonnage limite le nombre de phases d'exécution détectables à des phases d'une durée minimale de 1s. De plus, bien que le nombre de données nécessaires pour un apprentissage correct soit de seulement 2048, cela équivaut à 1024s d'exécution et représente donc une limite supplémentaire à l'usage de notre méthode i.e. contrôle d'application s'exécutant durant plusieurs heures afin de négliger cette période de pur apprentissage.

Ce chapitre concernait les travaux effectués dans le cadre de notre premier axe de recherche. Le chapitre suivant adresse notre second axe centré sur l'optimisation de la conception matérielle. Plus précisément, il est question d'optimiser les designs de NoC afin d'améliorer leur efficacité énergétique, ou tout autre métrique considérée. En effet, bien que motivée par l'optimisation de la consommation des systèmes, la solution proposée se voudra généraliste et non limitée à un seul levier d'amélioration.

Chapitre 5

Optimisation des topologies de réseaux de communication sur puce

Sommaire

Après s'être intéressé à l'optimisation temps réel d'applications parallélisées, nous nous intéressons à l'optimisation de la conception matérielle des supports d'applications. Pour cela, nous étudions plus particulièrement la conception de réseaux sur puce (NoC, pour *Network-on-Chip*), permettant d'interconnecter les éléments d'un même système sur puce (SoC, pour *System-on-Chip*). En effet, les SoC sont entre autres utilisés pour les systèmes embarqués, et sont donc particulièrement concernés par l'enjeu de l'efficacité énergétique.

Nous cherchons ici (ainsi que dans le chapitre 6) à démontrer qu'il est possible d'exploiter les méthodes d'apprentissage profond pour améliorer la conception des NoC, et plus globalement, la conception de réseaux de communication. Par amélioration, il est question

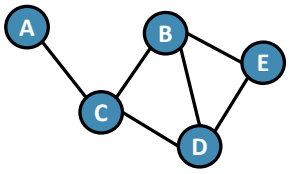
de designs optimisés selon des critères définis par l'utilisateur, dont fait partie l'efficacité énergétique.

Ce chapitre s'intéresse plus particulièrement à l'optimisation des topologies de NoC, i.e. structure du graphe. En effet, la topologie du NoC joue à elle seule un rôle important dans le fonctionnement du NoC, en fixant les bases des chemins possibles à emprunter par les flux de communication. Ce chapitre est appuyé par la publication suivante : [106].

5.1 NoC et théorie des graphes

Afin de concevoir un outil de réduction d'espace de conception des NoC, il est important de définir le type des données qui seront traitées. Pour cela, nous faisons l'analogie entre les NoC et les graphes. En effet, la description d'un NoC est analogue à celle d'un graphe, où les connexions et routeurs de l'un correspondent aux arêtes et sommets de l'autre. Cela nous permet de réduire notre problème de génération de NoC à un problème de génération de graphes.

À partir de cette analogie, nous pouvons nous référer à différents travaux sur la génération de graphes [28, 60, 172, 71]. Pour la plupart, la représentation choisie d'un graphe est sa matrice d'adjacence. Cette matrice offre une représentation formelle, non ambiguë d'un graphe. En effet, les propriétés essentielles comme le nombre d'arêtes et le degré d'un sommet peuvent être directement extraites de cette représentation.



Graphe

	A	B	C	D	E
A	0	0	1	0	0
B	0	0	1	1	1
C	1	1	0	1	0
D	0	1	1	0	1
E	0	1	0	1	0

Matrice d'Adjacence (A)

	c1	c2	c3
A	-1	0.1	3.2
B	0.2	5.1	-2.8
C	0.5	0.3	5
D	1.2	-1.5	-0.5
E	-0.6	-2	1.1

Matrice des Caractéristiques (X)

FIGURE 5.1 Graphe : représentations

Il existe différentes matrices permettant de représenter les graphes selon différents paramètres. Il y a la matrice d'adjacence (notée A) qui constitue la base de toute représentation de graphes. En effet, cette matrice contient les informations de type structurel du graphes, et décrit la topologie du graphe en indiquant le placement et la direction des connexions. Il existe aussi la matrice des paramètres (notée X) qui contient les informations sur différentes caractéristiques définissant le nœud d'un graphe. Dans notre cadre des NoC, nous pourrons parler du type de routeurs ou encore de la taille des buffers comme autant de paramètres

disponibles dans la matrice X. Enfin, il est possible d'enrichir la matrice d'adjacence avec une représentation en trois dimensions, où la troisième dimension pourra contenir des informations caractérisant les connexions. Cette dernière possibilité n'est pas exploitée dans nos travaux, mais doit être connue dans la perspective de futurs travaux.

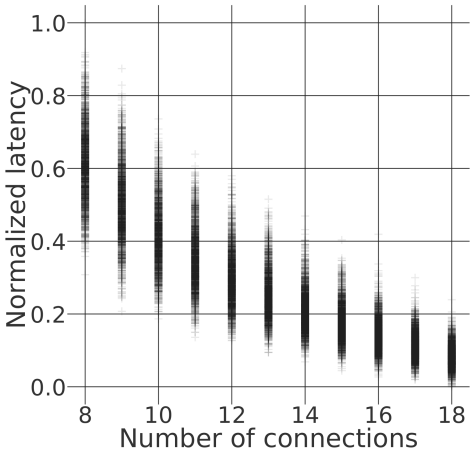
Sur la figure 5.1 est proposé un exemple de graphe simple afin d'illustrer les notions de matrice d'adjacence A et matrice de caractéristiques X (aussi appelée matrice de paramètres). Un NoC de n routeurs pourra donc être représenté par une matrice d'adjacence de taille nxn, où chacun des n^2 éléments est un booléen traduisant de la présence ou non d'une connexion entre deux routeurs. La matrice de paramètre X sera de dimension nxf, avec f le nombre de paramètres décrivant un routeur. Le type des éléments de X dépend des paramètres représentés.

5.2 NoC : topologie et performances

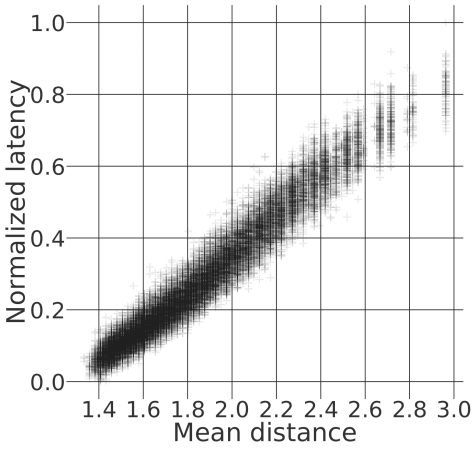
Nous nous concentrons ici sur les attributs de NoC relatifs à la topologie. Ces attributs comprennent le nombre de routeurs, le nombre de connexions, le nombre de connexions par routeur (i.e. le degré du routeur), etc. De plus, les performances des NoC sont évaluées pour un routage statique (voir section 5.4.1.2) et différents trafics synthétiques. Un trafic est caractérisé par la répartition de sa charge – e.g. uniforme, hotspot – et son taux d'injection (TI) (i.e. le volume de sa charge). Plusieurs métriques peuvent servir à évaluer les performances d'un NoC. Ici, nous choisissons la latence du réseau, généralement corrélée au débit et à la bande-passante du réseau, ce qui en fait une métrique pertinente de la performance d'un NoC.

Sur la figure 5.2, plusieurs graphiques sont présentés pour démontrer l'impact des attributs de topologie sur les performances d'un NoC. Les résultats présentés correspondent à une base de données de NoC à 9 routeurs. Ils ont été évalués avec le simulateur Ratatoskr [81], pour les trafics uniforme (voire figures 5.2a et 5.2b) et hotspot (figures 5.2c et 5.2d). On considère un taux d'injection de 10%. Les figures 5.2a et 5.2c montrent la latence en fonction du nombre de connexions des NoC simulés. Sur les figures 5.2b et 5.2d sont affichées les valeurs de latence en fonction de la distance moyenne entre les routeurs. La distance moyenne d'un NoC correspond au nombre moyen de routeurs qu'un message doit traverser avant d'atteindre sa destination. Cette valeur est donc impactée par le routage implémenté.

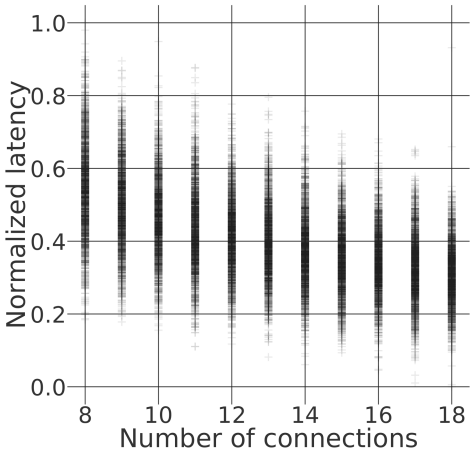
On remarque un impact significatif du nombre de connexions et de la distance moyenne sur les performances du réseau, soumis à un trafic uniforme. Bien que l'on puisse s'attendre à des résultats similaires (i.e. plus on dispose de connexions, meilleure est la bande-passante, et moins la distance moyenne est grande, plus la latence est faible), la même conclusion n'est



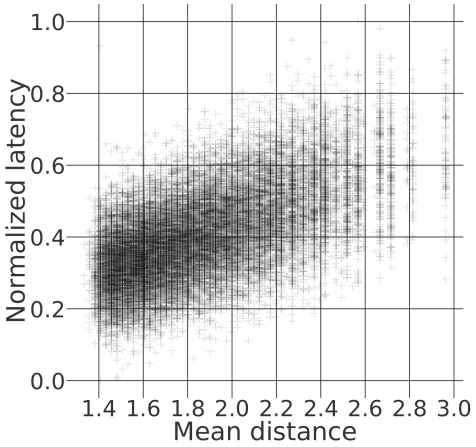
(a) Impact du nombre de connexions. Trafic = uniforme, TI=10%



(b) Impact de la distance moyenne. Trafic = uniforme, TI=10%



(c) Impact du nombre de connexions. Trafic = hotspot, TI=10%



(d) Impact de la distance moyenne. Trafic = hotspot, IR=10%

FIGURE 5.2 Évaluation des performances de NoC à 9 routeurs.

pas possible lorsque le réseau est soumis au trafic hotspot. En effet, bien que la tendance globale soit similaire, avec le trafic hotspot on remarque une plus grande disparité de latence, pour des valeurs d'attributs identiques.

Alors que l'analyse des résultats pour un trafic uniforme est intuitive, on remarque que le trafic hotspot demande une étude plus approfondie. En effet, on ne peut pas extraire de relation de causalité évidente entre les attributs considérés et les performances des NoC. De plus, on peut aisément admettre que des conclusions similaires peuvent être attendues pour des trafics plus complexes, en particulier si l'on considère la nature hétérogène des SoC. D'où l'idée d'entraîner un modèle génératif qui peut apprendre des corrélations non intuitives entre les paramètres et les performances de NoC.

5.3 Problématique et approche

Dans ce chapitre, nous nous intéressons donc à la conception de topologies de NoC optimisées et tentons de répondre à la question suivante : *Comment améliorer la conception des réseaux sur puce, dont l'espace de conception est de plus en plus étendu, à l'aide d'outils d'apprentissage profond ?*

Nous proposons d'exploiter le concept de réseau antagoniste génératif (GAN) pour créer un outil d'assistance à la conception de NoC. Cet outil doit pouvoir fournir au concepteur un ensemble de NoC optimisés selon certains critères, afin de réduire l'espace de conception.

5.3.1 Représentation des données

Pour notre approche, nous adressons le problème de conception de NoC comme un problème de conception de graphes. Ce passage vers le domaine des graphes nous permet d'identifier, à l'aide de techniques d'apprentissage, des propriétés non triviales des graphes relatives aux métriques de performances des réseaux, e.g. la latence moyenne d'envois de paquets et le seuil de saturation du réseau.

Notre intérêt se concentre sur la topologie des NoC. Ainsi, nous choisissons d'utiliser la matrice d'adjacence A comme représentation des données. En effet, comme expliqué en amont section 5.1, la matrice d'adjacence est une représentation compacte de la topologie d'un graphe, décrivant la manière dont les sommets sont connectés. Différentes caractéristiques clefs peuvent être extraites de cette représentation, comme le nombre de connexions, le degrés des sommets, la distance entre deux sommets, etc. La matrice d'adjacence est donc une représentation à la fois simple et riche d'une topologie de graphe. De plus, une caractéristique de cette matrice est qu'elle possède un axe de symétrie sur sa diagonale (i.e. haut-gauche vers bas-droit). Cette propriété découle de notre choix de ne considérer que des connexions bidirectionnelles entre les routeurs. Cela en fait un choix particulièrement pertinent pour notre première implémentation de modèle génératif, puisque ce sera le premier motif que notre réseau de neurones apprendra avant de converger vers plus de détails. De ce fait, c'est une représentation idéale pour notre problème.

Enfin, puisque nous implémentons des routeurs possédant quatre connexions extérieures — généralement nommées Nord, Sud, Est, Ouest, plus le port Local — nous fixons le degré maximum de nos graphes à 4.

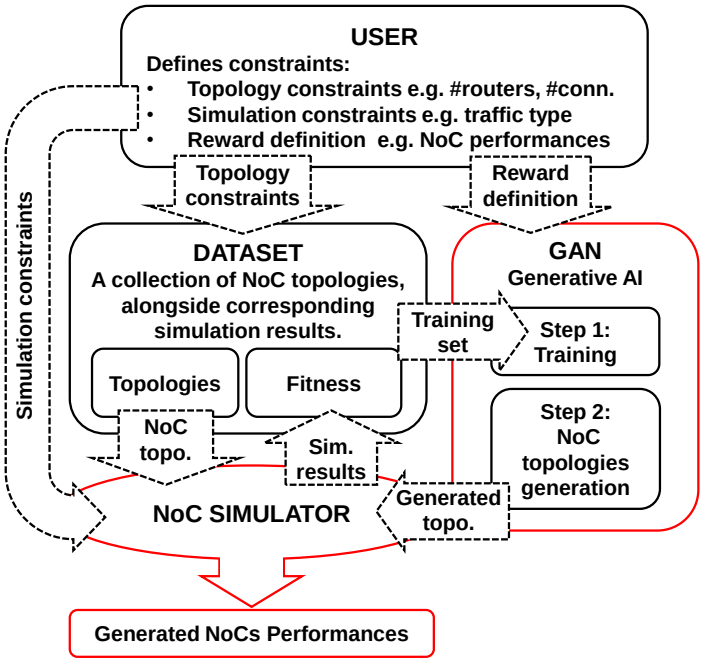


FIGURE 5.3 Le framework GANNoC.

5.3.2 Framework

Notre solution repose sur deux parties essentielles : (1) un réseau de neurones qui a pour objectif d'apprendre à générer des designs de NoC, et (2) un simulateur de NoC qui est utilisé pour évaluer les designs de NoC. Comme illustré sur la figure 5.3, notre diagramme de conception débute les contraintes imposées par l'utilisateur. On distingue trois types de contraintes :

— Les contraintes de conception du NoC, décrivant l'espace de conception de départ, i.e. type de topologie (fixe ou variable), les types de routeurs disponibles, les types de connexions disponibles, etc. ;

— Les paramètres de simulation, définissant l'environnement d'exécution dans lequel seront évalués les NoC. Dans notre cas, cela correspond exclusivement aux paramètres de trafic, e.g. type de trafic, taux d'injection ;

— La définition de la fonction de récompense. Ou exprimé différemment, les métriques de NoC à optimiser, e.g. seuil de saturation, puissance consommée.

À partir de ces contraintes, une base de données est construite. Deux étapes sont nécessaires pour produire une base de données. Premièrement, un ensemble de NoC doit être défini en fonction des contraintes de conception. Puis cet ensemble de NoC est évalué via le simulateur, selon les contraintes de simulations imposées par l'utilisateur. Ainsi, un

élément de la base de donnée comprend une description d'une architecture de NoC, plus ses résultats de simulation. De plus, on note qu'une base de données correspond à un ensemble de contraintes utilisateurs. Si ces dernières sont modifiées, un nouveau dataset devra être généré.

5.3.3 Le RWGAN pour de la génération optimisée

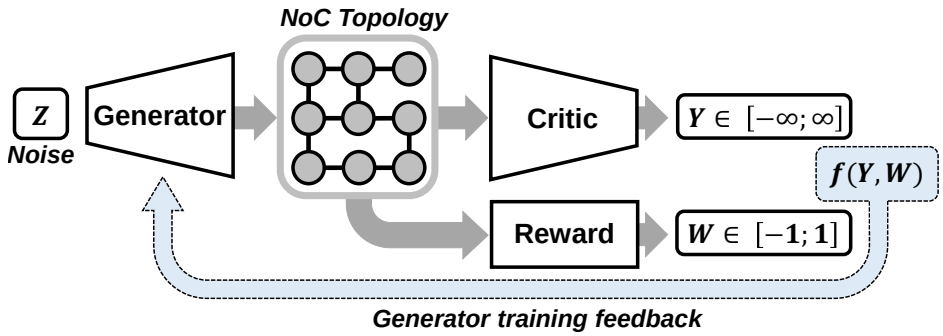


FIGURE 5.4 Schéma du Reward-Wasserstein GAN et la fonction de perte du générateur $f(Y,W)$.

Le GAN utilisé ici étend le principe des WGAN évoqué en section 2.5.3. Comme illustré sur la figure 5.4, l'architecture de GAN proposée consiste en un ensemble de trois réseaux de neurones (contre deux pour un GAN classique) : un *générateur G*, un *critique C* et un *reward R*. Les deux premiers sont les blocs basiques du WGAN. Le bloc reward a pour objectif d'évaluer les propriétés d'un NoC donné.

Le réseau reward R est entraîné indépendamment et en amont de l'entraînement du WGAN ($G \cup C$) pour estimer une fonction de récompense. Il est ensuite utilisé dans l'entraînement du WGAN pour guider l'apprentissage du générateur. Cette méthode d'apprentissage du générateur est décrite sur la figure 5.4. L'architecture finale, i.e. $G \cup C \cup R$, est appelée Reward-WGAN (RWGAN).

Le réseau Générateur Le générateur prend en entrée un élément de l'ensemble Z, correspondant à un espace aléatoire. Il génère en sortie une description de NoC. Cette description doit correspondre, après apprentissage, aux critères utilisateurs évoqués précédemment : les contraintes de conception avec les métriques optimisées. Le premier critère est appris via l'apprentissage du GAN basique. Le second est appris via la prise en compte de la sortie du reward dans l'apprentissage du générateur. En effet, le générateur apprends à maximiser la sortie du reward (i.e. W) qui correspond à la valeur de la fonction de récompense estimée pour les NoC générés.

La nouvelle fonction de perte (*loss*) du générateur ainsi obtenue est formulée dans l'équation 5.1.

$$L_G(z_i) = (1-\lambda)L_C(G(z_i)) + \lambda[\beta L_R(G(z_i))] \tag{5.1}$$

où G désigne le générateur, z_i représente un élément de l'espace aléatoire Z, λ est le ratio entre la perte venant du reward (L_R) et celle venant du critique (L_C), et β est un coefficient permettant d'équilibrer les deux pertes, en complément de λ. En effet, alors que la fonction de perte du critique n'est en théorie pas bornée, celle du reward est limitée. D'où le besoin d'un coefficient supplémentaire devant être modifié selon la façon dont converge les valeurs de *loss* du critique. Ainsi, la fonction de perte du générateur est une combinaison linéaire des pertes venant de la sortie du critique et de la sortie du reward (i.e. $f(Y,W)$ sur la figure 5.4). Pour lisser la transition de l'entraînement depuis un l'apprentissage des caractéristiques globales de NoC vers les caractéristique spécifiques désirées, le coefficient de combinaison linéaire λ initialement à 0 est progressivement incrémenté durant l'entraînement, jusqu'à atteindre la proportion souhaité e.g. 0.9 pour une fonction de perte correspondant à 90% à celle du reward, et 10% à celle du critique. Il est important de garder une proportion non négligeable de la perte liée au critique pour préserver l'apprentissage des caractéristiques de base d'un NoC.

Le réseau Critique La fonction de perte du critique est identique à celle présentée dans 2.5.3 pour le *WGAN-GP* (voir l'équation 2.2). L'architecture du critique dépendra de son utilisation. Plus de détails seront apportés dans la prochaine section où les différentes expérimentations menées sont décrites.

Le réseau Reward Le réseau reward possède la même architecture que le réseau critique. Il est utilisé pour guider le générateur durant son entraînement. En effet, alors que le critique assiste le générateur dans son apprentissage des caractéristiques générales de représentation d'un NoC, pour produire des données valides, le reward va préciser cet entraînement pour induire des propriétés spécifiques aux NoC générés. Ces propriétés spécifiques correspondent à la fonction de récompense.

5.4 Preuve de concept

Nous proposons ici de tester notre framework pour l'étude spécifique de topologies irrégulières de NoC à 9 routeurs.

Nous décrivons dans un premier temps le cadre d'expérimentation. Ensuite, nous analysons les NoC générés.

5.4.1 Simulateur utilisé

Ici, nous utilisons Ratatoskr [81] pour évaluer les performances de NoC en fonction du trafic considéré. Ratatoskr a été créé par J.M. Joseph *et al.*. C'est un simulateur de NoC de type *cycle-accurate*, à la fois rapide et flexible, permettant d'évaluer les performances de NoC avec un niveau de précision similaire à l'état-de-l'art [39]. C'est un projet open-sources qui donne la possibilité de mener différentes simulations de NoC, adaptées aux besoins utilisateurs via les paramètres de configurations disponibles. Entre autres, il est très simple de modifier la topologie d'un NoC, et certaines propriétés d'architecture (e.g. taille des buffers, nombre de VC (virtual channel), etc.). Différents trafics sont déjà implémentés par les auteurs, tels que les trafics *uniforme* et *hotspot*, et il est aussi possible d'injecter des traces d'exécution pré-enregistrées. De plus, le taux d'injection est modifiable, ce qui élargit le champ des possibles.

C'est donc cette grandes diversité de simulations possibles, alliée à la rapidité du simulateur qui nous a poussé à l'utiliser (environ 5s pour une simulation de 100k cycles d'un mesh 3x3 soumis à un trafic uniforme, avec un taux d'injection de 10% et un routage XY, sur un Intel E3-1225 à 3.2 GHz).

5.4.1.1 Paramètres du NoC

Dans ce paragraphe, nous listons les paramètres globaux de NoC fixé pour les simulations Ratatoskr. Nous utilisons uniquement les mesures de performances produites par Ratatoskr, ce qui ne requiert aucune information sur la technologie utilisée (c'est cependant nécessaire pour les estimations de puissance et surface). Ratatoskr utilise la méthode de commutation de paquets par *wormhole*. Ici, nous simulons des NoC avec des paquets de 32 *flits*, des buffers de 4 *flits* de type FIFO, une seule VC et pour une horloge de 1 GHz.

5.4.1.2 Technique de routage

Pour évaluer nos NoC, il nous faut définir un algorithme de routage particulier. En effet, comme nous considérons un ensemble important de topologies irrégulières, il nous faut définir dans Ratatoskr un routage universel permettant de router toutes les topologies. Ainsi, nous implémentons la méthode de routage proposée par J.C. Sancho *et al.* [150]. Cette méthode produit un routage statique (i.e. table de routage), efficace et sans risques d'inter-blocage. Dans nos expériences, nous considérons le routage comme une contrainte et non comme un paramètre. De ce fait, nous avons choisi cette méthode pour sa simplicité d'implémentation et l'efficacité de son routage universel. Ainsi, on exploite le fait d'utiliser un simulateur pour se reposer sur une méthode à base de tables de routage. De futures recherches pourront

s'intéresser à des méthodes de routage plus complexes telles que la réduction de tables ou les techniques de routage algorithmique.

5.4.2 Bases de données

La base de données que nous utilisons pour entraîner notre modèle de GAN est un ensemble de matrices d'adjacence de topologies de NoC correspondant aux critères suivants : i) les connexions du NoC forment un chemin entre chaque paire de routeurs, i.e. un ensemble connecté de liens définis par une paire de routeurs ; ii) chaque routeur r doit posséder strictement moins de cinq connexions dans un ensemble C de toutes les connexions d'un NoC, i.e. $degree(r,C) \leq 4$; et iii) toutes les connexions sont bidirectionnelles. L'algorithme 1 décrit une procédure simple pour générer la matrice d'adjacence M d'un NoC.

Algorithm 1: Création de la matrice d'adjacence M d'un NoC

Data: *nR* the number of routers, *nC* the desired number of connections, $\mathscr{P}$ the set of all possible NoC connections $c = [r_0, r_1]$ where r_0 and r_1 are routers, $MaxTry$ the maximum number of consecutive unsuccessful constructions of a connected set $\mathscr{C}$.

Result: The adjacency matrix M of the NoC to create.

```
1  𝒞 ← { } %initially empty set of NoC connections% ;
2  try ← 1 %first try% ;
3  while True do
4      for nC iterations do
5          if ∃c = [r0, r1] ∈ 𝒫, such that (degree(r0, 𝒞)< 4) and (degree(r1, 𝒞)< 4)
            then
6              select c ;
7              add c to 𝒞  %this increases the degrees of both r0 and r1 by
                1 in the set 𝒞% ;
8          else goto line 13  %restart the procedure% ;
9      if 𝒞 is connected then
10         Create M from the set of connections 𝒞 ;
11         return M ;
12     else
13         try++ ;
14         if try < MaxTry  then
15             𝒞 ← {} ;
16         else return ;
```

Nous mettons en œuvre cet algorithme en Python. Le choix de concevoir une base de données homogène suivant le nombre de connexions présentes dans un NoC vient du fait que

la topologie du NoC est connue comme le premier facteur jouant sur la latence, confirmé par notre analyse du trafic uniforme. Chaque NoC de la base de données est simulé avec Ratatoskr pour récupérer ses performances (i.e. latence), et la base de données finale se constitue d'une liste de NoC avec leur nombre de connexions et de leur latence moyenne. Ainsi, une base de données correspond à un trafic particulier.

5.4.3 Résultats

Dans cette section, nous présentons différents résultats obtenus illustrant les performances de notre framework à générer des topologies de NoC adaptées avec des caractéristiques particulières.

5.4.3.1 Base de données d'entraînement

Notre base de données d'entraînement du GAN consiste en un ensemble de NoC à 9 routeurs. Ces NoC possèdent entre 8 et 18 connexions, avec 10000 topologies uniques pour chaque classe (i.e. nombre de connexions). Ainsi, la base de données est homogène selon cette caractéristique. Les données de performance sont obtenues pour un trafic uniforme avec un taux d'injection de 10%.

Pour cette preuve de concept, nous proposons d'entraîner le reward à évaluer le nombre de connexions présentes dans une topologie donnée. En effet, comme montré précédemment dans la section 5.2, sous un trafic uniforme, la latence d'un NoC est directement corrélée au nombre de connexions. Ainsi, la fonction de récompense estimée par le reward est une fonction qui, pour une matrice d'adjacence donnée, sort une valeur de récompense correspondant au nombre de connexions.

5.4.3.2 Architecture des réseaux de neurones

On construit un WGAN et notre RWGAN à partir des mêmes blocs, dont les détails de dimensionnement sont donnés dans le tableau 5.1. Le générateur est un MLP (*Multi-Layer Perceptron*, i.e. réseau dense) à 2 couches cachées. La couche de sortie est ensuite réorganisée pour correspondre au format 2D des matrices à générer (ici, des matrices 9x9). Il prend en entrée un vecteur aléatoire de 100 unités. Le critique et le reward sont chacun composés de 3 couches cachées dont la fonction d'activation est LeakyReLU avec un coefficient de pente de 0.2 pour les valeurs négatives. Les premières et secondes couches sont de type CNN. La troisième couche et la couche de sortie sont des couches denses.

Les dimensions des réseaux sont déterminées expérimentalement. Une méthode d'optimisation (e.g. exploration des hyper-paramètres) pourra faire l'objet de futurs travaux pour

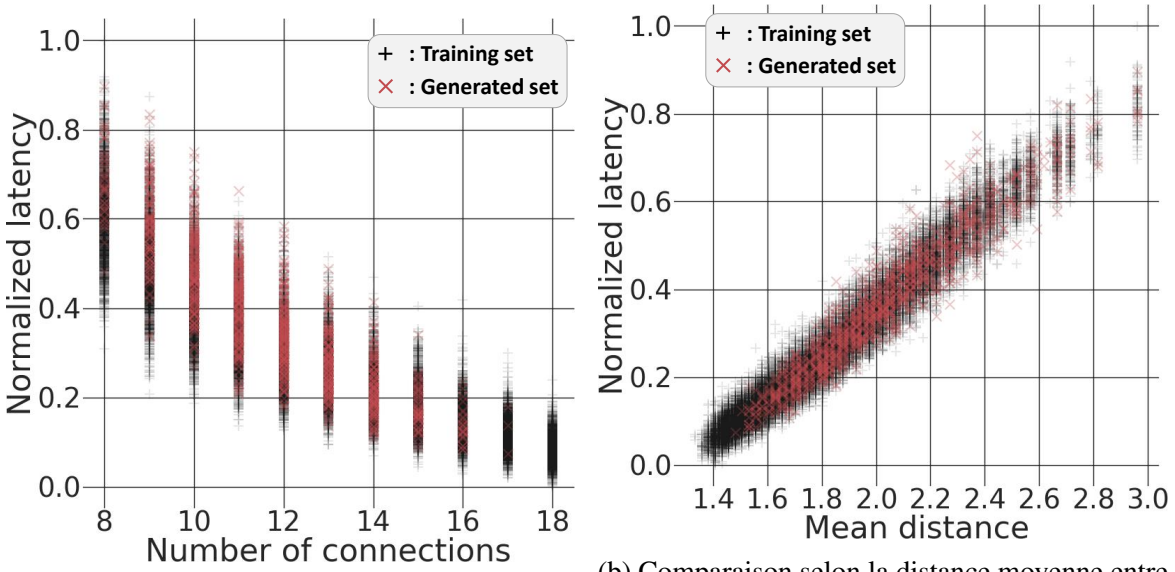
Modules :	Générateur				Critique & Reward				
Couches :	Entrée	L1	L2	Sortie	Entrée	L1	L2	L3	Sortie
Type	Input	Dense			Input	CNN		Dense	
Dimensions	100	162	162	81	81 format : 9x9	64 filtre : 9x9 pas : 1	128 filtre : 3x3 pas : 2	512	1
Fonction d'activation	–	*tanh*			–	*LeakyReLU* ($\alpha = 0.2$)			

TABLE 5.1 Dimensionnement du RWGAN.

un paramétrage plus fin de notre RWGAN. L'entraînement utilise l'optimiseur RMSprop avec 5×10^{-5} comme coefficient d'apprentissage, recommandé dans [70]. Les réseaux de neurones sont implémentés en Python, à l'aide de la bibliothèque Keras [44] basée sur Tensorflow [2].

5.4.3.3 WGAN

Nous nous concentrons dans un premier temps sur l'entraînement du WGAN seul (sans reward). L'entraînement global converge correctement. En effet, une fois entraîné le générateur est capable de créer des topologies de NoC possédant les caractéristiques de base (nombre de routeurs, degrés maximum, graphe connecté) dans jusqu'à 82% des cas.



(a) Comparaison selon le nombre de connexions.

(b) Comparaison selon la distance moyenne entre les routeurs

FIGURE 5.5 Comparaisons entre le dataset d'origine et des échantillons générés par le WGAN. Les valeurs de latence sont évaluées pour un trafic uniforme à 10% de taux d'injection.

La figure 5.5 présente une comparaison entre la base de données d'entraînement, et un ensemble de NoC produits par le générateur entraîné. Toutes les topologies générées sont uniques, ce qui met en exergue les capacités de création du générateur. On voit que la distribution des latences des NoC générés est similaire à celle de la base de données d'entraînement, aussi bien en fonction du nombre de connexions (i.e. Figure 5.5a) que de la distance moyenne entre les routeurs (i.e. Figure 5.5b). Ainsi, on peut en conclure que le générateur apprend correctement les caractéristiques de base des NoC à partir du dataset.

Néanmoins, on remarque que le générateur a des difficultés à produire des topologies de NoC ayant un nombre extrême de connexions i.e. 8 et 18. C'est directement causé par le processus d'apprentissage. En effet, durant l'entraînement, le générateur va apprendre à générer des données ayant une plus grande probabilité d'appartenir à l'espace des données d'entraînement. Cela le fait donc converger vers la tendance moyenne du dataset. Or, comme cet espace de données est homogène en nombre de connexions, la moyenne se trouve aux alentours de 13 connexions. Ajouté à cela, les topologies générées avec un plus grand nombre de connexions ont plus de chances de ne pas correspondre à la contrainte d'être de degrés quatre maximum. D'où la faible proportion de NoC générés avec 18 connexions. À l'inverse, une topologie générée avec 8 connexions a de forte chance de ne pas être connectée. D'où le peu de NoC à 8 connexions.

Résumé : le WGAN est capable d'apprendre à générer des topologies de NoC possédant les caractéristiques générales. Il génère jusqu'à 82% de topologies valides. Toutes les topologies générées sont uniques i.e. ne sont pas présentes dans la base de données d'entraînement.

5.4.3.4 RWGAN

Comme nous avons pu le remarquer section 5.2, sous un trafic uniforme les meilleures performances sont directement liées au nombre de connexions i.e. NoC densément connecté.

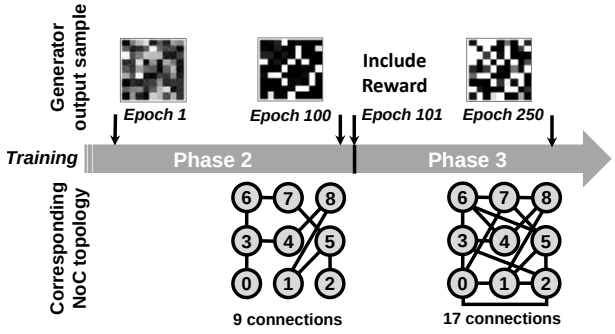


FIGURE 5.6 Impact du reward sur l'apprentissage du RWGAN.

À l'aide de notre réseau Reward, nous entraînons le générateur à produire des topologies possédant un nombre élevé de connexions. De ce fait, les topologies générées par le RWGAN devraient présenter en moyenne de meilleures performances que celles générées par le WGAN, i.e. moyenne du dataset d'entraînement.

Les expérimentations menées sur notre architecture RWGAN comportent trois phases : (1) le réseau reward est entraîné à part pour prédire le score – ici le nombre de connexions – d'une topologie de NoC donnée en entrée i.e. matrice d'adjacence. (2) le RWGAN est entraîné sans le reward (i.e. $\lambda = 0$), de la façon qu'un WGAN classique, jusqu'à ce que l'entraînement se stabilise. Cela permet au générateur d'apprendre en premier les caractéristiques basiques des NoC, via le feedback du critique. (3) la sortie du reward est progressivement incluse dans la boucle d'entraînement du générateur (voire Eq. 5.1). Il est important d'insister sur le fait que le reward n'est alors plus entraîné depuis la fin de l'étape (1). Sur la figure 5.6 est illustré l'impact du reward sur l'entraînement du générateur (étape (2) à (3)). La phase (2) correspond à la période entre les époques 0 et 100, et la phase (3) démarre à l'époque 101 et se prolonge jusqu'à la fin de l'entraînement (époque 250). Durant cette période (3), la répartition entre le retour du critique et celui du reward vers le générateur est progressivement modifiée pour passer de 0% et 100% (respectivement les proportions du reward et du critique), à 90% et 10%. Ainsi, pour une même entrée, le générateur apprend à augmenter le nombre de connexions au fur et à mesure que le reward est inclus dans la boucle d'apprentissage.

Sur la figure 5.7, nous comparons les topologies de NoC générées par le WGAN entraîné seul i.e. sans reward, et le RWGAN, après un apprentissage de 250 époques. Dans un premier temps, nous analysons le nombre de connexions. Comme attendu, le nombre moyen de

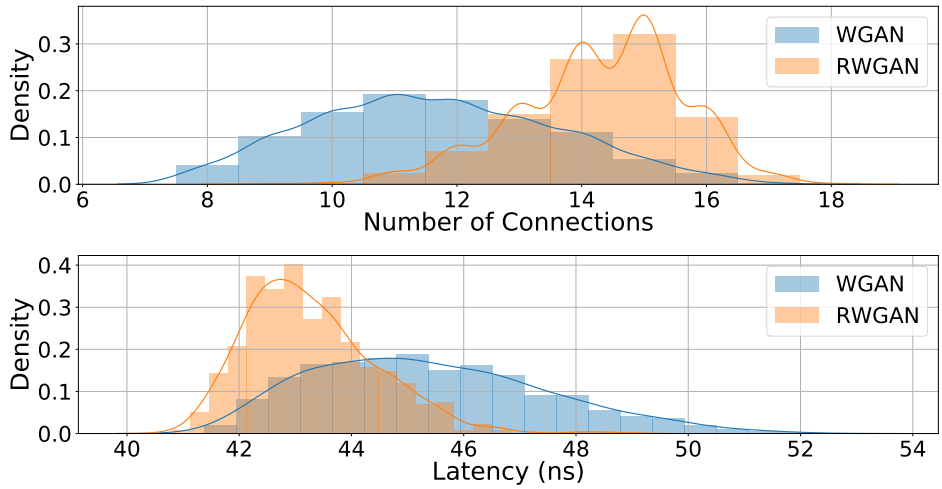


FIGURE 5.7 WGAN vs. RWGAN. Comparaison de la distribution des topologies de NoC générées, selon le nombre de connexions (haut) et la latence moyenne des paquets (bas).

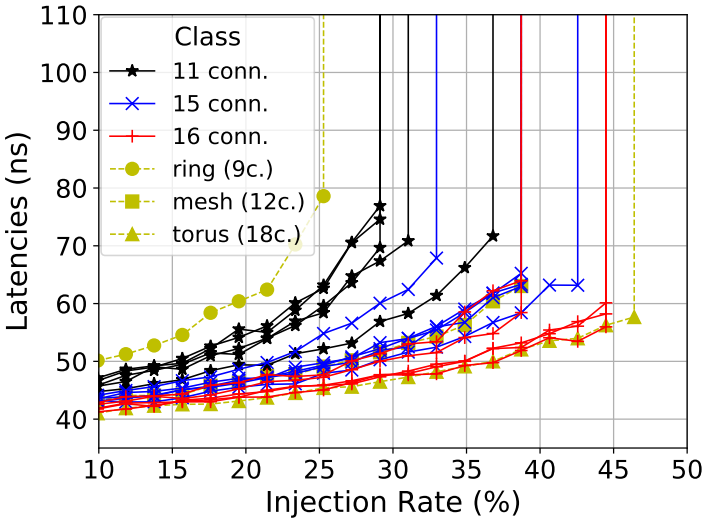


FIGURE 5.8 Courbes de saturation des NoC générés par le RWGAN et de topologies classiques. Les NoC générés sont répartis en trois groupes selon leur nombre de connexions : 11, 15 et 16 connexions.

connexions est augmenté de 36% (de 11 à 15), ce qui représente une augmentation de 36% par rapport aux possibilités min/max (entre 8 et 18). Ensuite, les distributions de la latence sont comparées sur le graphique du bas de la figure 5.7. La latence moyenne de transmission des paquets est diminuée pour passer de 45.4ns à 43.3ns. Normalisée en fonction de l'ensemble des latences possibles (de 40ns à 54ns environ), la latence moyenne est réduite de 0.38 à 0.24, ce qui représente une amélioration de 37% de la latence moyenne.

Nous étudions maintenant les courbes de saturation des NoC générés, pour un trafic uniforme, et pour la latence des paquets. Ces courbes sont obtenues via des simulations Ratatoskr, en parcourant un ensemble de taux d'injection jusqu'à atteindre le seuil de saturation.

Tout d'abord, sur la figure 5.8, nous comparons les performances de trois classes de NoC. Ces classes se différencient par leur valeur de fonction de récompense, ici le nombre de connexions. Ainsi, pour chaque classe nous traçons cinq courbes de saturation correspondant à différents NoC générés. Les trois classes sont respectivement pour des NoC de 11 connexions (noir), 15 connexions (bleu) et 16 connexions (rouge). Pour comparer avec des topologies régulières existantes, nous traçons également les courbes de saturation d'un *ring*, *mesh* et *torus* de 9 routeurs (respectivement 9, 12 et 18 connexions). On peut observer que les NoC avec un nombre plus élevé de connexions ont des performances globalement meilleures, bien que de significatives superpositions existent. En effet, on constate un NoC à 11 connexions possédant un seuil de saturation plus élevé qu'un NoC à 15 connexions. De même, un NoC à 15 connexions est plus performant qu'un NoC à 16 connexions et le *torus* à 18

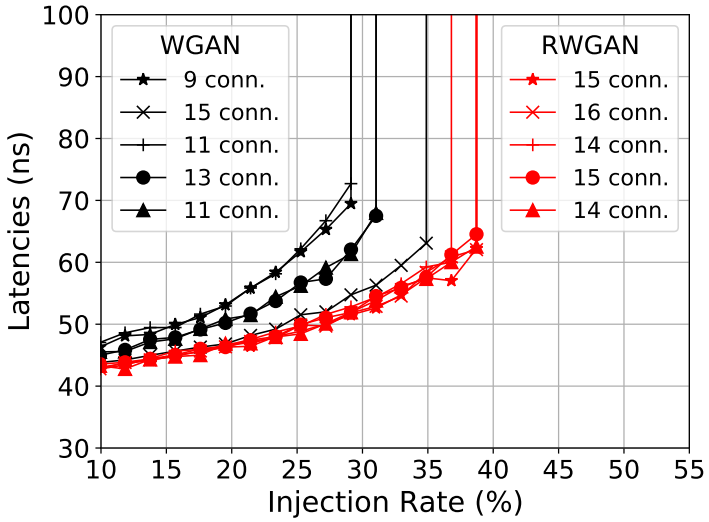


FIGURE 5.9 Courbes de saturation des NoC générés par le RWGAN (rouge) et le WGAN (noir), après un entraînement de 250 époques training.

connexions possède des performances similaires à un NoC à 16 connexions. Cela met en avant le fait que le nombre de connexions n'est pas le seul facteur de performance. Cela suggère également qu'un reward entraîné pour approximer une fonction de récompense plus fine (e.g. latence) peut permettre de produire des NoC aux performances optimisées. Par exemple, le générateur pourrait produire des NoC ayant des latences plus faibles, mais pour un nombre de connexions similaire.

Enfin, sur la figure 5.9, nous proposons de comparer les résultats de saturation de topologies de NoC générées par le WGAN (i.e. courbes noires) et le RWGAN (i.e. courbes rouges) après un entraînement de 250 époques. Ces courbes sont obtenues en donnant au WGAN et RWGAN les mêmes 5 entrées. Sur la figure 5.9, un type de marqueur dénote une entrée particulière. Les topologies ainsi générées sont ensuite simulées avec Ratatoskr, donnant les résultats tracés.

À partir de ces résultats, on peut voir que, malgré des entrées identiques, les générateurs produisent des topologies de NoC aux performances et nombre de connexions différentes. En particulier, les sorties du RWGAN ont de meilleures performances que celles du WGAN. Cela illustre bien l'efficacité du reward. De plus, on remarque une plus grande disparité dans les NoC générés par le WGAN, venant de l'entraînement global qui conduit le générateur à reproduire l'espace des données d'apprentissage.

Résultats : L'architecture de GAN proposée montre des améliorations significatives des NoC générés. Bien que le nombre de générations valides soit réduit par l'utilisation du reward, ce dernier permet d'augmenter la qualité des topologies générées en termes de performances souhaitées (i.e. fonction de récompense, ici le nombre de connexions). En effet, comme le

générateur apprend à augmenter le nombre de connexions, cela détériore ses capacités à respecter la contrainte principale qui est d'avoir des NoC avec un degré maximum de 4. Ainsi, on a une probabilité de 54% d'obtenir une topologie valide (contre 82% sans le reward i.e. WGAN seul). Cependant, nous obtenons une amélioration de 36% relative à la fonction de récompense considérée i.e. nombre de connexions. Cette amélioration significative démontre les facultés du reward à guider l'apprentissage du générateur vers la génération de topologies de NoC possédant les caractéristiques souhaitées.

5.5 Résumé

Pour conclure ce chapitre, nous avons proposé d'utiliser les GAN pour réaliser une réduction d'espace de conception vers un ensemble de données optimisées. Une architecture particulière de GAN permettant de générer ces données est présentée. Cette architecture appelée RWGAN permet ainsi de produire des topologies de NoC optimisées. Nous avons illustré cette utilisation pour la création de NoC possédant un nombre élevé de connexions. La suite de ce travail consiste à implémenter une fonction de récompense moins évidente que le nombre de connexions, telle que la latence du réseau ou encore l'efficacité énergétique du NoC.

Dans le chapitre suivant, nous explorerons une autre façon d'améliorer la conception de NoC, en adressant le problème de la génération de NoC hétérogènes. Une amélioration du RWGAN est proposée, permettant à l'utilisateur d'implémenter autant de fonctions de récompense que souhaité, via une multitude de blocs reward. Cette nouvelle architecture est nommée M-RWGAN pour *Multi-Reward Wasserstein GAN*.

Chapitre 6

Génération de réseaux sur puce hétérogènes optimisés

Sommaire

Nous nous intéressons dans ce chapitre à la génération de NoC hétérogènes. Ici, contrairement au chapitre précédent, nous considérons des NoC à la topologie fixe – i.e. mesh – et nous nous focalisons sur le type des routeurs implémentés. Nous avons pu constater que notre solution GANNoC présentée précédemment est suffisamment généraliste pour pouvoir s'appliquer à différents problèmes. Ainsi, nous proposons d'appliquer cette méthode pour la production de NoC hétérogènes, c'est-à-dire possédant des routeurs de nature différentes, et d'étendre le concept de RWGAN pour considérer plusieurs fonctions de récompense à la fois, i.e. apprentissage multi-objectif. Ce chapitre est appuyé par la publication suivante : [105].

6.1 NoC hétérogènes : motivation et enjeux

6.1.1 Des trafics asymétriques

Dans le but de simplifier le processus de conception des NoC, ces derniers sont généralement conçus en se basant sur des trafics synthétiques uniformes ou très réguliers, e.g. *transpose*, *bitreverse*, etc. [49]. Cependant, la nature des applications exécutées sur un SoC est souvent bien différente [69]. Malheureusement, une compréhension insuffisante du trafic attendu peut mener vers un dimensionnement du NoC non optimal et coûteux. Ainsi, pour illustrer ce propos, nous avons réalisé une étude sur de réelles applications. Nous avons analysé des traces venant de l'outil Netrace [76]. Ces traces sont extraites de l'exécution d'applications PARSEC [24] sur le simulateur M5 [25] pour un système multi-cœurs homogène de 64 cœurs et à mémoire partagée. Les entrées de type *Simmedium* sont utilisées sur l'ensemble de applications PARSEC, mis à part pour les benchmarks *Bodytrack* et *Swaption* pour lesquels les entrées de type *simlarge* sont utilisées.

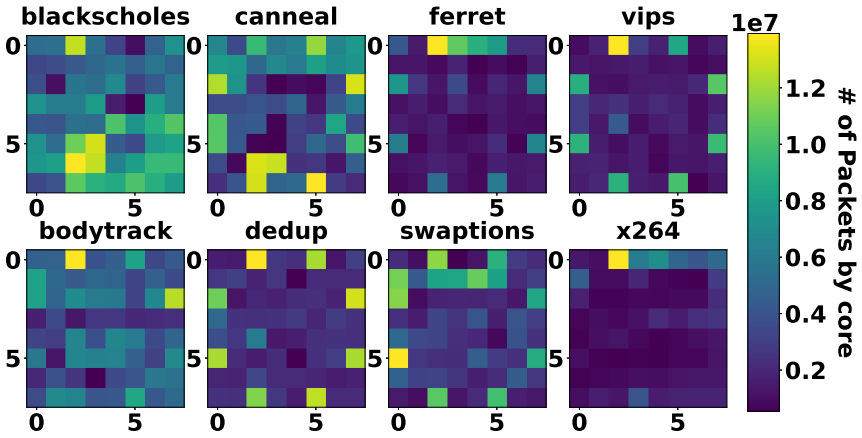


FIGURE 6.1 Nombre de paquets reçus par cœur pour 8 applications du benchmark Parsec exécuté sur 64 cœurs

La figure 6.1 expose le nombre de paquets reçus durant l'exécution de huit applications du benchmark PARSEC. Les 64 cœurs du SoC sont organisés en un *mesh* de taille 8 par 8, avec leurs ID traduits en coordonnées X-Y. Pour chacune des applications, on remarque une large disparité dans le nombre de paquets reçus par les cœurs, sans aucun motif particulier de trafic symétrique. Ces différences montrent que certains routeurs sont plus sollicités que d'autres, et révèle l'utilisation inefficace des ressources du NoC homogène. Il faut noter que

ces hotspots sont observés pour un SoC également homogène. Cependant, les SoC conçus de nos jours sont de plus en plus hétérogènes (CPU, GPU, accélérateurs, ...) pour des raisons d'efficacité énergétique, ce qui apporte de nouvelles sources d'asymétrie dans le trafic. De ce fait, une solution pour améliorer l'efficacité énergétique des SoC est de concevoir des NoC hétérogènes adaptés aux contraintes de trafic.

6.1.2 Un espace de conception immense

Ce travail adresse donc le problème de la conception de NoC hétérogènes optimisés. En particulier, nous proposons une solution permettant de couvrir de grands espaces de conception pour lesquels les méthodes classiques de recherches exhaustives ne sont pas envisageables, dû à un temps de calcul trop important.

En effet, les espaces de conception augmentent rapidement étant donnés l'ensemble des paramètres définissant un NoC hétérogène. Par exemple, prenons un réseau de type mesh 8x8, et considérons seulement le type des routeurs. Pour n possibilités de type de routeur, on obtient le nombre de combinaisons $C = n^{64}$. Ainsi, pour seulement 3 types de routeur, i.e. $n = 3$, nous obtenons $C = 3^{64} \approx 10^{30}$ combinaisons de design possibles. De plus, le type de connexion et la topologie du réseau sont d'autres paramètres pouvant être considérés, menant à une espace de conception immense. De ce fait, une évaluation exhaustive de l'espace de conception par un concepteur de NoC n'est pas concevable.

6.2 Un apprentissage multi-objectif

6.2.1 Données générées

Toujours pour faire le lien avec le domaine des graphes, dans le chapitre 5 précédent nous générons des matrices d'adjacence, et dans ce chapitre nous générons les matrices de paramètres (soit X, pour reprendre la notation dans le domaine des graphes, voire section 5.1). De ce fait, nous considérons ici la génération de NoC hétérogènes pour une topologie fixe (matrice d'adjacence A constante). L'hétérogénéité vient donc des propriétés des routeurs implémentés.

Dans ce travail, nous considérons uniquement comme paramètre le type des routeurs. Ainsi, la matrice X décrit la classe de chacun des routeurs implémentés. Pour décrire la classe des routeurs, une description en vecteur *one-hot* est utilisée. En classification, un vecteur one-hot est une représentation binaire de la classe et fonctionne de la manière suivante : pour n classes possibles de données, le vecteur one-hot possède n éléments, appelé bit. Chaque bit représente une catégorie possible. Ainsi, pour encoder la classe d'une donnée avec un vecteur

one-hot, il suffit de mettre à 1 le bit correspondant à la classe, et laisser les autres à 0. Cette méthode de classification se différencie des méthodes d'encodage numérique (e.g. classeA = 1, classeB = 2, classeC = 3) qui contiennent de manière intrinsèque un ordonnancement des classes, et donc un biais de valeur. Dans notre cas, nos catégories ne possèdent pas de relation d'ordonnancement, ainsi l'encodage neutre one-hot est l'idéal. Cette description permet donc un meilleur apprentissage de la part du réseau de neurones pour une classification ne possédant pas de relation d'ordonnancement entre ses catégories.

6.2.2 Architecture du M-RWGAN

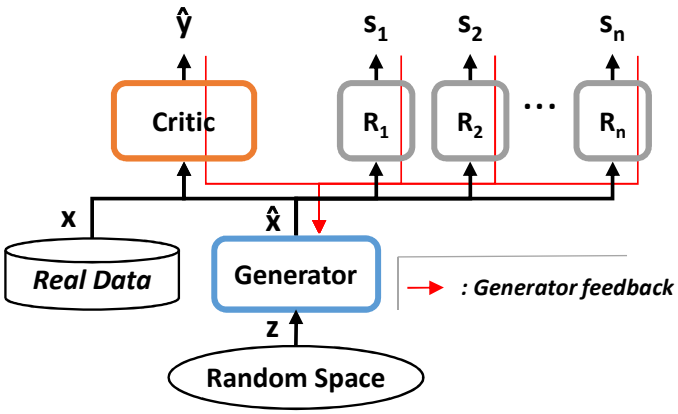


FIGURE 6.2 Schéma du Multi-Objectives RWGAN, avec la descente de gradient de l'apprentissage du générateur en flèches rouges.

Nous proposons ici une amélioration du RWGAN présenté dans le chapitre précédent. Alors que le RWGAN ne comporte qu'un seul objectif d'optimisation – i.e. un reward – l'architecture présentée ici permet d'implémenter plusieurs objectifs d'optimisation pour un même générateur. Nous nommons cette nouvelle architecture Multi-Reward WGAN (M-RWGAN). Comme évoqué dans la section 6.2.1, les données générées dans ce travail sont des matrices de paramètres de NoC hétérogènes. Notre architecture de M-RWGAN est donc utilisée pour générer des configurations de NoC hétérogènes à la topologie fixée (ici, mesh 8x8), optimisées selon différents objectifs. Par exemple, il est possible d'implémenter un M-RWGAN à deux objectifs permettant de considérer à la fois le débit et la puissance consommée, via deux rewards distincts, pour globalement optimiser l'efficacité énergétique.

Description du M-RWGAN Le concept de M-RWGAN étend donc l'architecture du RWGAN en donnant la possibilité d'implémenter plusieurs modules de reward, permettant un entraînement multi-objectif du générateur (voire figure 6.2). Nous restons sur du MLP pour

l'architecture du générateur. Nous nous sommes intéressés au GCN (Graph Convolutional Network [89]) pour l'architecture des modules critique et rewards, puisque ce type de réseau de neurones est spécifique pour le traitement de données de type graphe. Cependant, nous verrons dans les sections de résultats que l'architecture CNN peut parfois s'avérer plus efficace dans l'apprentissage de motifs. L'entraînement du critique est similaire au WGAN-GP traditionnel. Il prend en entré x et $\hat{x}$, respectivement les données réelles et générées, et produit un score $\hat{y}$ relatif au "réalisme" de son entrée (i.e. pouvant appartenir ou non à l'espace des données réelles). Il est ensuite entraîné à réduire son erreur de prédiction i.e. minimiser le W-loss. Le générateur produit $\hat{x}$ à partir d'une entrée aléatoire z. $\hat{x}$ est traitée par le critique et les rewards (R_i, avec $i \in [1,n]$). Comme illustré sur la figure 6.2, les paramètres du générateur sont ensuite entraînés à partir de la sorties de ces modules, i.e. $\hat{y}$ venant du critique, et s_i venant des rewards R_i. Cet apprentissage se fait via la rétro-propagation du gradient des pertes respectives, en considérant les blocs rewards et le critique en mode inférence. Durant son apprentissage, le générateur cherche ainsi à minimiser le W-loss provenant du critique, ainsi que les pertes provenant des rewards. Ces dernières sont obtenues par une mesure de l'erreur quadratique moyenne par rapport aux objectifs correspondants. Les rewards sont des modèles ayant suivi un apprentissage supervisé avant d'être inclus dans l'apprentissage du générateur. Ils sont entraînés sur des datasets labellisés à l'aide d'un simulateur de NoC, nous permettant d'associer des mesures de performances à des configurations de NoC. Une fois inclus dans le M-RWGAN, les rewards ne sont plus entraînés, et sont uniquement utilisés en mode inférence.

Fonction de perte multi-objectif Nous illustrons maintenant le fonctionnement de notre implémentation multi-objectif avec la fonction de perte du générateur. En effet, c'est à ce niveau là que l'aspect multi-objectif est réellement considéré. Comme pour tout réseau de neurones, durant son entraînement le générateur cherche à minimiser son erreur (i.e. perte). Le principe de la fonction de perte du générateur d'un RWGAN est rappelé ici :

$$L_G(z) = (1-\lambda)L_C(\hat{x}) + \lambda[\beta L_R(\hat{x})] \tag{6.1}$$

, où L_G, L_C et L_R sont respectivement les fonctions de pertes du générateur, critique et reward. *lambda* est le coefficient de répartition entre l'erreur du critique et du reward considérée dans l'entraînement du générateur. β est un coefficient permettant d'équilibrer la différence de grandeur qui peut apparaître entre les valeurs de perte du critique et du reward. Étant donnée notre architecture de M-RWGAN, la nouvelle fonction de perte du générateur est définie

comme suit :

$$L_G(z) = (1-\lambda)L_C(\hat{x}) + \lambda[\sum_{i=1}^{n} \beta_i L_{R_i}(\hat{x})] \tag{6.2}$$

, où $\sum_{i=1}^{n} \beta_i = \beta$, β vient de Eq. 5.1 et L_{R_i} la fonction de perte du $i^{\text{ème}}$ reward. *Note :* avec n qui augmente, il sera important d'ajuster avec précaution la valeur de β pour éviter une détérioration des valeurs de pertes.

Ainsi, notre architecture fournit une solution pour entraîner un réseau de neurones génératif selon plusieurs objectifs. Il est de plus possible d'ajuster comme souhaité l'impact de chacun des objectifs à l'aide de coefficients (i.e. β_i), permettant de régler la fonction multi-objectif global avec précision.

6.3 Résultats et analyses

Nous présentons ici différents résultats prometteurs obtenus avec notre outil. Ces résultats constituent une preuve de concept venant compléter les travaux présentés dans le chapitre 5.

6.3.1 Conditions expérimentales

Espace de conception Nous limitons nos expérimentation à la génération de matrices de paramètres X, correspondant aux types des routeurs. Les topologies des NoC sont identiques, et nous considérons des mesh 8x8.

La liste des classes de routeurs est présentée dans le tableau 6.1. Nous avons 3 classes de routeurs, qui se différencient par la taille des buffers. Ce sont des routeurs homogènes, i.e. tous les buffers ont la même taille, et de structure classique (5 ports : Nord, Sud, Est, Ouest, Local).

Nom	Big	Medium	Small
Taille des buffers	12	4	2

TABLE 6.1 Taille des buffers en *flits*.

Simulateur Afin d'obtenir, pour différents trafics, les mesures de performances et de consommation énergétique des NoC produits, nous utilisons *Omnet++* [164], un simulateur haut-niveau à évènements discrets pour les réseaux de communication. Sur ce simulateur, nous exploitons le framework HNOCS [19], qui fournit les modèles de base pour simuler des NoC tels que les canaux virtuels et le routeur conventionnel avec pipeline de 3 étages. Nous complétons ce framework avec l'ajout de trafics synthétiques, e.g. hotspot [49]. Nous ajoutons

à framework l'utilisation de la bibliothèque *Orion3* [83] pour obtenir des estimations de la consommation en puissance statique et dynamique. À partir des paramètres matériels de la technologie ciblée, et du taux de basculement simulé, cette bibliothèque permet d'estimer la consommation d'un système avec une erreur faible : inférieur à 10% comparé à des modèles RTL plus bas niveaux. Le tableau 6.2 présente les principaux paramètres de technologie utilisés dans notre étude.

Manufacturing	**Vdd**	**Frequency**	**Crossbar type**
$45nm$	$1.0V$	$650MHz$	Matrix

TABLE 6.2 Orion3.0 : paramètres de la technologie

Architecture et paramètres du M-RWGAN Notre M-RWGAN est construit avec trois blocs de rewards : un reward pour le seuil de saturation, un second reward pour la consommation énergétique au seuil de saturation et un dernier pour la surface du NoC. Le but de ces rewards est que, en jouant sur leur importance dans l'entraînement du générateur, il soit possible modifier les spécificités des NoC générés avec soit une tendance sur l'économie d'énergie, au détriment de la bande-passante (i.e. seuil de saturation bas), soit une préférence pour un seuil de saturation élevé mais pour une consommation énergétique plus élevée. Enfin, il sera intéressant de trouver une configuration qui optimise les deux objectifs, pour tendre vers une efficacité énergétique optimale – i.e. seuil de saturation élevé pour une consommation énergétique minimale.

À l'instar du RWGAN dans le chapitre précédent, les dimensions des réseaux ont été déterminées empiriquement, et ne découlent donc pas d'une méthodologie particulière d'optimisation. Les détails des différents modules du M-RWGAN sont donnés dans les tableaux 6.3 et 6.4.

Modules :	Générateur				Critique				
Couches :	Entrée	L1	L2	Sortie	Entrée A Entrée X	L1	L2	L3	Sortie
Type	Input	Dense			Input x2	GCN			Dense
Dimensions	100	768	1536	192	A : 64x64 X : 64x3	16	32	64	1
Fonction d'activation	–	*LeakyReLU* ($\alpha = 0.2$)			–	*LeakyReLU* ($\alpha = 0.2$)			*Linear*

TABLE 6.3 Dimensionnement des modules Générateur et Critique du M-RWGAN

Le générateur est un réseau MLP de 2 couches cachées (voire tableau 6.3). Il prend en entrée un vecteur aléatoire de 100 unités. Sa sortie est redimensionnée en une matrice 64x3 pour correspondre aux dimensions de la matrice X à générer. Enfin, pour forcer l'encodage *one-hot*, la fonction d'activation *GumbelSoftmax* est appliquée à la sortie du générateur [38].

Le critique est un réseau de neurones de type GCN convolutif [89]. Ses couches de type GCN sont implémentées via la librairie *Spektral* [68] basée sur Keras. Le critique est ainsi composé de 3 couches de type GCN, et d'une couche dense en sortie. Le critique possède deux entrées, correspondant à la matrice d'adjacence A et la matrice X des graphes évalués.

Les rewards implémentés peuvent être de deux types : GCN convolutif ou CNN, dont les détails sont donnés dans le tableau 6.4. Comme pour le critique, les rewards GCN permettent de traiter les matrices X en deux dimensions : 64x3, pour le nombre de routeurs (i.e. 64) et le nombre de classes de routeurs (i.e. 3 classes possibles). Ils nécessitent également la matrice d'adjacence A en entrée pour réaliser la convolution. Ils sont construits avec 4 couches GCN, et une couche dense en sortie avec *Sigmoïde* comme fonction d'activation (i.e. pour garantir un score $\in$ [0,1]). Les rewards de type CNN, permettant de traiter les matrices X en trois dimensions : 8x8x3, pour les dimensions du mesh 8x8 et la dimension de la classe (i.e. 3, pour les 3 catégories possibles). Ils ne considèrent pas la matrice A. Nous l'utiliserons par exemple pour approximer la fonction de surface, qui prédit la surface d'un NoC en fonction de ses routeurs, et qui ne nécessite donc pas de connaissances de la topologie. Les rewards CNN sont implémentés avec 2 couches convolutives, suivies d'une couche dense. La couche de sortie est une couche dense avec *Sigmoïde* comme fonction d'activation.

L'entraînement de chaque reward est réalisé en amont de l'entraînement du M-RWGAN, afin de les utiliser uniquement en inférence durant l'entraînement du M-RWGAN. Leurs apprentissages utilisent l'optimiseur *Adam* et convergent rapidement (environ 20 époques) vers des erreurs minimales, inférieures à 1% sur des données de test, ce qui assure une bonne évaluation des sorties du générateur. L'entraînement global du M-RWGAN est similaire au RWGAN présenté dans le chapitre 5.

Modules :	Reward GCN						Reward CNN				
Couches :	Entrée A Entrée X	L1	L2	L3	L4	Sortie	Entrée	L1	L2	L3	Sortie
Type	Input x2	GCN				Dense	Input	CNN		Dense	
Dimensions	A : 64x64 X : 64x3	128	64	64	32	1	X : 8x8x3	128 filtre : 3x3 pas : 1	32 filtre : 3x3 pas : 2	128	1
Fonction d'activation	–	*LeakyReLU* ($\alpha = 0.2$)				*Sigmoïde*	–	*LeakyReLU* ($\alpha = 0.2$)			*Sigmoïde*

TABLE 6.4 Dimensionnement des différents Rewards du M-RWGAN

6.3.2 Datasets : pré-analyse

Nous étudions l'apprentissage de notre modèle pour deux trafics différents : uniforme et hotspot30, circulant sur un NoC de type mesh 8x8. La charge de ces trafics est représentée sur la figure 6.3. Ainsi, l'objectif de l'entraînement du M-RWGAN sera de générer des matrices X de NoC optimisés pour ces trafics, selon différents critères d'optimisation. Nos critères d'optimisation seront le seuil de saturation, la puissance consommée au seuil de saturation et la surface. Ainsi, en adaptant l'importance de ces critères d'optimisation, il est possible de faire varier les conditions d'apprentissage et les caractéristiques finales des NoC générés.

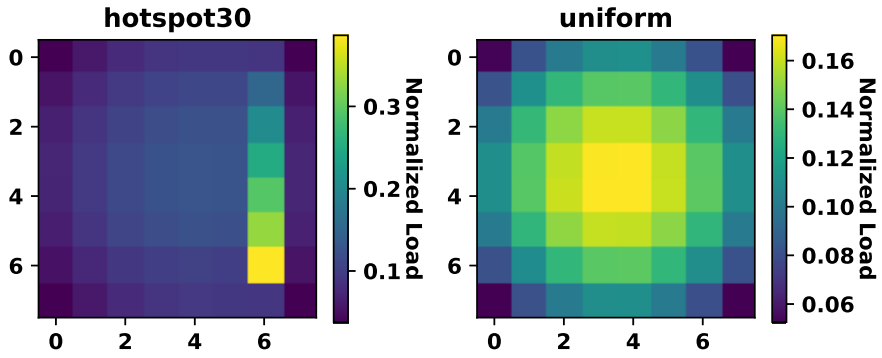


FIGURE 6.3 Quantité de trafic normalisée reçue par routeur pour 2 trafics synthétiques exécuté sur un mesh 8x8 : hotspot30 (i.e. 30% du trafic est à destination du routeur 54), et uniforme.

La base de données, ou dataset, joue un rôle majeur dans l'apprentissage machine. En effet, c'est à partir de ce dataset que l'IA va s'entraîner pour construire son modèle final. L'analyse du dataset permet donc de vérifier d'une part que ce dernier ne dispose pas de biais particulier (e.g. déséquilibre dans les proportions des labels pour une classification), et d'autre part d'anticiper les motifs pouvant être appris par l'IA. Dans notre cas, le dataset doit permettre à l'IA d'apprendre à attribuer une taille de routeur en fonction de sa position, du trafic et des paramètres d'optimisation. Nous proposons donc ici d'analyser les datasets d'entraînement des M-RWGAN afin de déterminer les potentiels biais d'apprentissage présents.

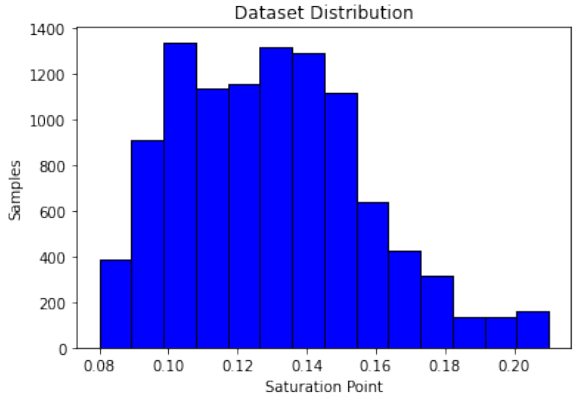
Nos datasets contiennent uniquement des matrices X (i.e. type des routeurs) pour des mesh 8x8. Afin de limiter un premier biais dans la construction du dataset, ces matrices sont générées de façon aléatoire, assurant à la fois une homogénéité sur les types de chaque routeur, mais aussi sur la répartition des types de routeur. Ainsi, nous assurons qu'il n'y ait pas de déséquilibre dans les datasets, afin d'éviter de se trouver dans un cas d'apprentissage déséquilibré [91]. Ces datasets contiennent, pour chaque trafic, 10k matrices X, associées aux données de simulation du NoC correspondant (i.e. latences, puissance, surface, etc.). Ces

données de simulation ne sont utilisées que pour l'entraînement des Rewards. Les modules du GAN (générateur et critique) ne s'entraînent que sur les matrices X.

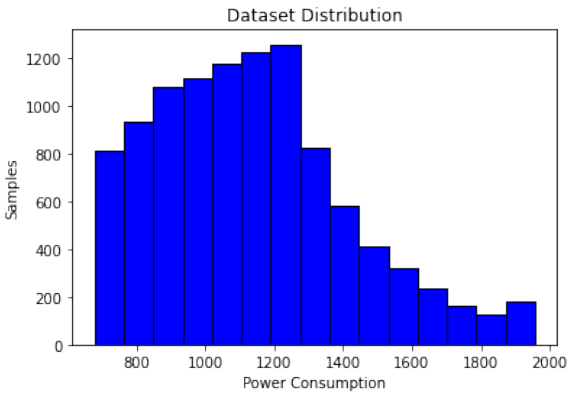
Nous analysons donc nos datasets pour le trafic uniforme et le trafic hotspot30, en nous intéressant plus particulièrement aux données de seuil de saturation et de puissance, et leur corrélation avec les types des routeurs (i.e. taille). La donnée de surface n'est pas étudiée ici. En effet, les types de routeurs sont directement liés à la taille des routeurs (i.e. taille des buffers). Or, ayant garanti l'homogénéité de la répartition des classes, il en va de même pour la répartition des surfaces.

6.3.2.1 Trafic Uniforme

Regardons tout d'abord les biais existant dans notre dataset pour le trafic uniforme. On définit un biais comme une non-uniformité du dataset, en fonction d'une donnée particulière. La distribution du dataset est présentée figure 6.4, en fonction du seuil de saturation et de la puissance consommée, respectivement les figures 6.4a et 6.4b.



(a) Distribution des NoC du dataset selon leur seuil de saturation.

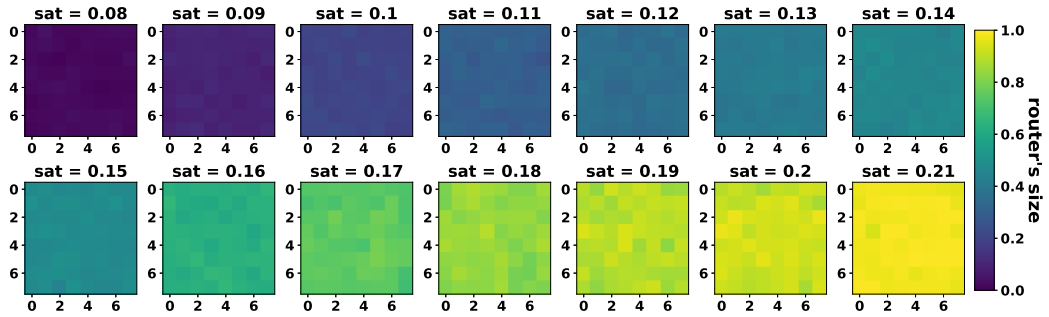


(b) Distribution des NoC du dataset selon leur puissance (mW) au seuil de saturation.

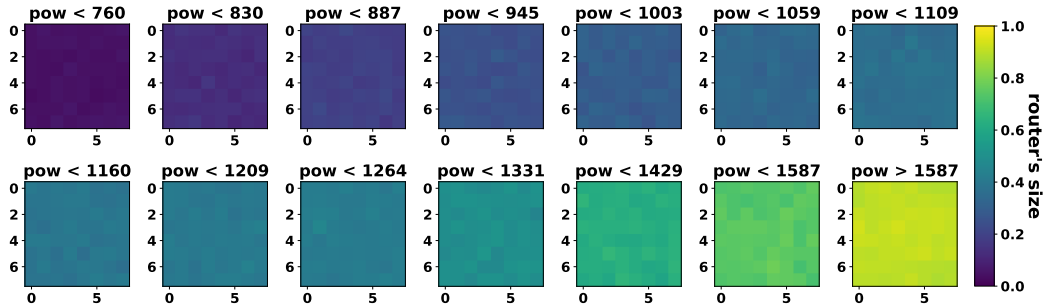
FIGURE 6.4 Distribution du dataset uniforme.

On constate un biais important autour du seuil de saturation de 13% et de la puissance de 1200mW. On peut donc attendre de l'entraînement du GAN, sans les rewards, de tendre vers la génération de NoC ayant un seuil de saturation moyen et une puissance consommée basse. On relève par ailleurs pour ce dataset un seuil de saturation moyen à 12.88% et une puissance moyenne au seuil de saturation de 1136mW. De plus, les valeurs extrêmes de seuil de saturation sont peu présentes, indiquant qu'il y a peu de configurations supportant un taux d'injection supérieur à 18%. On remarque enfin que les deux distributions sont similaires, indiquant une corrélation entre le seuil de saturation et la puissance consommée.

Cela correspond aux résultats attendus pour un NoC soumis à un trafic uniforme. En effet, afin d'améliorer le seuil de saturation d'un NoC soumis au trafic uniforme, il faut entre autres augmenter la bande passante des routeurs et donc leur taille, ce qui augmente la consommation globale du NoC. Cette analyse se confirme logiquement lorsqu'on regarde en détails la taille moyenne des routeurs des NoC, en fonction de la puissance consommée et du seuil de saturation, figure 6.5. En effet, on peut voire que plus le seuil de saturation des NoC est élevé, plus la taille des routeurs est grande. Il en va de même pour la puissance. On remarque cependant que cette simple analyse ne permet pas d'extraire le motif du trafic, qui doit logiquement être corrélé avec la taille des routeurs pour optimiser l'efficacité énergétique. Nous verrons par la suite comment l'apprentissage du GAN se comporte.



(a) Seuil de saturation.

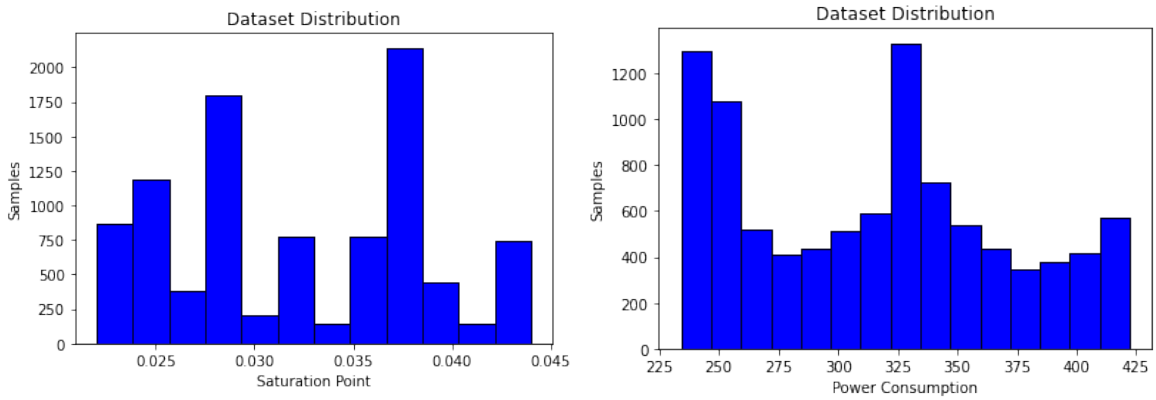


(b) Puissance (mW) au seuil de saturation.

FIGURE 6.5 Taille moyenne des routeurs, en fonction du seuil de saturation 6.5a et de la puissance consommée 6.5b des NoC, pour le trafic **uniforme**.

6.3.2.2 Trafic Hotspot30

Regardons maintenant les biais existant dans notre dataset pour le trafic hotspot30. La distribution du dataset est présentée figure 6.6, en fonction du seuil de saturation et de la puissance consommée, respectivement les figures 6.6a et 6.6b.



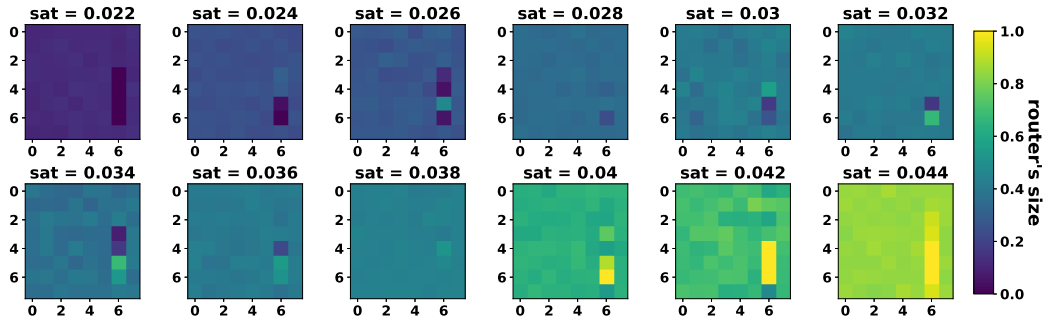
(a) Distribution des NoC du dataset selon leur seuil de saturation.

(b) Distribution des NoC du dataset selon leur puissance (mW) au seuil de saturation.

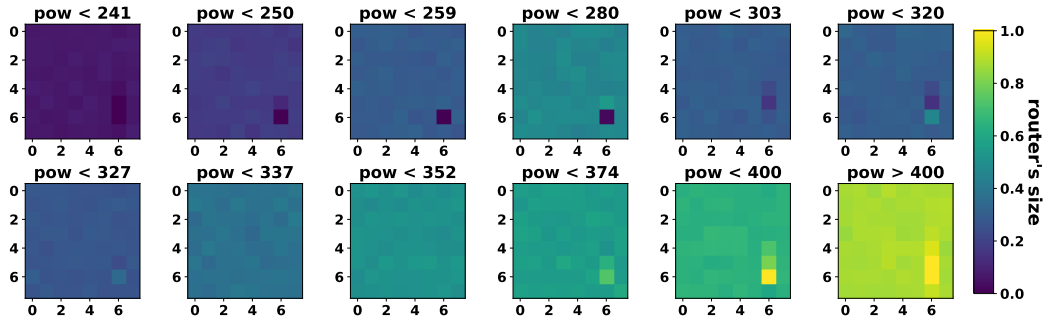
FIGURE 6.6 Distribution du dataset hotspot30.

On remarque la présence de biais sur certaines valeurs. En particulier, on note une densité plus élevée de NoC ayant une puissance autour de 325mW et 250mW. De même, deux valeurs de seuil de saturation ressortent avec 2.75% et 3.75%. Ces points particuliers peuvent être expliqués par le fait que le trafic hotspot va impacter un nombre limité de routeurs. Ainsi, de fortes différences de valeurs peuvent apparaître entre des configurations similaires mais dont les quelques différences se trouvent au niveau du hotspot. Il sera intéressant de voir comment l'apprentissage du RWGAN sera impacté par ces biais. On note pour ce dataset un seuil de saturation moyen à 3.22% et une puissance moyenne au seuil de saturation de 315mW.

De la même façon que pour le trafic uniforme, nous observons figure 6.7 la taille moyenne des routeurs des NoC en fonction du seuil de saturation et de la puissance consommée. Cette fois-ci, nous remarquons très clairement la corrélation entre le motif du trafic et la taille des routeurs. Bien que, de manière intuitive, la tendance reste la même que pour le trafic uniforme (le seuil de saturation et la consommation augmentent lorsque la taille des routeurs augmente), on distingue clairement la localisation du point chaud, confirmant son impact sur les performances du NoC. Ainsi, on peut confirmer que les performances d'un NoC soumis à un trafic hotspot sont principalement impactées par l'architecture des routeurs situés aux abords du point chaud (ici verticalement, de par le routage XY). Plus globalement, cela confirme l'intérêt à ce que l'architecture des routeurs soit corrélée avec la charge du trafic.



(a) Seuil de saturation.



(b) Puissance (mW) au seuil de saturation.

FIGURE 6.7 Taille moyenne des routeurs, en fonction du seuil de saturation 6.7a et de la puissance consommée 6.7b des NoC, pour le trafic **hotspot30**.

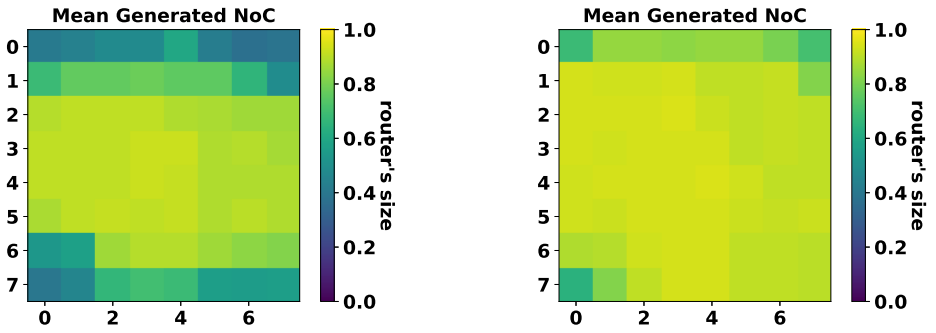
6.3.3 Expérimentations

Sur la figure 6.3 est représentée la répartition de la charge du trafic circulant sur un mesh 8x8, pour deux trafics synthétiques. L'objectif ici est d'entraîner le M-RWGAN à générer des configurations de NoC hétérogènes selon différents critères d'optimisation, et pour les trafics présentés figure 6.3. Nous rappelons que le générateur produit des matrices X décrivant la classe de chacun des routeurs, pour un mesh 8x8, et pour 3 classes possibles décrites dans le tableau 6.1.

Pour chaque apprentissage, nous fixons la proportion des rewards impactant l'entraînement du générateur. Ainsi, dans la suite de ce manuscrit, nous utiliserons la notation <reward><proportion> pour décrire les poids des blocs reward, avec <reward> égal à *Sat*, *Pow*, *Area* respectivement pour le reward du seuil de saturation, le reward de la puissance au seuil de saturation et le reward de la surface. La valeur <proportion> sera comprise en 0 et 100, décrivant ainsi la proportion du rewards dans la fonction de loss finale (i.e. équivalent à β_i dans l'équation 6.2). Par exemple, un apprentissage prenant en considération à 90% le seuil de saturation et à 10% la puissance consommée verra son reward désigné par la notation "Sat90Pow10" (sous-entendu Area0).

6.3.3.1 Paramètres et Illustration des apprentissages

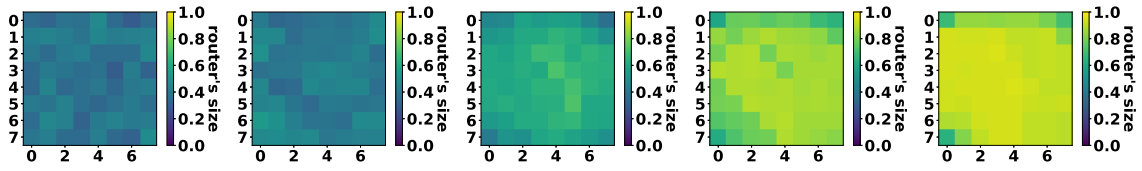
Les entraînements se font sur 300 époques et λ (voire équation 6.2) décroît progressivement de 1 vers 0.2 à partir de l'époque 50. Ces valeurs ont été déterminées de manière empirique, pour garantir une transition "douce" entre l'apprentissage du générateur lié uniquement au critique ($\lambda = 1$), et l'apprentissage du générateur principalement guidé par le reward global ($\lambda = 0.2$).



(a) $\beta = 1$ (b) $\beta = 10$

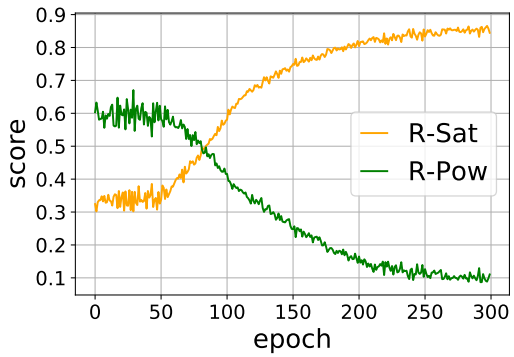
FIGURE 6.8 Taille moyenne des routeurs, en fonction de la valeur de β, pour un entraînement de 300 époques sur le trafic uniform, avec le reward saturation uniquement (Sat100).

La variable β est fixée à 10 afin de limiter l'impact du biais lié à la moyenne du dataset. La figure 6.8 illustre cet intérêt. Sur cet figure sont comparées la taille moyenne des routeurs des NoC générés, après deux entraînements de M-RWGAN considérant uniquement le reward de seuil de saturation, pour $\beta = 1$ et $\beta = 10$. Ainsi, les résultats attendus sont que le générateur produise des NoC ayant des routeurs de grandes tailles afin de maximiser le seuil de saturation (i.e. maximiser le reward). On constate qu'avec $\beta = 1$, les routeurs extérieurs sont réduits, ce qui ne correspond pas à l'analyse du dataset et à l'impact du biais. Or, avec $\beta = 10$ (i.e. le loss du reward est plus important que le loss du critique), on constate logiquement un impact plus important du reward, permettant de pallier au biais du critique.

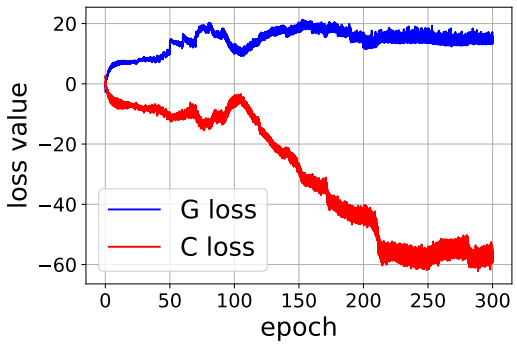


(a) époque 0 (b) époque 50 (c) époque 100 (d) époque 200 (e) époque 300

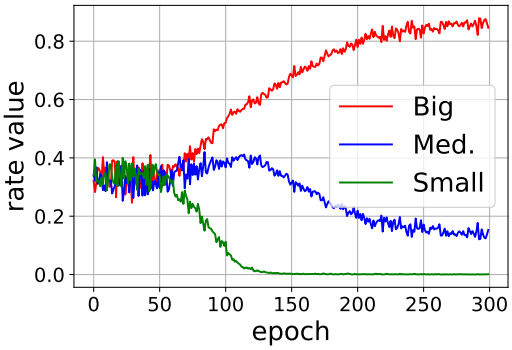
FIGURE 6.9 Taille moyenne des routeurs, en fonction de l'étape d'entraînement (époque), pour un entraînement de 300 époques sur le trafic uniforme, avec le reward de saturation uniquement (Sat100).



(a) Sortie des rewards Sat et Pow.



(b) Fonction de perte du générateur et du critique.



(c) Taux de présence des types de routeurs.

FIGURE 6.10 Évolution des différentes valeurs associées à l'entraînement du M-RWGAN, pour un entraînement de 300 époques sur le trafic uniforme, avec le reward saturation uniquement (Sat100).

L'historique de l'entraînement est proposé sur les figures 6.9 et 6.10. Sur la figure 6.9 est détaillée l'évolution des NoC générés par notre GAN. On constate une augmentation progressive de la taille des routeurs, à partir de l'époque 50, soit à partir du moment où le reward est pris en compte dans l'entraînement du générateur. Cette influence du reward est confirmée sur la figure 6.10a qui décrit l'évolution de la sortie des rewards de saturation et de consommation énergétique. Ces sorties correspondent à une évaluation des NoC générés, selon leur performances en termes de seuil de saturation et de puissance consommée. Ainsi, on constate qu'à partir de l'époque 50 le score associé au reward de saturation augmente progressivement pour tendre vers 0.9 (le score maximal étant de 1). Cela correspond bien aux résultats attendus. En contrepartie, le score en termes de puissance décroît pour tendre vers 0.1, indiquant que les NoC générés consomment davantage.

L'évolution du loss du générateur et du critique en fonction de l'avancement de l'entraînement est présenté figure 6.10b. On peut tout d'abord constater que les deux loss sont symétriques jusqu'à l'époque 50 (début de la prise en compte du reward), ce qui indique que

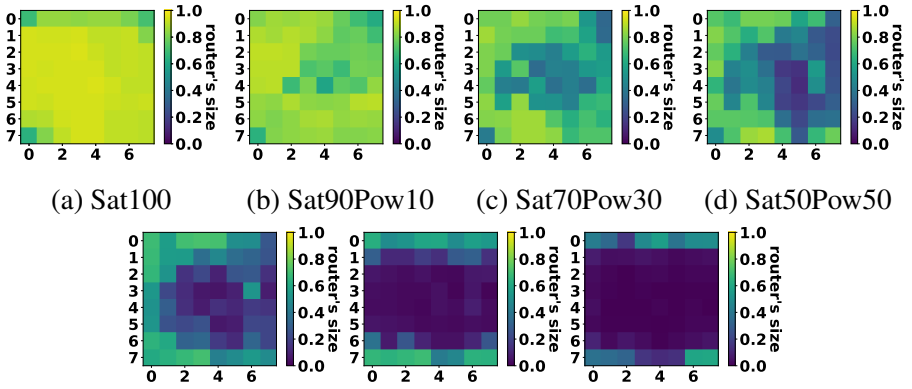
l'apprentissage est stable. Ensuite, on constate une bonne adaptation du générateur au reward puisque le loss du générateur décroît seulement 50 époques après son entrée. Enfin, les deux loss se stabilisent à nouveau en fin d'entraînement (de l'époque 200 à la fin), avec un facteur 3 de différence, directement lié à β qui favorise la prise en compte du reward.

Enfin, l'impact de cet apprentissage sur le taux de présence des types de routeurs est détaillé figure 6.10c. On y constate logiquement une tendance vers l'implémentation de routeurs de grandes tailles (i.e. type Big), et une présence quasi nulle des routeurs de plus petite taille. Ainsi, les routeurs de type Small sont inexistants dans les réseaux produits en fin d'entraînement, et les Medium sont en large infériorité (12.4%) face aux Big (87.6%).

Conclusion : Nous pouvons donc confirmer le bon fonctionnement de notre set-up. Nous analyserons par la suite plus en détails la façon dont les rewards impactent l'apprentissage selon leur poids, et comment notre outil permet de produire efficacement des NoC optimisés.

6.3.3.2 Résultats pour le trafic uniforme

Nous commençons par analyser les résultats du M-RWGAN pour le trafic uniforme. Dans le but de produire des NoC optimisés du point de vue de l'efficacité énergétique, nous analysons tout d'abord les réseaux générés par le générateur lorsque ce dernier est entraîné avec les rewards Sat et Pow en différentes proportions. En effet, l'efficacité énergétique étant un ratio entre la performance et la puissance, la première approche naïve est de mettre en concurrence les rewards de seuil de saturation et de puissance consommée afin de guider l'apprentissage du générateur vers la production de NoC optimisant ces deux caractéristiques (i.e. maximiser le seuil de saturation et minimiser la puissance consommée).



(a) Sat100 (b) Sat90Pow10 (c) Sat70Pow30 (d) Sat50Pow50

(e) Sat30Pow70 (f) Sat10Pow90 (g) Pow100

FIGURE 6.11 Taille moyenne des routeurs, en fonction de la proportion entre les rewards Sat et Pow, pour un entraînement de 300 époques sur le trafic uniforme.

Valeurs		R-Sat	R-Pow	t-Big	t-Medium	t-Small
Rewards	Sat100	0.86	0.09	87.6%	12.4%	0%
	Sat90Pow10	0.81	0.22	76.5%	23.5%	0%
	Sat70Pow30	0.67	0.38	55.2%	42.8%	2%
	Sat50Pow50	0.52	0.54	40%	49.9%	10.1%
	Sat30Pow70	0.35	0.70	26.2%	43.7%	30.1%
	Sat10Pow90	0.18	0.86	15.2%	33.3%	51.5
	Pow100	0.08	0.95	5.3%	23.7%	71%

TABLE 6.5 Valeurs des différentes variables d'entraînement. R-Sat et R-Pow correspondent respectivement aux scores du reward de seuil de saturation et du reward de consommation. t-Big, t-Medium et t-Small sont les taux de présence (i.e. répartition) des types de routeur respectifs dans les NoC générés. Trafic uniforme.

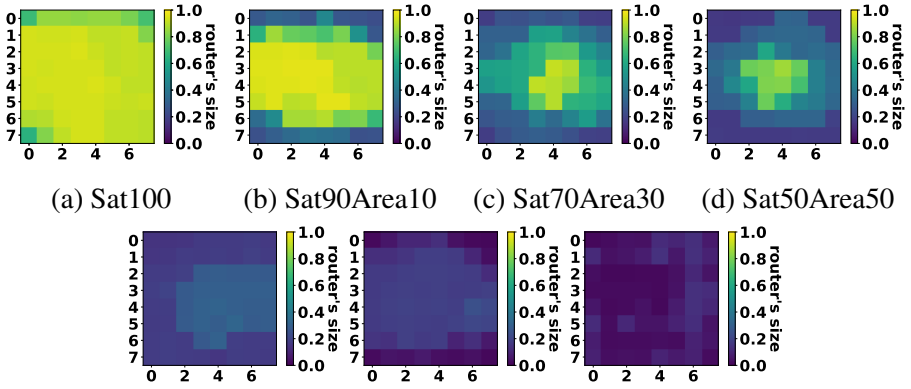
Sur la figure 6.11 sont affichées les tailles moyennes des routeurs généré par le générateur, pour différents entraînements correspondant à différentes proportions des rewards Sat et Pow. Ainsi, de la figure 6.11(a) à la figure 6.11(g), la proportion du reward de la puissance est augmentée, tout en diminuant la proportion du reward de performance (seuil de saturation). Les valeurs détaillées des sorties de reward (i.e. scores) et des taux de présence des types de routeurs dans les NoC générés sont disponibles dans le tableau 6.5. Nous observons comme attendue une réduction globale de la taille des routeurs, à mesure que la proportion du reward Pow est augmentée. En effet, la proportion de routeurs de type Big décroît de 87.6% à 5.3%, alors que la proportion des routeurs de type Small augmente de 0% à 71%. Nous observons ensuite une réduction plus importante de la taille des routeurs centraux. Ce résultat s'explique par le fait que la puissance dynamique du routeur est dominante dans sa puissance totale, lorsque soumis à un trafic important. Or, le reward Pow est entraîné pour attribuer un score correspondant à la consommation au seuil de saturation du NoC évalué. Ainsi, plus les routeurs soumis à un trafic élevé seront grands, plus ces derniers consommeront et plus le score du NoC sera bas. Cela explique le fait que, dès que le reward Pow est considéré, les premiers routeurs réduits sont les routeurs centraux (i.e. charge de trafic plus élevée, voir figure 6.3).

Ce résultat est en contradiction avec le résultat attendu pour obtenir un NoC efficace énergétiquement. En effet, afin d'optimiser les performance et la consommation énergétique du NoC, l'objectif est que la taille des routeurs suive une répartition similaire au trafic – i.e. plus la charge du trafic est élevée sur un routeur, plus ce routeur doit être grand pour supporter la charge. Pour se rapprocher de ce comportement, ce n'est pas la puissance au seuil de

saturation qui doit être considérée (car largement dominée par la puissance dynamique), mais la puissance statique, soit la puissance pour un trafic nul. Cette dernière est corrélée avec notre troisième caractéristique considérée : la surface du NoC.

Ainsi nous proposons dans un second temps d'étudier les résultats d'apprentissage du générateur, pour un reward composé du reward Sat et du reward Area.

Ces résultats sont présentés sur la figure 6.12, et le détail des valeurs des variables d'entraînement (scores en sortie des rewards et taux de présence des types de routeur) est disponible dans la table 6.6.



(a) Sat100 (b) Sat90Area10 (c) Sat70Area30 (d) Sat50Area50

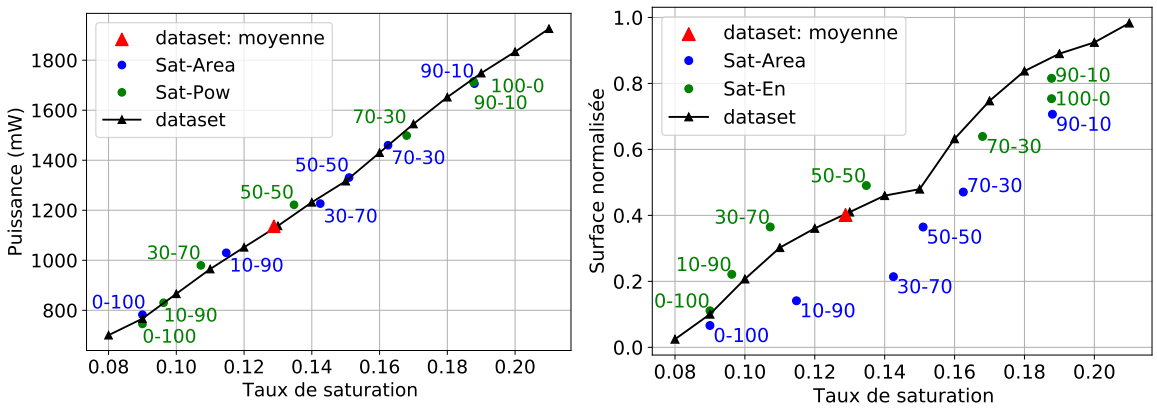
(e) Sat30Area70 (f) Sat10Area90 (g) Area100

FIGURE 6.12 Taille moyenne des routeurs, en fonction de la proportion entre les rewards Sat et Ar, pour un entraînement de 300 époques sur le trafic uniforme.

Valeurs		R-Sat	R-Area	t-Big	t-Medium	t-Small
Rewards	Sat100	0.86	0.11	87.6%	12.4%	0%
	Sat90Area10	0.78	0.29	63.2%	36.8%	0%
	Sat70Area30	0.66	0.54	35%	64%	1%
	Sat50Area50	0.58	0.66	20%	78%	2%
	Sat30Area70	0.49	0.78	4.6%	88.1%	7.3%
	Sat10Area90	0.33	0.86	0.8%	66.3%	32.9%
	Area100	0.12	0.93	0%	32.8%	67.2%

TABLE 6.6 Valeurs des différentes variables d'entraînement. Reward Sat et Area, trafic uniforme.

Comme attendu, on constate une réduction globale de la taille des routeurs. Cependant, cette fois-ci ce sont bien les routeurs centraux qui conservent une taille plus grande que la



(a) Seuil de saturation et puissance.

(b) Seuil de saturation et surface.

FIGURE 6.13 Comparaison des NoC générés et du dataset selon différentes métriques (seuil de saturation, énergie consommée, surface), lorsque soumis à un trafic uniforme.

moyenne. L'utilisation du reward Area permet ainsi d'approcher de notre optimal théorique en faisant correspondre la taille des routeurs avec la charge du trafic considéré.

On remarque ainsi que le choix des rewards utilisés est primordial pour que notre générateur puisse produire des NoC compris dans un espace de conception optimisé. Il est aussi important de noter que l'entraînement de notre R-WGAN permet au générateur d'extraire des motifs du dataset qui n'était pas visibles à partir de notre pré-analyse. En effet, sur la figure 6.4, le motif du trafic uniforme n'est pas détectable. Or, en fonction des rewards utilisés, et notamment lorsqu'on combine un reward de performance (i.e. Sat) à un reward de consommation (i.e. Pow et Area), on reconnaît clairement ce motif dans les NoC générés.

Comparaison globale avec le dataset : Pour revenir à notre objectif de générer un ensemble de NoC optimisés, nous proposons de comparer les performances des NoC générés avec les NoC du dataset. Pour chacun des apprentissages correspondant à une combinaison de rewards, nous évaluons 100 NoC générés avec le simulateur OMNET++, et récupérons la moyenne des résultats. Sur la figure 6.13 sont positionnées les moyennes pour chacune des configurations et la moyenne du dataset, pour les valeurs de seuil de saturation, puissance consommée et surface. Ainsi, deux graphiques sont proposés : figure 6.13a pour le positionnement en fonction du seuil de saturation et de la puissance consommée, et figure 6.13b pour le seuil de saturation et la surface des NoC. En noire sont représentées les données du dataset de départ, partitionnées selon leur seuil de saturation. Ces dernières permette de tracer une frontière où tout point "en-dessous" de cette frontière (i.e. seuil de saturation plus élevé et/ou puissance/surface inférieure) peut être considéré comme un NoC ayant une meilleure efficacité énergétique.

Nous remarquons tout d'abord de très légères variations sur le graphique 6.13a en comparaison avec le dataset. Certaines configurations sont effectivement meilleures que le dataset, mais il est difficile d'extraire la meilleure configuration. Sur le graphique 6.13b positionnant les différents résultats selon la surface moyenne et le seuil de saturation moyen, on note logiquement que l'ensemble des configurations issues d'un entraînement possédant le reward Area sont en-dessous de la frontière et possèdent donc une meilleure efficacité énergétique que le dataset.

Gains en efficacité énergétique : Nous proposons d'extraire la meilleure configuration pour chaque type de combinaison (Sat-Pow et Sat-Area). Pour cela nous procédons au calcul suivant : pour chaque configuration, un ratio représentant l'efficacité énergétique au seuil de saturation est calculé, soit $\frac{\text{Taux de Saturation}}{Puissance}$. Les configurations ayant le meilleur score sont les suivantes : Sat0Pow100 pour le type de reward Sat-Pow, et Sat30Area70 pour le type Sat-Area.

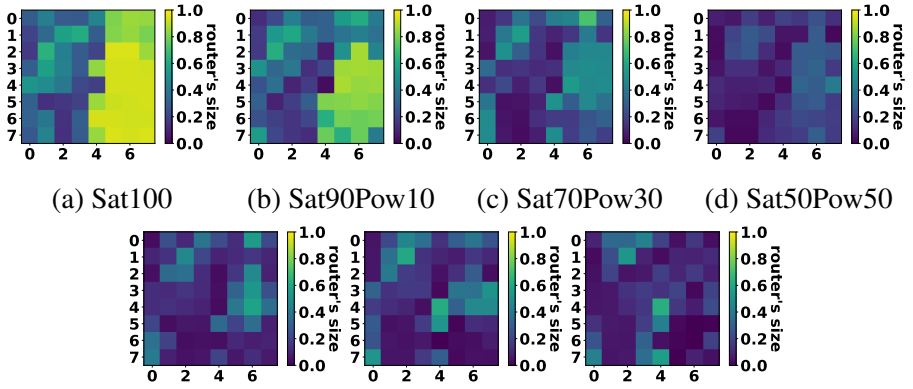
— *Sat0Pow100 :* Les NoC produits par le générateur après un entraînement avec le reward Sat0Pow100 ont en moyenne un seuil de saturation inférieur de 3.88% à la moyenne du dataset (seuil de saturation à 12.88% pour la moyenne du dataset contre 9% pour les NoC générés) soit une diminution de 30.1%. Pour ce qui est de la puissance consommée au seuil de saturation, cette dernière est de 746mW contre 1136mW pour la moyenne du dataset, ce qui représente une réduction de 34.3%. Enfin, si on mesure l'efficacité énergétique comme la performance (i.e. seuil de saturation) divisée par la puissance consommée, on obtient une amélioration d'environ 6.45% de l'efficacité énergétique des NoC générés, en comparaison avec la moyenne du dataset de départ.

— *Sat30Area70 :* Les NoC produits par le générateur après un entraînement avec le reward Sat30Area70 ont en moyenne un seuil de saturation 14.25%, similaire à la moyenne du dataset. Pour ce qui est de la puissance au seuil de saturation, cette dernière est de 1000mW contre 1136mW pour la moyenne du dataset, ce qui représente une diminution de 12%. Enfin, on obtient une amélioration d'environ 14.6% de l'efficacité énergétique des NoC générés, en comparaison avec la moyenne du dataset de départ.

6.3.3.3 Résultats pour le trafic hotspot30

Les mêmes expérimentations sont menées sur le dataset hotspot30. Les résultats obtenus pour les combinaisons de reward Sat et Pow sont présentés figure 6.14. Des conclusions similaires aux expérimentations sur le trafic uniforme peuvent être tirées. En effet, une nouvelle fois ce sont les routeurs soumis à la plus grande charge de trafic qui sont minimisés

en priorité. L'apprentissage lié aux rewards est validé aux vues des scores présenté table 6.7. En effet, le score en sortie du reward Sat décroît de 0.86 à 0.12 avec la proportion de reward Sat qui décroît de 100% vers 0%. À l'inverse, le reward de la puissance augmente de 0.16 à 0.92 avec l'augmentation de la proportion du reward Pow de 0% à 100%.



(a) Sat100 (b) Sat90Pow10 (c) Sat70Pow30 (d) Sat50Pow50

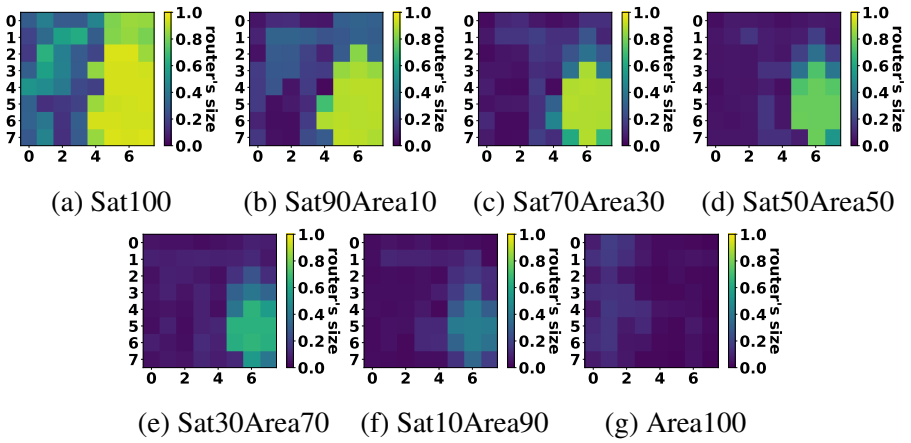
(e) Sat30Pow70 (f) Sat10Pow90 (g) Pow100

FIGURE 6.14 Taille moyenne des routeurs, en fonction de la proportion entre les rewards Sat et Pow, pour un entraînement de 300 époques sur le trafic hotspot30.

Valeurs		R-Sat	R-Pow	t-Big	t-Medium	t-Small
Rewards	Sat100	0.9	0.16	52%	34.9%	13.1%
	Sat90Pow10	0.83	0.29	35%	41.5%	23.5%
	Sat70Pow30	0.71	0.50	16%	54%	30%
	Sat50Pow50	0.64	0.58	5.8%	51%	43.2%
	Sat30Pow70	0.45	0.75	11%	47%	42%
	Sat10Pow90	0.28	0.9	12.5%	40%	47.5%
	Pow100	0.13	0.92	8.5%	34.1%	57.4%

TABLE 6.7 Valeurs des différentes variables d'entraînement. Reward Sat et Pow, trafic hotspot30.

Enfin, nous entraînons notre M-RWGAN avec des combinaisons du reward Sat et du reward Area. Comme pour le trafic uniforme, on voit sur la figure 6.15 que le générateur produit des NoC dont la taille des routeurs est directement corrélée à la charge du trafic. En effet, même pour la combinaison Sat30Area70 dont le score donné par le reward Area est de 0.85 (voire table 6.8), les routeurs autour du hotspot sont les seuls à ne pas être totalement minimisés. Ainsi, la réduction de la surface du NoC est optimisée afin de ne pénaliser au minimum les performances du NoC.



(a) Sat100 (b) Sat90Area10 (c) Sat70Area30 (d) Sat50Area50

(e) Sat30Area70 (f) Sat10Area90 (g) Area100

FIGURE 6.15 Taille moyenne des routeurs, en fonction de la proportion entre les rewards Sat et Ar, pour un entraînement de 300 époques sur le trafic hotspot30.

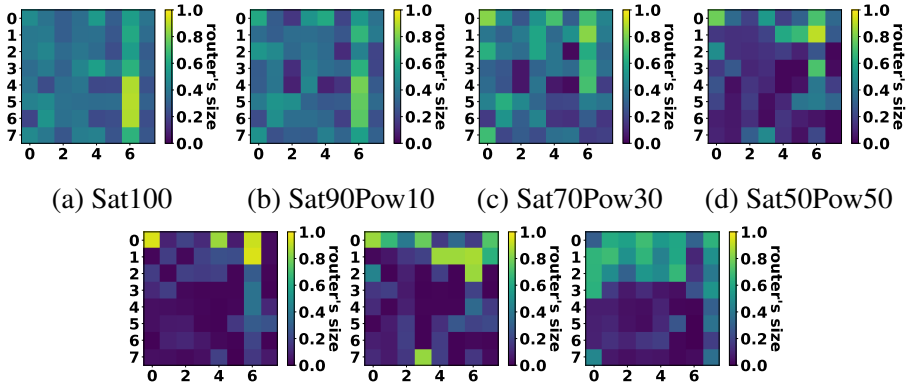
Valeurs		R-Sat	R-Area	t-Big	t-Medium	t-Small
Rewards	Sat100	0.9	0.41	52%	34.9%	13.1%
	Sat90Area10	0.84	0.63	28%	45%	27%
	Sat70Area30	0.80	0.74	18.3%	45%	36.7%
	Sat50Area50	0.76	0.8	16%	31.1%	52.9%
	Sat30Area70	0.72	0.85	10%	39%	51%
	Sat10Area90	0.66	0.9	4%	40%	56%
	Area100	0.21	0.94	0%	35%	65%

TABLE 6.8 Valeurs des différentes variables d'entraînement. Reward Sat et Area, trafic hotspot30.

On remarque cependant un manque de précision sur le trafic hotspot30. En effet, on remarque notamment sur l'apprentissage avec les rewards Sat et Area un motif de croix autour du routeur hotspot, s'éloignant ainsi du motif du trafic. Cela peut s'expliquer par l'utilisation des GCN pour construire les rewards. Ce type d'apprentissage repose sur le voisinage des nœuds du graphe. Ainsi, si l'apprentissage du reward mène à la conclusion qu'un routeur doit être de grande taille, ce même apprentissage peut se propager sur les routeurs voisins. Cela expliquerait donc que le reward de seuil de saturation attribue un meilleur score aux NoC générés avec des routeurs de grande taille voisins du réel hotspot. De futurs travaux devront éclaircir ce comportement, en étudiant plus en détails les paramètres du GCN.

Avant d'analyser les différents ensembles de NoC produits par les générateurs en fonction de la combinaison de rewards attribuée durant l'apprentissage, nous proposons d'implémenter les rewards du seuil de saturation et de la puissance avec un CNN. En effet, bien que le GCN se soit montré plus efficace pour le trafic uniforme, il semblerait que ce ne soit pas le cas pour le motif particulier du hotspot.

Supplément CNN : Nous commençons par analyser les apprentissages avec les rewards Sat et Pow à base de CNN. Les tailles des routeurs des NoC générés pour chaque entraînement sont présentées sur la figure 6.16, et les valeurs des scores et taux de présence sont détaillées dans la table 6.9. La différence majeure avec les précédentes expérimentations est l'apparition du motif du trafic hotspot, plus précisément que lors des apprentissages avec les rewards GCN. Ensuite, les conclusions sur les tailles des routeurs des NoC générés restent inchangés : pour optimiser la puissance consommée au seuil de saturation des NoC générer, les routeurs soumis aux charges de trafic les plus élevés (i.e. hotspot) sont réduits.



(a) Sat100 (b) Sat90Pow10 (c) Sat70Pow30 (d) Sat50Pow50

(e) Sat30Pow70 (f) Sat10Pow90 (g) Pow100

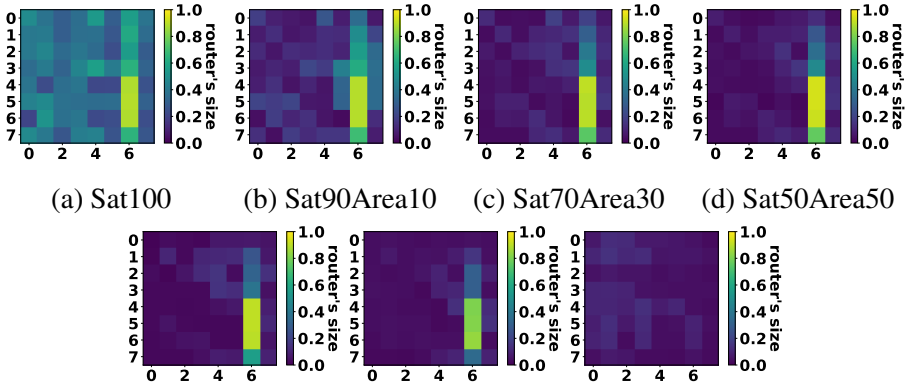
FIGURE 6.16 Taille moyenne des routeurs, en fonction de la proportion entre les rewards Sat et Pow, pour un entraînement de 300 époques sur le trafic hotspot30. Rewards reposant sur un CNN.

Enfin, les dernières expérimentations étudiées sont pour les rewards Sat et Area en CNN. Les résultats affichés figure 6.17 présentent un motif similaire au motif du trafic hotspot30, suggérant une optimisation quasi idéale de l'architecture des NoC générés. Cette conclusion est renforcée par les valeurs des scores attribués par les rewards, table 6.10. En effet, alors que le score du reward Sat (i.e. performance) est de 0.97 pour un reward Sat100, ce score reste élevé malgré l'augmentation de la proportion du reward Area, pour se maintenir à 0.89 pour le reward Sat10Area90. Bien entendu, ce score chute avec le reward Area100 puisque l'influence du reward Sat n'est plus présente dans l'entraînement du générateur. Ainsi, le

Valeurs		R-Sat	R-Pow	t-Big	t-Medium	t-Small
Rewards	Sat100	0.97	0.26	33.6%	32%	34.4%
	Sat90Pow10	0.93	0.32	29.8%	37.1%	33.1%
	Sat70Pow30	0.80	0.47	31.1%	31.5%	37.4%
	Sat50Pow50	0.74	0.61	7.5%	37%	55.5%
	Sat30Pow70	0.48	0.82	14.4%	40%	45.6%
	Sat10Pow90	0.37	0.9	14.7%	35.9%	49.4%
	Pow100	0.10	0.95	21%	36.3%	42.7%

TABLE 6.9 Valeurs des différentes variables d'entraînement. Reward Sat et Pow, trafic hotspot30. Rewards reposant sur un CNN.

reward Sat10Area90 retient particulièrement notre attention puisque ce dernier affiche des scores élevés pour chacun des rewards (0.89 pour le score Sat et 0.91 pour le score Area) tout en minimisant au maximum la taille des routeurs avec seulement 5% de routeurs Big et jusqu'à 68.4% de routeurs Small.



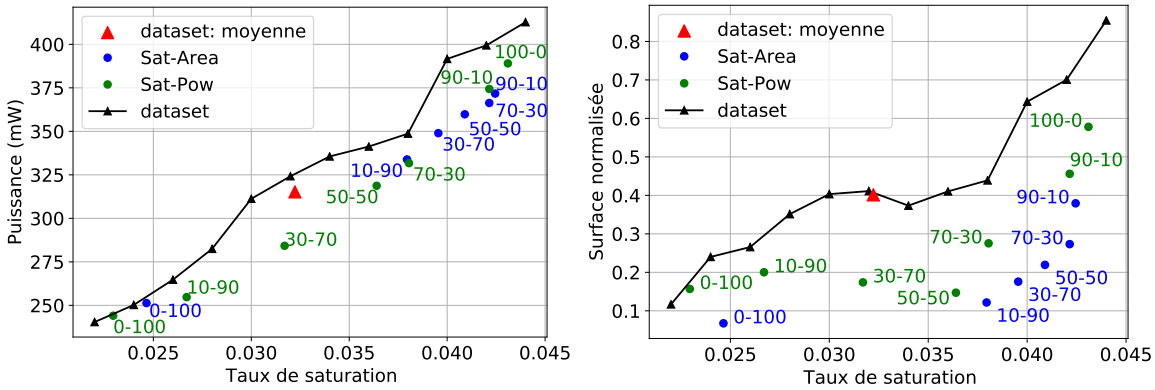
(a) Sat100 (b) Sat90Area10 (c) Sat70Area30 (d) Sat50Area50

(e) Sat30Area70 (f) Sat10Area90 (g) Area100

FIGURE 6.17 Taille moyenne des routeurs, en fonction de la proportion entre les rewards Sat et Area, pour un entraînement de 300 époques sur le trafic hotspot30. Rewards reposant sur des CNN.

Comparaison globale avec le dataset : Nous analysons ici les NoC générés par les générateurs entraînés avec les différentes combinaisons de rewards (GCN et CNN) pour extraire les combinaisons produisant les NoC avec la meilleure efficacité énergétique.

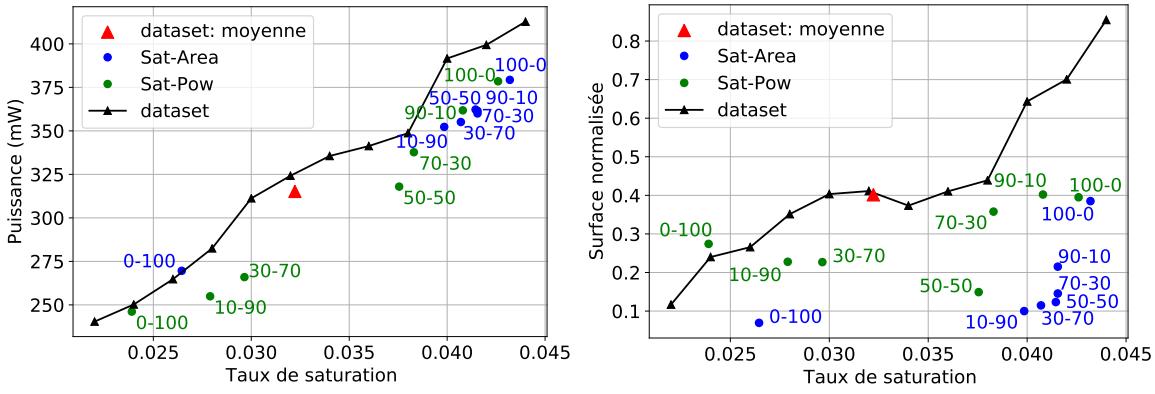
Tout d'abord les résultats de simulation des générations liées aux rewards GCN sont présentés sur la figure 6.18. On remarque directement que l'ensemble des résultats montrent



(a) Seuil de saturation et puissance.

(b) Seuil de saturation et surface

FIGURE 6.18 Comparaison des NoC générés et du dataset selon différentes métriques (seuil de saturation, énergie consommée, surface), lorsque soumis à un trafic hotspot30, pour les rewards GCN.



(a)

(b)

FIGURE 6.19 Comparaison des NoC générés et du dataset selon différentes métriques (seuil de saturation, énergie consommée, surface), lorsque soumis à un trafic hotspot30, pour les rewards CNN.

Valeurs		R-Sat	R-Area	t-Big	t-Medium	t-Small
Rewards	Sat100	0.97	0.60	33.6%	32%	34.4%
	Sat90Area10	0.97	0.8	14%	36%	50%
	Sat70Area30	0.96	0.87	7%	38.9%	54.1%
	Sat50Area50	0.96	0.88	5.5%	34.5%	60%
	Sat30Area70	0.94	0.89	5.5%	30%	64.5%
	Sat10Area90	0.89	0.91	5%	27.6%	68.4%
	Area100	0.20	0.94	0%	33.4%	66.6%

TABLE 6.10 Valeurs des différentes variables d'entraînement. Reward Sat et Area, trafic hotspot30. Rewards reposant sur un CNN.

une meilleure efficacité énergétique que le dataset. En effet, les données de simulation des ensembles générés sont toutes positionnées en-dessous des données du dataset.

Ensuite les résultats de simulation des générations liées aux rewards CNN sont présentés sur la figure 6.19. Excepté pour le reward Pow100 sur le graphique 6.19b, l'ensemble des résultats montrent une meilleure efficacité énergétique que le dataset. En effet, les données de simulation des ensembles générés sont une nouvelle fois toutes positionnées en-dessous des données du dataset.

Gains en efficacité énergétique : Les meilleures combinaisons de rewards sont déterminées comme précédemment, donnant Sat70Pow30 et Sat70Area30 comme les meilleures combinaisons de rewards GCN respectivement pour les types Sat-Pow et Sat-Area. Pour les rewards CNN, nous obtenons Sat50Pow50 et Sat70Area30 comme meilleures combinaisons, respectivement pour les types Sat-Pow et Sat-Area.

— *Sat70Pow30_GCN :* Les NoC produits par le générateur après un entraînement avec le reward Sat70Pow30 en GCN ont en moyenne un seuil de saturation supérieur de 0.58% à la moyenne du dataset (seuil de saturation de 3.8% contre 3.22% pour la moyenne du dataset), soit une augmentation de 18%. Pour ce qui est de la puissance consommée au seuil de saturation, cette dernière est de 332mW contre 315mW pour la moyenne du dataset, ce qui représente une augmentation de 5.4%. Enfin, comme pour le trafic uniforme, on mesure l'efficacité énergétique comme la performance (i.e. seuil de saturation) divisée par la puissance consommée. On obtient une amélioration d'environ 12.2% de l'efficacité énergétique des NoC générés, en comparaison avec la moyenne du dataset de départ.

— *Sat70Area30_GCN :* Les NoC produits par le générateur après un entraînement avec le reward Sat70Area30 en GCN ont en moyenne un seuil de saturation supérieur de 1% (seuil de saturation à 3.22% pour la moyenne du dataset contre 4.22% pour les NoC générés) soit une augmentation de 31%. Pour ce qui est de la puissance au seuil de saturation, cette dernière est de 366mW contre 315mW pour la moyenne du dataset, ce qui représente une augmentation de 16.2%. Enfin, on obtient une amélioration d'environ 12.55% de l'efficacité énergétique des NoC générés, en comparaison avec la moyenne du dataset de départ.

— *Sat50Pow50_CNN :* Les NoC produits par le générateur après un entraînement avec le reward Sat50Pow50 en CNN ont en moyenne un seuil de saturation supérieur de 0.54% à la moyenne du dataset (3.76% contre les 3.22% du dataset), soit une augmentation de 16.8%. Pour ce qui est de la puissance au seuil de saturation, cette dernière est de 318mW contre 315mW pour la moyenne du dataset, ce qui représente une augmentation inférieure à 1%. Enfin, on obtient une amélioration d'environ 15.5% de l'efficacité énergétique des NoC générés, en comparaison avec la moyenne du dataset de départ.

— *Sat70Area30_CNN :* Les NoC produits par le générateur après un entraînement avec le reward Sat70Area30 en CNN ont en moyenne un seuil de saturation supérieur de 0.96% (seuil de saturation à 3.2% pour la moyenne du dataset contre 4.16% pour les NoC générés), soit une augmentation de 29.8%. Pour ce qui est de la puissance au seuil de saturation, cette dernière est de 360mW contre 315mW pour la moyenne du dataset, ce qui représente une augmentation de 14.3%. Enfin, on obtient une amélioration d'environ 12.9% de l'efficacité énergétique des NoC générés, en comparaison avec la moyenne du dataset de départ.

6.3.3.4 Qualité d'optimisation : une analyse préliminaire

Dans les sections précédentes, nous avons montré que notre outil permet de rapidement balayer un espace de données de dimension importante en variant les combinaisons de rewards. Nous nous sommes concentrés sur l'intérêt d'un tel outil pour extraire un sous-ensemble optimisé en terme d'efficacité énergétique. Cependant, notre outil est plus globalement un outil d'optimisation multi-objectif.

Dans cette dernière section, nous proposons d'évaluer les qualités d'optimisation de notre outil, en fonction des réels objectifs modélisés par les rewards (i.e. seuil de saturation, puissance et surface). Pour cela, nous choisissons la métrique IGD (Inverted Generational Distance [45]) pour déterminer la qualité d'optimisation de notre outil. La métrique IGD se définit comme la distance euclidienne moyenne entre le vrai front Pareto et le front Pareto

des données générées par l'outil. Globalement, l'IGD permet de mesurer la convergence d'un ensemble A de solutions obtenues vers un ensemble de référence R, qui est dans l'idéal le vrai front Pareto. Elle est formulée par l'équation 6.3.

$$IGD(A,R) = \frac{1}{|R|} \sum_{r \in R} (min\{dist(r,a) | a \in A\}) \tag{6.3}$$

avec $|R|$ le nombre de solutions contenues dans R et dist(r, a) la distance euclidienne entre les solutions r et a.

Dans notre cas, nous souhaitons évaluer la qualité des données générées en fonction du vrai front Pareto. Cela nécessite de connaître l'intégralité de l'espace de conception, pour en extraire le front. Ainsi, nous nous plaçons dans un espace réduit et considérons la génération de NoC de type mesh 4x3, composés de 12 routeurs pouvant être de 3 types. Cela représente un espace de 531 441 combinaisons possibles, que nous simulons intégralement pour le trafic hotspot30.

L'ensemble des résultats (saturation, puissance et surface de chaque NoC) est normalisé entre 0 et 1 en fonction des minimum et maximum de chaque mesure. Ainsi, la valeur optimale du seuil de saturation est 1 (i.e. on souhaite le maximiser), et la valeur optimale de surface et de puissance est 0 (i.e. on souhaite minimiser ces métriques). Nous déterminons ensuite R à partir des données normalisées.

De cet espace de conception, nous extrayons 10k données pour construire notre dataset d'entraînement. À partir de ce dataset, nous entraînons nos trois rewards (seuil de saturation, puissance, surface), puis nous entraînons notre M-RWGAN avec x combinaisons de rewards.

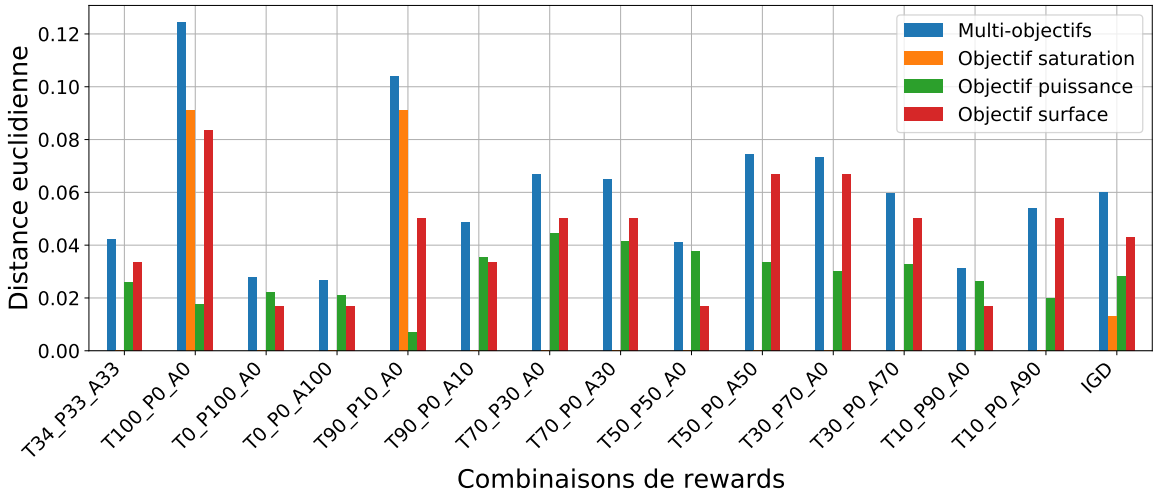


FIGURE 6.20 Distance euclidienne entre le meilleur NoC généré de chaque apprentissage et le vrai front Pareto, ainsi que le détail pour chaque objectif. Mesure de l'IGD - i.e. moyenne.

Pour chaque entraînement, nous générons un ensemble N de 100 NoC, et nous calculons $a_i = min\{dist(r,n)|n \in N\}$, le meilleur NoC généré par le $i^{\text{ème}}$ apprentissage. Finalement, nous calculons l'IGD, en appliquant la formule 6.3 avec $A = \{a_i|i \in [1..x]\}$.

Les résultats sont affichés sur la figure 6.20. Pour chaque entraînement (i.e. combinaison de rewards), les mesures de la meilleure solution sont affichées, soient : la distance euclidienne au vrai front Pareto (multi-objectif) i.e. a_i , et le détail des distances euclidiennes pour chaque objectif. Enfin, nous affichons l'IGD calculé pour l'optimisation multi-objectif ainsi que le détail des distances moyennes pour chaque objectif. Ces derniers résultats sont décrits dans la suite, et comparés avec la moyenne de l'espace de conception :

— multi-objectif (i.e. IGD) : 0.06, moyenne de l'espace de conception : 0.373

— Seuil de saturation : 0.013, moyenne de l'espace de conception : 0.044

— Puissance : 0.028, moyenne de l'espace de conception : 0.053

— Surface : 0.043, moyenne de l'espace de conception : 0.367

En comparant notre IGD avec la distance moyenne de l'ensemble des points de l'espace de conception, nous constatons que notre outil possède d'importantes performances d'optimisation. En effet, nous obtenons une réduction de la distance au front Pareto de 85% (de 0.373 pour la moyenne de l'espace de conception, à 0.06 pour l'IGD de notre outil). De plus, considérant une distance maximale de $\sqrt{3} \approx 1.73$ (i.e. 3 objectifs), notre score de 0.06 démontre une grande qualité d'optimisation. Enfin, nous obtenons également une réduction des distances pour chaque objectif de 70%, 47% et 88% respectivement pour le seuil de saturation, la puissance et la surface, en comparaison avec la moyenne de l'espace de conception. Ainsi, ces derniers résultats soulignent le potentiel d'optimisation multi-objectif de notre outil.

6.4 Résumé

Dans cette dernière contribution, nous avons proposé une architecture complexe de réseau de neurones, nommée M-RWGAN. Cette architecture se différencie du RWGAN du chapitre précédent par le niveau complexité possible de la fonction de récompense servant d'optimisation du générateur. En effet, nous montrons que, via l'utilisation de plusieurs rewards simples, il est possible de créer un reward hybride approximant une fonction d'optimisation complexe.

Ce travail ouvre des perspectives intéressantes sur le développement d'outils de CAO. De futurs travaux devront explorer les capacités du M-RWGAN et combiner la génération de matrice d'adjacence A du chapitre précédent, avec la génération de matrice de caractéristiques

X de ce dernier chapitre pour proposer une génération complète de NoC. Par extension, cet outil se généralise à la production de graphes optimisés, et n'est donc pas restreint au domaine des NoC.

Les pistes pour de futurs travaux sont développées plus en détails dans les perspectives de cette thèse, chapitre 7.

Chapitre 7

Conclusion et perspectives

Sommaire

7.1 Conclusion générale

L'efficacité énergétique est le nouveau moteur guidant l'amélioration des systèmes de calcul. En effet, après une longue période d'amélioration des performances, la prise en compte des dépenses énergétiques s'est ajoutée aux critères d'amélioration des calculateurs. Avec le développement de systèmes de calcul de plus en plus complexes – i.e. multi-coeurs, many-coeurs, hétérogènes, etc. – on parle désormais de calcul distribué et/ou parallèle, en référence aux multiples ressources de calcul disponibles. Ainsi, pour optimiser le calcul parallèle, il est nécessaire d'optimiser d'une part le support matériel exécutant les tâches, mais aussi la façon dont le calcul sera réparti parmi les ressources. L'essor des techniques d'apprentissage automatique a permis d'apporter des solutions toujours plus performantes dans un grand nombre de domaines. Alors que le contrôle et la conception de systèmes de

calcul efficaces énergétiquement sont des tâches de plus en plus complexes au regard du nombre croissant de paramètres à prendre en compte, l'usage de l'IA s'avère prometteur. Cette thèse avait donc pour but d'explorer les techniques d'apprentissage pour apporter des solutions aux problématiques du calcul parallèle.

Dans un premier temps, nous avons tenté d'adresser le problème du contrôle adaptatif multi-niveaux du calcul parallèle, pour l'optimisation de l'efficacité énergétique du système exécutant le calcul. Une analyse de la littérature nous a tout d'abord permis de recenser les métriques et les outils existants pour connaître l'efficacité énergétique de l'exécution d'une tâche sur un système de calcul. Nous avons remarqué un manque de solutions proches du niveau d'exécution des applications, ne dépendant pas des compteurs matériels propres à chaque système. De plus, peu de travaux adressaient en particulier les applications parallélisées avec OpenMP qui pourtant est le modèle de programmation parallèle le plus utilisé. Ensuite, nous nous sommes intéressés aux méthodes de contrôle existantes. L'étude de l'état de l'art a permis d'identifier les techniques d'apprentissage par renforcement comme les plus efficaces pour l'apprentissage et le contrôle dynamique. Bien que les solutions proposées adressent différents paramètres clés dans l'efficacité énergétique, tels que le DVFS et DPM ou encore le mapping, peu de solutions adressent l'optimisation de plusieurs de ces leviers à la fois.

Ainsi, notre travail s'est porté sur la formalisation d'une métrique collectée au niveau du runtime OpenMP. Cette solution a pour avantage de ne pas dépendre des compteurs matériels propres à chaque système de calcul et de bénéficier de la portabilité d'OpenMP. De ce fait, notre métrique, appelée CpJ, permet un suivi en temps réel de l'efficacité énergétique d'une application OpenMP, et peut être utilisée sur tout système supportant OpenMP. Cette métrique nous a ensuite permis de créer un système de contrôle basé directement sur l'efficacité énergétique. Ce contrôle repose sur un apprentissage par renforcement ne nécessitant aucun entraînement avant utilisation. Ce système a été évalué pour des applications statiques, sur un benchmark synthétique montrant un potentiel d'amélioration de l'efficacité énergétique allant jusqu'à 469% selon le type de workload exécuté et jusqu'à 17% pour l'application DGEMM, comparé aux gouverneurs Linux. Ensuite, afin de prendre en compte les phases d'exécution d'une application, un outil basé sur le principe des auto-encodeurs a été développé. Grâce à cette notion supplémentaire de phase d'exécution, le contrôle dynamique implémenté permet d'adapter les paramètres du système de calcul aux phases de fonctionnement de l'application considérée. Les premiers tests sur une application synthétique à deux phases montrent un gain de 50% comparé au gouverneur *ondemand* de Linux. Ces résultats très positifs montrent le potentiel de l'IA pour résoudre des problèmes complexes améliorant l'efficacité énergétique de nos systèmes de calcul. Cependant, les résultats proposés dans

cette thèse sont insuffisants pour valider une utilisation sur des applications réelles. En effet, une grande partie des résultats reposent sur une suite de benchmarks synthétiques. De plus, les résultats sur applications réelles sont mitigés, avec un contrôle efficace montrant des gains positifs sur l'application statique DGEMM, et des résultats moyens sur l'application SRAD. Cette analyse est donc à mener dans de prochains travaux. Ensuite, la définition des actions de l'apprentissage par renforcement semble sous-optimale. En effet, bien que l'utilisation de l'auto-encodeur permette d'obtenir une notion de phase en entrée du contrôleur, une formulation plus minutieuse de notre problème en problème d'apprentissage par renforcement devrait permettre de ne pas nécessiter une détection de phase complémentaire.

Après s'être intéressé à l'optimisation en-ligne du calcul parallèle via la proposition d'un contrôle dynamique, nous nous sommes intéressés à l'optimisation de la conception du matériel exécutant le calcul. Notre étude s'est rapidement dirigée vers les SoC, particulièrement sensibles aux dépenses énergétiques, en témoigne leur utilisation en tant que systèmes embarqués. De plus, les SoC sont généralement utilisés pour des applications spécifiques, et non pour un usage généraliste. Ainsi, les perspectives d'optimisation sont vastes puisque dépendantes de l'usage.

Un premier constat sur les dépenses énergétiques des SoC nous a permis de cibler l'optimisation des réseaux sur puce, appelés NoC. Depuis leur apparition, les NoC se sont imposés comme le module de communication par défaut des SoC, de par leur flexibilité et leur facilité d'implémentation. Cependant, ces derniers représentent une part significative de la consommation énergétique des SoC, en partie liée à un surdimensionnement comparé aux trafics qui les parcourent. Ainsi, une optimisation du design des NoC en fonction de leur usage permettrait de réduire leur consommation énergétique tout en maintenant leurs performances, ce qui équivaut à améliorer leur efficacité énergétique.

Nous avons adressé ce problème de conception de NoC optimisés via le prisme de l'espace de conception. Une difficulté majeure qui fait face aux concepteurs de NoC est le nombre important de paramètres à ajuster. En effet, la topologie du réseau et les caractéristiques des routeurs sont autant de paramètres à régler, tout en considérant le type de trafic auquel sera sujet le NoC, et la méthode de routage. Après une étude de la littérature, nous nous sommes aperçus que les solutions existantes consistent à générer une seule configuration optimisée. De plus, ces méthodes se concentrent sur un nombre limité de critères d'optimisation, et elles ne garantissent pas que le design généré soit le meilleur possible. Pour remédier à cela, nous avons donc proposé un outil permettant de réduire l'espace de conception afin de proposer au concepteur de NoC un sous-espace optimisé des designs de NoC. Ainsi, plutôt que de proposer directement une configuration de NoC possiblement sous-optimisée, nous générons un ensemble de solutions afin de faciliter l'analyse d'un expert.

Notre solution repose sur une architecture particulière de GAN. Nous avons démontré l'utilité de cette méthode sur différentes preuves de concept. En particulier, nous avons proposé l'architecture de M-RWGAN permettant de générer une donnée optimisée selon une multitude de critères définis par l'utilisateur. Alors que cette solution est appliquée à la génération de NoC, il s'avère que le concept de M-RWGAN est bien plus large et ne se restreint pas aux réseaux sur puce. Dans notre cas de la génération de NoC, une première application a consisté à générer avec succès des topologies de NoC optimisées. Bien que probants, de futurs travaux pourront développer ces résultats. Par la suite, nous nous sommes intéressés à la génération de NoC hétérogènes et nous avons obtenu des résultats encourageants, démontrant l'efficacité de notre M-RWGAN. Cette dernière contribution ouvre en particulier des perspectives pour le développement de nouveaux outils de CAO. Enfin, l'objectif initial de générer des NoC complets optimisés n'est pas encore rempli. En effet, un NoC est décrit par sa topologie et par les éléments qui le composent. Nous avons proposé une solution pour chacun de ces problèmes (i.e. génération de topologies puis génération de matrices de caractéristiques), mais il manque une solution hybride permettant de générer l'ensemble des éléments d'un NoC optimisé. Cela fera l'objet de futurs travaux.

7.2 Perspectives

7.2.1 Amélioration du contrôle d'application basée sur l'apprentissage par renforcement

. **Contribution sur l'auto-encodeur :** l'auto-encodeur utilisé pour détecter automatiquement les phases d'une application est un outil développé dans le premier axe. N'étant qu'un simple outil, il n'a été validé que sur les applications à optimiser. Des travaux intéressants pourront être conduits pour développer cet outil et pousser sa validation sur un large panel d'applications afin d'en déterminer précisément ses points forts et ses limites.

. **Extension des résultats de validation :** notre système de contrôle a été testé sur deux applications réelles, une monotone (i.e. DGEMM) et une à deux phases (i.e. SRAD). Afin d'apporter une validation plus robuste, les résultats devront être étendus sur un plus grand nombre d'applications réelles. En particulier, il s'est avéré difficile de trouver des benchmarks réels multi-phases avec plus de deux phases visibles sur les traces de chunks. À cette fin, une suite de benchmarks pseudo-synthétiques pourrait être produite à partir d'un ensemble d'applications réelles monotones exécutées successivement, donnant un workload final avec plusieurs phases d'exécutions.

. **Extension pour la gestion d'applications concurrentes :** la contribution présentée pour le contrôle dynamique d'applications OpenMP n'agit que sur un seul workload. Or, les systèmes de calcul exécutent généralement plusieurs tâches en même temps. Une amélioration de notre système de contrôle serait de considérer plusieurs applications à la fois, et de déterminer le meilleur compromis quant à la répartition des ressources.

. **Amélioration de la loi de contrôle :** les capacités de l'apprentissage par renforcement sur d'autres cas d'usage suggèrent qu'il serait possible de se passer d'un outil de détection de phase en entrée du RL. De futurs travaux devront consister à revoir la définition de la loi de contrôle et des actions possibles afin de se dispenser de l'utilisation du module de détection de phase. Une première piste pourrait se diriger vers une reformulation des actions possibles. Avec un nouvel ensemble d'actions, la formule complète de la fonction de perte du DQN pourrait être pertinente. Cela permettrait de tirer profit de la notion temporelle de ce loss, qui prend en considération les actions passées et non uniquement la dernière action.

7.2.2 Développement de l'outil M-RWGAN pour la réduction d'espace de conception

. **Génération de graphes complets :** dans ces travaux, le RWGAN et son extension en M-RWGAN ont permis de respectivement implémenter des preuves de concept pour la production de matrices d'adjacences optimisées, et de matrices de caractéristiques optimisées. La suite logique sera de produire, par le biais d'une même IA, les deux matrices de description de graphes (i.e. A et X). Pour cela, la piste des GNN (*Graph Neural Networks*) devra être approfondie.

. **Généralisation de l'outil :** Le problème de la réduction de l'espace de conception n'est pas un problème spécifique à la conception de NoC. Le concept de M-RWGAN pourrait tout d'abord être étendu au domaine des graphes grâce à l'utilisation des GNN. Ensuite, on peut généraliser le principe de réduction de l'espace de conception à la réduction d'un espace de données. Ainsi, l'usage pourrait se diversifier à tout problème nécessitant une réduction d'un espace de données.

7.3 Publications

7.3.1 Publications dans des conférences internationales

. **M. Mirka**, G. Devic, F. Bruguier, G. Sassatelli et A. Gamatié, "Automatic Energy-Efficiency Monitoring of OpenMP Workloads," *14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2019, pp. 43-50, DOI : 10.1109/ReCoSoC48741.2019.9034988, HAL : lirmm-02183901.

. **M. Mirka**, G. Sassatelli et A. Gamatié, "Online Learning for Dynamic Control of OpenMP Workloads," *9th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2020, pp. 1-6, DOI : 10.1109/MOCAST49295.2020.9200292, HAL : hal-02565961.

. **M. Mirka**, M. France Pillois, G. Sassatelli, et A. Gamatié, "GANNoC : A Framework for Automatic Generation of NoC Topologies using Generative Adversarial Networks", *In Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation : Methods and Tools Proceedings (DroneSE and RAPIDO '21)*, 2021, pp. 51–58, DOI :10.1145/3444950.3447283, HAL : lirmm-03107918v2.

. **M. Mirka**, M. France Pillois, G. Sassatelli, et A. Gamatié, "A Generative AI for Heterogeneous Network-on-Chip Design Space Pruning," *Design, Automation and Test in Europe Conference (DATE)*, 2022, HAL : lirmm-0347591.

7.3.2 Poster

. **M. Mirka**, G. Sassatelli et A. Gamatié, "Energy-Efficiency Metric for Real-Time Monitoring of OpenMP Programs Executing on Multicore Systems," *13ème Colloque National du GDR SOC²*, 2019, HAL : lirmm-03326276v2.

Bibliographie

[1] Intel performance counter monitor. https://software.intel.com/en-us/articles/intel-performance-counter-monitor. Accessed : 2019-09-15.

[2] Martín 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 *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 265–283, 2016.

[3] Ahmed Ben Achballah and Slim Ben Saoud. A survey of network-on-chip tools. *International Journal of Advanced Computer Science and Applications*, 4(9), 2013. URL : http://dx.doi.org/10.14569/IJACSA.2013.040910, doi:10.14569/ijacsa.2013.040910.

[4] Ankur Agarwal and Ravi Shankar. Survey of network on chip (noc) architectures & contributions. *Journal of Engineering, Computing and Architecture*, 3, 01 2009.

[5] Khurshid Ahmad and Muhammad Athar Javed Sethi. Review of network on chip routing algorithms. *EAI Endorsed Transactions on Context-aware Systems and Applications*, 7(22), 12 2020. doi:10.4108/eai.23-12-2020.167793.

[6] Kishwar Ahmed, Jason Liu, Abdel-Hameed Badawy, and Stephan Eidenbenz. A brief history of hpc simulation and future challenges. In *2017 Winter Simulation Conference (WSC)*, pages 419–430, 2017. doi:10.1109/WSC.2017.8247804.

[7] Ferdinando Alessi, Peter Thoman, Giorgis Georgakoudis, Thomas Fahringer, and Dimitrios Nikolopoulos. Application-level energy awareness for openmp. volume 9342, pages 219–232, 10 2015. doi:10.1007/978-3-319-24595-9_16.

[8] Lulwah Alhubail, Masoomeh Jasemi, and Nader Bagherzadeh. Noc design methodologies for heterogeneous architecture. In *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 299–306, 2020. doi:10.1109/PDP50117.2020.00052.

[9] Isiaka A. Alimi, Romil K. Patel, Oluyomi Aboderin, Abdelgader M. Abdalla, Ramoni A. Gbadamosi, Nelson J. Muga, Armando N. Pinto, and António L. Teixeira. Network-on-chip topologies : Potentials, technical challenges, recent advances and research direction. *IntechOpen*, 2021. URL : https://www.intechopen.com/online-first/

network-on-chip-topologies-potentials-technical-challenges-recent-advances-and-research-direction, doi:10.5772/intechopen.97262.

[10] Francisco Almeida, Javier Arteaga, Vicente Blanco, and Alberto Cabrera. Energy measurement tools for ultrascale computing : A survey. *Supercomputing Frontiers and Innovations*, 2(2) :64–76, 2015. URL : http://superfri.org/superfri/article/view/45, doi:10.14529/jsfi150204.

[11] Xin An, Sarra Boumedien, Abdoulaye Gamatié, and Éric Rutten. Classy : A clock analysis system for rapid prototyping of embedded applications on mpsocs. In *Proceedings of the 15th International Workshop on Software and Compilers for Embedded Systems*, SCOPES '12, page 3–12, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2236576.2236577.

[12] Xin An, Abdoulaye Gamatié, and Eric Rutten. High-level design space exploration for adaptive applications on multiprocessor systems-on-chip. *Journal of Systems Architecture*, 61(3) :172–184, 2015. URL : https://www.sciencedirect.com/science/article/pii/S1383762115000053, doi:https://doi.org/10.1016/j.sysarc.2015.02.002.

[13] Xin An, Eric Rutten, Jean-Philippe Diguet, and Abdoulaye Gamatié. Model-based design of correct controllers for dynamically reconfigurable architectures. *ACM Transactions on Embedded Computing Systems*, 15(3), May 2016. doi:10.1145/2873056.

[14] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017. arXiv:1701.07875.

[15] Mario Badr and Natalie Enright Jerger. Synfull : Synthetic traffic models capturing cache coherent behaviour. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 109–120, 2014. doi:10.1109/ISCA.2014.6853236.

[16] Sung-Yong Bang, Kwanhu Bang, Sungroh Yoon, and Eui-Young Chung. Run-time adaptive workload estimation for dynamic voltage scaling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(9) :1334–1347, Sep. 2009. doi:10.1109/TCAD.2009.2024706.

[17] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12) :33–37, 2007. doi:10.1109/MC.2007.443.

[18] Karunakar R. Basireddy, Amit Kumar Singh, Bashir Al-Hashimi, and Geoff V. Merrett. Adamd : Adaptive mapping and dvfs for energy-efficient heterogeneous multi-cores. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, August 2019. URL : https://eprints.soton.ac.uk/433327/.

[19] Yaniv Ben-Itzhak, Eitan Zahavi, Israel Cidon, and Avinoam Kolodny. Hnocs : Modular open-source simulator for heterogeneous nocs. In *2012 International Conference on Embedded Computer Systems (SAMOS)*, pages 51–57, 2012. doi:10.1109/SAMOS.2012.6404157.

[20] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3) :299–316, 2000. doi:10.1109/92.845896.

[21] Malini Bhandaru and Eric Dehaemer. Providing energy efficient turbo operation of a processor, Sep. 19 2013, patent App. PCT/US2012/028,865.[Online] : http ://www.google.com/patents/WO2013137859A1 ?cl=en.

[22] A. Bhattacharyya, S. Sotiriadis, and C. Amza. Online phase detection and characterization of cloud applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 98–105, Dec 2017. doi:10.1109/CloudCom.2017.21.

[23] Muhammad Bhatti, Cécile Belleudy, and Michel Auguin. Hybrid power management in real time embedded systems : An interplay of dvfs and dpm techniques. *Real-Time Systems*, 47 :143–162, 03 2011. doi:10.1007/s11241-011-9116-y.

[24] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[25] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The m5 simulator : Modeling networked systems. *IEEE Micro*, 26(4) :52–60, 2006. doi:10.1109/MM.2006.82.

[26] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2) :1–7, aug 2011. doi:10.1145/2024716.2024718.

[27] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1) :1–es, June 2006. doi:10.1145/1132952.1132953.

[28] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN : Generating graphs via random walks. volume 80 of *Proceedings of Machine Learning Research*, pages 610–619. PMLR, 2018.

[29] Bokhari. On the mapping problem. *IEEE Transactions on Computers*, C-30(3) :207–214, 1981. doi:10.1109/TC.1981.1675756.

[30] Haseeb Bokhari, Haris Javaid, Muhammad Shafique, Jörg Henkel, and Sri Parameswaran. Malleable noc : Dark silicon inspired adaptable network-on-chip. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1245–1248, 2015. doi:10.7873/DATE.2015.0694.

[31] Shekhar Borkar. Thousand core chips : A technology perspective. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, page 746–749, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1278480.1278667.

[32] Rabab Bouziane, Erven Rohou, and Abdoulaye Gamatié. Compile-time silent-store elimination for energy efficiency : An analytic evaluation for non-volatile cache memory. In *Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation : Methods and Tools*, RAPIDO '18, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3180665.3180666.

[33] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *International Journal of High Performance Computing Applications (IJHPCA)*, 14(3) :189–204, August 2000. URL : http://dx.doi.org/10.1177/109434200001400303, doi:10.1177/109434200001400303.

[34] Miles Brundage, Shahar Avin, Jack Clark, Helen Toner, Peter Eckersley, Ben Garfinkel, Allan Dafoe, Paul Scharre, Thomas Zeitzoff, Bobby Filar, Hyrum S. Anderson, Heather Roff, Gregory C. Allen, Jacob Steinhardt, Carrick Flynn, Seán Ó hÉigeartaigh, Simon Beard, Haydn Belfield, Sebastian Farquhar, Clare Lyle, Rebecca Crootof, Owain Evans, Michael Page, Joanna Bryson, Roman Yampolskiy, and Dario Amodei. The malicious use of artificial intelligence : Forecasting, prevention, and mitigation. *CoRR*, abs/1802.07228, 2018. URL : http://arxiv.org/abs/1802.07228, arXiv:1802.07228.

[35] A. Butko, F. Bruguier, A. Gamatié, G. Sassatelli, D. Novo, L. Torres, and M. Robert. Full-system simulation of big.little multicore architecture for performance and energy exploration. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 201–208, Sep. 2016. doi:10.1109/MCSoC.2016.20.

[36] Anastasiia Butko, Abdoulaye Gamatié, Gilles Sassatelli, Lionel Torres, and Michel Robert. Design exploration for next generation high-performance manycore on-chip systems : Application to big.little architectures. In *2015 IEEE Computer Society Annual Symposium on VLSI*, pages 551–556, 2015. doi:10.1109/ISVLSI.2015.28.

[37] Anastasiia Butko, Rafael Garibotti, Luciano Ost, Vianney Lapotre, Abdoulaye Gamatie, Gilles Sassatelli, and Chris Adeniyi-Jones. A trace-driven approach for fast and accurate simulation of manycore architectures. In *The 20th Asia and South Pacific Design Automation Conference*, pages 707–712, 2015. doi:10.1109/ASPDAC.2015.7059093.

[38] Nicola De Cao and Thomas Kipf. Molgan : An implicit generative model for small molecular graphs, 2018. arXiv:1805.11973.

[39] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. Cycle-accurate network on chip simulation with noxim. *ACM Trans. Model. Comput. Simul.*, 27(1), aug 2016. doi:10.1145/2953878.

[40] Georgios Chasparis, Michael Rossbory, and Vladimir Janjic. Efficient dynamic pinning of parallelized applications by reinforcement learning with applications. pages 164–176, 08 2017. doi:10.1007/978-3-319-64203-1_12.

[41] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron. Rodinia : A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, Oct 2009. doi:10.1109/IISWC.2009.5306797.

[42] J. Chen, P. Gillard, and Cheng Li. Network-on-chip (noc) topologies and performance : A review. 2011.

[43] Yen-Lin Chen, Ming-Feng Chang, Chao-Wei Yu, Xiu-Zhi Chen, and Wen-Yew Liang. Learning-directed dynamic voltage and frequency scaling scheme with adjustable performance for single-core and multi-core embedded and mobile systems. In *Sensors*, 2018. doi:10.3390/s18093068.

[44] François Chollet et al. Keras. https://keras.io, 2015.

[45] Carlos A. Coello Coello and Margarita Reyes Sierra. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In Raúl Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Humberto Sossa, editors, *MICAI 2004 : Advances in Artificial Intelligence*, pages 688–697, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-24694-7_71.

[46] Odilia Coi, Guillaume Patrigeon, Sophiane Senni, Lionel Torres, and Pascal Benoit. A novel SRAM -STT-MRAM hybrid cache implementation improving cache performance. In *NANOARCH : Nanoscale Architectures*, pages 39–44, Newport, United States, July 2017. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-01548938, doi:10.1109/NANOARCH.2017.8053704.

[47] Junio C R Da Silva, Lorena Leão, Vinicius Petrucci, Abdoulaye Gamatié, and Fernando Pereira. Mapping Computations in Heterogeneous Multicore Systems with Statistical Regression on Inputs. In *Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–8, Online, Brazil, November 2020. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-03018543, doi:10.1109/SBESC51047.2020.9277863.

[48] Junio Cezar Ribeiro da Silva, Fernando Magno Quinto Pereira, Michael Frank, and Abdoulaye Gamatié. A compiler-centric infra-structure for whole-board energy measurement on heterogeneous android systems. In *2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8, 2018. doi:10.1109/ReCoSoC.2018.8449378.

[49] William James Dally and Brian Patrick Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[50] W.J. Dally and B. Towles. Route packets, not wires : on-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 684–689, 2001. doi:10.1109/DAC.2001.156225.

[51] Bhavya K. Daya, Chia-Hsin Owen Chen, Suvinay Subramanian, Woo-Cheol Kwon, Sunghyun Park, Tushar Krishna, Jim Holt, Anantha P. Chandrakasan, and Li-Shiuan Peh. Scorpio : A 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 25–36, 2014. doi:10.1109/ISCA.2014.6853232.

[52] R. Dick. Embedded system synthesis benchmark suites (e3s). URL : http://ziyang.eecs.umich.edu/~dickrp/e3s/.

[53] Thanh Do, Suhib Rawshdeh, and Weisong Shi. ptop : A process-level power profiling tool. In *in Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09*, 2009.

[54] Bryan Donyanavard, Tiago Mück, Santanu Sarma, and Nikil Dutt. Sparta : Runtime task allocation for energy efficient heterogeneous manycores. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, Oct 2016. doi:10.1145/2968456.2968459.

[55] Charles Effiong, Gilles Sassatelli, and Abdoulaye Gamatie. Exploration of a scalable and power-efficient asynchronous network-on-chip with dynamic resource allocation. *Microprocessors and Microsystems*, 60 :173–184, 2018. URL : https://www.sciencedirect.com/science/article/pii/S0141933118300474, doi:https://doi.org/10.1016/j.micpro.2018.05.003.

[56] Charles Emmanuel Effiong. *Exploration of multicore systems based on silicon integrated communication networks*. Theses, Université Montpellier, November 2017. URL : https://tel.archives-ouvertes.fr/tel-01944111.

[57] Charles Emmanuel Effiong, Gilles Sassatelli, and Abdoulaye Gamatié. Distributed and Dynamic Shared-Buffer Router for High-Performance Interconnect. In *NOCS : Networks-on-Chip Symposium*, number Art N°2, pages 1–8, Seoul, South Korea, October 2017. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-01622889, doi:10.1145/3130218.3130223.

[58] Charles Emmanuel Effiong, Gilles Sassatelli, and Abdoulaye Gamatié. Scalable and Power-Efficient Implementation of an Asynchronous Router with Buffer Sharing. In *DSD : Digital System Design*, pages 171–178, Vienna, Australia, August 2017. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-01622885, doi:10.1109/DSD.2017.55.

[59] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, 2011. doi:10.1145/2000064.2000108.

[60] Shuangfei Fan and Bert Huang. Labeled graph generative adversarial networks, 2019. arXiv:1906.03220.

[61] Abdoulaye Gamatié, Xin An, Ying Zhang, An Kang, and Gilles Sassatelli. Empirical Model-Based Performance Prediction for Application Mapping on Multicore Architectures. *Journal of Systems Architecture*, 98 :1–16, September 2019. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-02151502, doi:10.1016/j.sysarc.2019.06.001.

[62] Abdoulaye Gamatié, Guillaume Devic, Gilles Sassatelli, Stefano Bernabovi, Philippe Naudin, and Michael Chapman. Towards Energy-Efficient Heterogeneous Multicore Architectures for Edge Computing. *IEEE Access*, 7 :49474–49491, April 2019. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-02099306, doi:10.1109/ACCESS.2019.2910932.

[63] Abdoulaye Gamatié, Roman Ursu, Manuel Selva, and Gilles Sassatelli. Performance Prediction of Application Mapping in Manycore Systems with Artificial Neural Networks. In *MCSoC : Embedded Multicore/Many-core Systems-on-Chip*, pages 185–192, Lyon, France, September 2016. IEEE. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-01385641, doi:10.1109/MCSoC.2016.17.

[64] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The scalasca performance toolset architecture. *Concurr. Comput. : Pract. Exper.*, 22(6) :702–719, April 2010. URL : http://dx.doi.org/10.1002/cpe.v22:6, doi:10.1002/cpe.v22:6.

[65] Giorgis Georgakoudis, Hans Vandierendonck, Peter Thoman, Bronis R. De Supinski, Thomas Fahringer, and Dimitrios S. Nikolopoulos. Scalo : Scalability-aware parallelism orchestration for multi-threaded workloads. *ACM Transactions on Architecture and Code Optimization*, 14(4) :54 :1–54 :25, 2017. URL : http://doi.acm.org/10.1145/3158643, doi:10.1145/3158643.

[66] Jose Ricardo Gomez-Rodriguez, Remberto Sandoval-Arechiga, Salvador Ibarra-Delgado, Viktor Ivan Rodriguez-Abdala, Jose Luis Vazquez-Avila, and Ramon Parra-Michel. A survey of software-defined networks-on-chip : Motivations, challenges and opportunities. *Micromachines*, 12(2), 2021. URL : https://www.mdpi.com/2072-666X/12/2/183, doi:10.3390/mi12020183.

[67] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. arXiv:1406.2661.

[68] Daniele Grattarola and Cesare Alippi. Graph neural networks in tensorflow and keras with spektral, 2020. arXiv:2006.12138.

[69] Paul V. Gratz and Stephen W. Keckler. Realistic workload characterization and analysis for networks-on-chip design. In *The 4th Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, 2010.

[70] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017. arXiv:1704.00028.

[71] Tinghao Guo, Daniel R. Herber, and James T. Allison. Circuit synthesis using generative adversarial networks (gans). In *AIAA Scitech 2019 Forum*, 2019. doi:10.2514/6.2019-2350.

[72] Ujjwal Gupta, Sumit K. Mandal, Manqing Mao, Chaitali Chakrabarti, and Umit Y. Ogras. A deep q-learning approach for dynamic management of heterogeneous processors. *IEEE Computer Architecture Letters*, 18(1) :14–17, 2019. doi:10.1109/LCA.2019.2892151.

[73] Ujjwal Gupta, Chetan Arvind Patil, Ganapati Bhat, Prabhat Mishra, and Umit Y. Ogras. Dypo : Dynamic pareto-optimal configuration selection for heterogeneous mpsocs. *ACM Transactions on Embedded Computing Systems*, 16(5s), September 2017. doi:10.1145/3126530.

[74] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. An energy efficiency feature survey of the intel haswell processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 896–904, May 2015. doi:10.1109/IPDPSW.2015.70.

[75] John L. Hennessy and David A. Patterson. A new golden age for computer architecture : Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development, June 2018. URL : https://iscaconf.org/isca2018/turing_lecture.html, doi:10.1109/ISCA.2018.00011.

[76] Joel Hestness, Boris Grot, and Stephen W. Keckler. Netrace : Dependency-driven trace-based network-on-chip simulation. In *Proceedings of the Third International Workshop on Network on Chip Architectures (NoCArc '10)*, page 31–36, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1921249.1921258.

[77] Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 3–10, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. URL : http://dl.acm.org/citation.cfm?id=2987189.2987190.

[78] Torsten Hoefler, Emmanuel Jeannot, and Guillaume Mercier. An Overview of Process Mapping Techniques and Algorithms in High-Performance Computing. In Emmanuel Jeannot and Julius Zilinskas, editors, *High Performance Computing on Complex Environments*, pages 75–94. Wiley, June 2014. URL : https://hal.inria.fr/hal-00921626.

[79] John H. Holland. *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[80] Yatin Hoskote, Sriram Vangal, Arvind Singh, Nitin Borkar, and Shekhar Borkar. A 5-ghz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5) :51–61, 2007. doi:10.1109/MM.2007.4378783.

[81] Jan Moritz Joseph, Lennart Bamberg, Imad Hajjar, Behnam Razi Perjikolaei, Alberto García-Ortiz, and Thilo Pionteck. Ratatoskr : An open-source framework for in-depth power, performance, and area analysis and optimization in 3d nocs, sep 2021. doi:10.1145/3472754.

[82] J. Jumper, R. Evans, A. Pritzel, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021. doi:10.1038/s41586-021-03819-2.

[83] A. B. Kahng, B. Lin, and S. Nath. Orion3.0 : A comprehensive noc router estimation tool. *IEEE Embedded Systems Letters*, 7(2) :41–45, 2015. doi:10.1109/LES.2015.2402197.

[84] Diethelm Kai. Tools for assessing and optimizing the energy requirements of high performance scientific computing software. *PAMM*, 16 :837–838, 2016. doi:10.1002/pamm.201610407.

[85] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017. URL : http://arxiv.org/abs/1710.10196, arXiv:1710.10196.

[86] Manho Kim, Daewook Kim, and G.E. Sobelman. Network-on-chip link analysis under power and performance constraints. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4 pp.–, 2006. doi:10.1109/ISCAS.2006.1693546.

[87] Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. Machine learning and manycore systems design : A serendipitous symbiosis. *Computer*, 51(7) :66–77, 2018. doi:10.1109/MC.2018.3011040.

[88] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4) :307–392, 2019. URL : http://dx.doi.org/10.1561/2200000056, doi:10.1561/2200000056.

[89] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. arXiv:1609.02907.

[90] Robert Kirby, Saad Godil, Rajarshi Roy, and Bryan Catanzaro. Congestionnet : Routing congestion prediction using deep graph neural networks. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 217–222, 2019. doi:10.1109/VLSI-SoC.2019.8920342.

[91] Bartosz Krawczyk. Learning from imbalanced data : open challenges and future directions. *Progress in Artificial Intelligence*, 2016. doi:10.1007/s13748-016-0094-0.

[92] Thibault Béziers la Fosse, Jean-Marie Mottu, Massimo Tisi, Jérôme Rocheteau, and Gerson Sunyé. Characterizing a source code model with energy measurements. In Nelly Condori-Fernández, Alessandra Bagnato, and Eva Kern, editors, *Proceedings of the 4th International Workshop on Measurement and Metrics for Green and Sustainable Software Systems co-located with Empirical Software Engineering International Week (ESEIW 2018), Oulu, Finland, - October 9, 2018*, volume 2286 of *CEUR Workshop Proceedings*, page 26. CEUR-WS.org, 2018. URL : http://ceur-ws.org/Vol-2286/paper_3.pdf.

[93] Sandia National Laboratories. High performance computing power application programming interface (api) specification. http ://powerapi.sandia.gov, 2014.

[94] Khalid Latif, Manuel Selva, Charles Effiong, Roman Ursu, Abdoulaye Gamatie, Gilles Sassatelli, Leonardo Zordan, Luciano Ost, Piotr Dziurzanski, and Leandro Soares Indrusiak. Design space exploration for complex automotive applications : An engine control system case study. In *Proceedings of the 2016 Workshop on Rapid Simulation and Performance Evaluation : Methods and Tools*, RAPIDO '16, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2852339.2852341.

[95] Hiliwi Leake Kidane and El-Bay Bourennane. Run-time reconfigurable network-on-chip : A survey. In *2018 15th International Multi-Conference on Systems, Signals Devices (SSD)*, pages 846–851, 2018. doi:10.1109/SSD.2018.8570678.

[96] Minhyeok Lee and Junhee Seok. Controllable generative adversarial network, 2019. arXiv:1708.00598.

[97] Minhyeok Lee and Junhee Seok. Score-guided generative adversarial networks, 2020. arXiv:2004.04396.

[98] Guangshuo Liu, Jinpyo Park, and Diana Marculescu. Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 54–61, 2013. doi:10.1109/ICCD.2013.6657025.

[99] Piotr Luszczek, Jack Dongarra, David Koester, Rolf Rabenseifner, Bob Lucas, Jeremy Kepner, John McCalpin, David Bailey, and Daisuke Takahashi. Introduction to the hpc challenge benchmark suite. 12 2004.

[100] Luis Alfonso Maeda-Nunez, Anup K. Das, Rishad A. Shafik, Geoff V. Merrett, and Bashir Al-Hashimi. Pogo : an application-specific adaptive energy minimisation approach for embedded systems. In *HiPEAC Workshop on Energy Efficiency with Heterogenous Computing (EEHCO) (19/01/15 - 21/01/15)*, January 2015. URL : https://eprints.soton.ac.uk/372694/.

[101] Sumit K. Mandal, Ganapati Bhat, Chetan Arvind Patil, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y. Ogras. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12) :2842–2854, 2019. doi:10.1109/TVLSI.2019.2926106.

[102] Suejb Memeti, Sabri Pllana, Alécio Binotto, Joanna Kołodziej, and Ivona Brandic. Using meta-heuristics and machine learning for software optimization of parallel computing systems : A systematic literature review. *Computing*, 101(8) :893–936, August 2019. doi:10.1007/s00607-018-0614-9.

[103] Amirhossein Mirhosseini, Mohammad Sadrosadati, Behnaz Soltani, Hamid Sarbazi-Azad, and Thomas F. Wenisch. Binochs : Bimodal network-on-chip for cpu-gpu heterogeneous systems. In *2017 Eleventh IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8, 2017. doi:10.1145/3130218.3130222.

[104] Maxime Mirka, Guillaume Devic, Florent Bruguier, Gilles Sassatelli, and Abdoulaye Gamatié. Automatic energy-efficiency monitoring of openmp workloads. In *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 43–50, 2019. doi:10.1109/ReCoSoC48741.2019.9034988.

[105] Maxime Mirka, Maxime France-Pillois, Gilles Sassatelli, and Abdoulaye Gamatié. A Generative AI for Heterogeneous Network-on-Chip Design Space Pruning. In *25th Design, Automation, and Test in Europe Conference (DATE 2022)*, Antwerp, Belgium, March 2022. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-03475912.

[106] Maxime Mirka, Maxime France France Pillois, Gilles Sassatelli, and Abdoulaye Gamatié. Gannoc : A framework for automatic generation of noc topologies using generative adversarial networks. In *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation : Methods and Tools Proceedings*, DroneSE and RAPIDO '21, page 51–58, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3444950.3447283.

[107] Maxime Mirka, Gilles Sassatelli, and Abdoulaye Gamatié. Energy-Efficiency Metric for Real-Time Monitoring of OpenMP Programs Executing on Multicore Systems. 13ème Colloque National du GDR SOC^2, June 2019. Poster. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-03326276.

[108] Maxime Mirka, Gilles Sassatelli, and Abdoulaye Gamatié. Online learning for dynamic control of openmp workloads. In *2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–6, 2020. doi:10.1109/MOCAST49295.2020.9200292.

[109] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014. arXiv:1411.1784.

[110] Asit K. Mishra, N. Vijaykrishnan, and Chita R. Das. A case for heterogeneous on-chip interconnects for cmps. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 389–399, 2011. doi:10.1145/2000064.2000111.

[111] Sparsh Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology (IJCAET)*, 6(4) :440–459, 2014. doi:10.1504/IJCAET.2014.065419.

[112] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540) :529–533, February 2015. doi:10.1038/nature14236.

[113] A. Molnos, S. Lesecq, J. Mottin, and D. Puschini. Investigation of q-learning applied to dvfs management of a system-on-chip. *IFAC-PapersOnLine*, 49(5) :278–284, 2016. 4th IFAC Conference on Intelligent Control and Automation SciencesICONS 2016. URL : https://www.sciencedirect.com/science/article/pii/S2405896316303299, doi:https://doi.org/10.1016/j.ifacol.2016.07.126.

[114] Fernando Moraes, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost. Hermes : an infrastructure for low area overhead packet-switching networks on chip. *Integration*, 38(1) :69–93, 2004. URL : https://www.sciencedirect.com/science/article/pii/S0167926004000185, doi:https://doi.org/10.1016/j.vlsi.2004.03.003.

[115] Thomas Moscibroda and Onur Mutlu. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, page 196–207, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1555754.1555781.

[116] Misbah Mubarak, Christopher D. Carothers, Robert B. Ross, and Philip Carns. Enabling parallel simulation of large-scale hpc network systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(1) :87–100, 2017. doi:10.1109/TPDS.2016.2543725.

[117] Rajeev Muralidhar, Renata Borovica-Gajic, and Rajkumar Buyya. Energy efficient computing systems : Architectures, abstractions and modeling to techniques and standards, 2020. arXiv:2007.09976.

[118] Anuja Naik and Tirumale K. Ramesh. Efficient network on chip (noc) using heterogeneous circuit switched routers. In *2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA)*, pages 1–6, 2016. doi:10.1109/VLSI-SATA.2016.7593043.

[119] Alejandro Nocua, Florent Bruguier, Gilles Sassatelli, and Abdoulaye Gamatie. Elasticsimmate : A fast and accurate gem5 trace-driven simulator for multicore systems. In *2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8, 2017. doi:10.1109/ReCoSoC.2017.8016146.

[120] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier. Runtime monitoring of software energy hotspots. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 160–169, Sep. 2012. doi:10.1145/2351676.2351699.

[121] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. A review of energy measurement approaches. *ACM SIGOPS Operating Systems Review*, 47(3) :42–49, November 2013. URL : http://doi.acm.org/10.1145/2553070.2553077, doi:10.1145/2553070.2553077.

[122] Marcelo Novaes, Vinicius Petrucci, Abdoulaye Gamatié, and Fernando Magno Quintão Pereira. Compiler-assisted adaptive program scheduling in big.LITTLE systems. In *PPoPP : Principles and Practice of Parallel Programming*, pages 429–430, Washington, United States, February 2019. ACM Press. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-02100287, doi:10.1145/3293883.3301493.

[123] E. Ofori-Attah and M. O. Agyeman. A survey of recent contributions on low power noc architectures. In *2017 Computing Conference*, pages 1086–1090, 2017. doi:10.1109/SAI.2017.8252226.

[124] U.Y. Ogras and R. Marculescu. "it's a small world after all" : Noc performance optimization via long-range link insertion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(7) :693–706, 2006. doi:10.1109/TVLSI.2006.878263.

[125] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Computing Surveys*, 46(4) :47 :1–47 :31, March 2014. URL : http://doi.acm.org/10.1145/2532637, doi:10.1145/2532637.

[126] Marta Ortín-Obón, Darío Suárez-Gracia, María Villarroya-Gaudó, Cruz Izu, and Víctor Viñals-Yúfera. Analysis of network-on-chip topologies for cost-efficient chip multiprocessors. *Microprocessors and Microsystems*, 42 :24–36, 2016. URL : https://www.sciencedirect.com/science/article/pii/S0141933116000077, doi:https://doi.org/10.1016/j.micpro.2016.01.005.

[127] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor : past, present and future. In *Proceedings of Linux Symposium, vol. 2, pp. 223-238*, 2006.

[128] Partha Pratim Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Transactions on Computers*, 54(8) :1025–1040, 2005. doi:10.1109/TC.2005.134.

[129] Emanuele Parisi, Francesco Barchi, Andrea Bartolini, Giuseppe Tagliavini, and Andrea Acquaviva. Source code classification for energy efficiency in parallel ultra low-power microcontrollers. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 878–883. IEEE, 2021. doi:10.23919/DATE51398.2021.9474085.

[130] Jeff Parkhurst, John Darringer, and Bill Grundmann. From single core to multi-core : Preparing for a new exponential. In *2006 IEEE/ACM International Conference on Computer Aided Design*, pages 67–72, 2006. doi:10.1109/ICCAD.2006.320067.

[131] Guillaume Patrigeon. *Systèmes intégrés adaptatifs ultra basse consommation pour l'Internet des Objets*. Theses, Université de Montpellier, July 2020. URL : https://hal-lirmm.ccsd.cnrs.fr/tel-02947275.

[132] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Aridhi. Preesm : A dataflow-based rapid prototyping framework for simplifying multicore dsp programming. In *Education and Research Conference (EDERC), 2014 6th European Embedded Design in*, pages 36–40, Sept 2014. doi:10.1109/EDERC.2014.6924354.

[133] Maxime Pelcat, Jean François Nezan, Jonathan Piat, Jerome Croizer, and Slaheddine Aridhi. A System-Level Architecture Model for Rapid Prototyping of Heterogeneous Multicore Embedded Systems. In *Conference on Design and Architectures for Signal and Image Processing (DASIP) 2009*, page 8 pages, nice, France, September 2009. URL : https://hal.archives-ouvertes.fr/hal-00429397.

[134] Pierre-Yves Péneau, David Novo, Florent Bruguier, Lionel Torres, Gilles Sassatelli, and Abdoulaye Gamatié. Improving the Performance of STT-MRAM LLC through Enhanced Cache Replacement Policy. In *ARCS : Architecture of Computing Systems*, volume LNCS, pages 168–180, Braunschweig, Germany, April 2018. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-01669254, doi:10.1007/978-3-319-77610-1_13.

[135] Fernando Magno Quintão Pereira, Guilherme Vieira Leobas, and Abdoulaye Gamatié. Static prediction of silent stores. *ACM Transactions on Architecture and Code Optimization*, 15(4), November 2018. doi:10.1145/3280848.

[136] Pierre-Yves Péneau, Rabab Bouziane, Abdoulayse Gamatié, Erven Rohou, Florent Bruguier, Gilles Sassatelli, Lionel Torres, and Sophiane Senni. Loop optimization in presence of stt-mram caches : A study of performance-energy tradeoffs. In *2016 26th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 162–169, 2016. doi:10.1109/PATMOS.2016.7833682.

[137] Rezaur Rahman. *Intel Xeon Phi Coprocessor Architecture and Tools : The Guide for Application Developers*. Apress, USA, 1st edition, 2013.

[138] Divyanshu Tiwari Pranjali Singh Ruchi Bhatt Rajesh Singh Thakur, Devid Sahu. Network switching : Traditional techniques and evolution. *International Journal of Engineering and Computer Science*, 5(11), May 2016. URL : http://www.ijecs.in/index.php/ijecs/article/view/3053.

[139] Ville Rantala, Teijo Lehtonen, and Juha Plosila. Network on chip routing algorithms. 12 2008.

[140] N. Rao, A. Ramachandran, and A. Shah. Mlnoc : A machine learning based approach to noc design. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 1–8, 2018. doi:10.1109/CAHPC.2018.8645914.

[141] Basireddy Karunakar Reddy, Amit Kumar Singh, Dwaipayan Biswas, Geoff V. Merrett, and Bashir M. Al-Hashimi. Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores. *IEEE Transactions on Multi-Scale Computing Systems*, 4(3) :369–382, July 2018. doi:10.1109/TMSCS.2017.2755619.

[142] Karunakar Reddy Basireddy, Eduardo Weber Wachter, Bashir M. Al-Hashimi, and Geoff Merrett. Workload-aware runtime energy management for hpc systems. In *International Workshop on Optimization of Energy Efficient HPC & Distributed Systems (OPTIM 2018)*, pages 292–299, May 2018. doi:10.1109/HPCS.2018.00057.

[143] M. F. Reza, T. T. Le, B. De, M. Bayoumi, and D. Zhao. Neuro-noc : Energy optimization in heterogeneous many-core noc using neural networks in dark silicon era. In *2018 IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, pages 1–5, 2018. doi:10.1109/ISCAS.2018.8351580.

[144] Md Farhadur Reza. Reinforcement learning based dynamic link configuration for energy-efficient noc. In *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 468–473, 2020. doi:10.1109/MWSCAS48704.2020.9184490.

[145] Dhruva R. Rinku and M. Asharani. Reinforcement learning based multi core scheduling (rlbmcs) for real time systems. *International Journal of Electrical and Computer Engineering*, 10 :1805–1813, 2020. doi:10.11591/ijece.v10i2.pp1805-1813.

[146] Barry Rountree, Dong H. Ahn, Bronis R. de Supinski, David K. Lowenthal, and Martin Schulz. Beyond dvfs : A first look at performance under a hardware-enforced power bound. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 947–953, 2012. doi:10.1109/IPDPSW.2012.116.

[147] Karl Rupp. Microprocessor trend data. https://github.com/karlrupp/microprocessor-trend-data, 2020.

[148] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet : Leveraging graph neural networks for network modeling and optimization in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10) :2260–2270, 2020. doi:10.1109/JSAC.2020.3000405.

[149] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016. arXiv:1606.03498.

[150] José Carlos Sancho, Antonio Robles, and José Duato. A new methodology to compute deadlock-free routing tables for irregular networks. In Babak Falsafi and Mario Lauria, editors, *Network-Based Parallel Computing. Communication, Architecture, and Applications*, pages 45–60. Springer Berlin Heidelberg, 2000.

[151] Robert Schone, Thomas Ilsche, Mario Bielert, Andreas Gocht, and Daniel Hackenberg. Energy efficiency features of the intel skylake-sp processor and their impact on performance. *2019 International Conference on High Performance Computing & Simulation (HPCS)*, Jul 2019. URL : http://dx.doi.org/10.1109/HPCS48598.2019.9188239, doi:10.1109/hpcs48598.2019.9188239.

[152] Robert Schöne, Daniel Molka, and Michael Werner. Wake-up latencies for processor idle states on current x86 processors. *Computer Science - Research and Development*, 30, 05 2014. doi:10.1007/s00450-014-0270-z.

[153] Sophiane Senni, Thibaud Delobelle, Odilia Coi, Pierre-Yves Peneau, Lionel Torres, Abdoulaye Gamatie, Pascal Benoit, and Gilles Sassatelli. Embedded systems to high performance computing using stt-mram. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 536–541, 2017. doi:10.23919/DATE.2017.7927046.

[154] Sophiane Senni, Lionel Torres, Gilles Sassatelli, Abdoulaye Gamatie, and Bruno Mussard. Emerging non-volatile memory technologies exploration flow for processor architecture. In *2015 IEEE Computer Society Annual Symposium on VLSI*, pages 460–460, 2015. doi:10.1109/ISVLSI.2015.126.

[155] Rishad A. Shafik, Sheng Yang, Anup Das, Luis A. Maeda-Nunez, Geoff V. Merrett, and Bashir M. Al-Hashimi. Learning transfer-based adaptive energy minimization in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(6) :877–890, June 2016. doi:10.1109/TCAD.2015.2481867.

[156] Hao Shen, Jun Lu, and Qinru Qiu. Learning based dvfs for simultaneous temperature, performance and energy management. In *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, pages 747–754, 2012. doi:10.1109/ISQED.2012.6187575.

[157] Sameer S. Shende and Allen D. Malony. The tau parallel performance system. *International Journal of High Performance Computing Applications (IJHPCA)*, 20(2) :287–311, May 2006. URL : http://dx.doi.org/10.1177/1094342006064482, doi:10.1177/1094342006064482.

[158] Shuran Sheng, Peng Chen, Zhimin Chen, Lenan Wu, and Yuxuan Yao. Deep reinforcement learning-based task scheduling in iot edge computing. *Sensors*, 21(5), 2021. URL : https://www.mdpi.com/1424-8220/21/5/1666, doi:10.3390/s21051666.

[159] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel,

and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676) :354–359, 2017. doi:10.1038/nature24270.

[160] Amit Kumar Singh, Somdip Dey, Karunakar Reddy Basireddy, Klaus McDonald-Maier, Geoff Merrett, and Bashir Al-Hashimi. Dynamic energy and thermal management of multi-core mobile platforms : a survey. *IEEE Design and Test*, March 2020. URL : https://eprints.soton.ac.uk/439000/.

[161] Nripendra Kumar Singh and Khalid Raza. Medical image generation using generative adversarial networks, 2020. arXiv:2005.10687.

[162] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, Jae-Wook Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The raw microprocessor : a computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2) :25–35, 2002. doi:10.1109/MM.2002.997877.

[163] S. Umamaheswari, J. Rajapaul Perinbam, K. Monisha, and J. Jahir Ali. Comparing the performance parameters of network on chip with regular and irregular topologies. In David C. Wyld, Michal Wozniak, Nabendu Chaki, Natarajan Meghanathan, and Dhinaharan Nagamalai, editors, *Trends in Network and Communications*, pages 177–186, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-22543-7_18.

[164] A. Varga. Using the omnet++ discrete event simulation system in education. *IEEE Transactions on Education*, 42(4) :11 pp.–, 1999. doi:10.1109/13.804564.

[165] Balaji Venu. Multi-core processors - an overview, 2011. arXiv:1110.3535.

[166] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett. Accurate and stable run-time power modeling for mobile and embedded cpus. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(1) :106–119, Jan 2017. doi:10.1109/TCAD.2016.2562920.

[167] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 341–349. Curran Associates, Inc., 2012. URL : http://papers.nips.cc/paper/4686-image-denoising-and-inpainting-with-deep-neural-networks.pdf.

[168] Gen Xu, Huda Ibeid, Xin Jiang, Vjekoslav Svilan, and Zhaojuan Bian. Simulation-based performance prediction of hpc applications : A case study of hpl, 2020. arXiv:2011.02617.

[169] Thomas Canhao Xu, Ville Leppänen, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen. Pdnoc : Partially diagonal network-on-chip for high efficiency multicore systems. *Concurr. Comput. : Pract. Exper.*, 27(4) :1054–1067, March 2015. doi:10.1002/cpe.3364.

[170] Rong Ye and Qiang Xu. Learning-based power management for multi-core processors via idle period manipulation. In *17th Asia and South Pacific Design Automation Conference*, pages 115–120, Jan 2012. doi:10.1109/ASPDAC.2012.6164929.

[171] Jie Yin. An empirical study on the noc architecture based on bidirectional ring and mesh topologies. 2016.

[172] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn : Generating realistic graphs with deep auto-regressive models, 2018. arXiv:1802.08773.

[173] Kisoo Yu, Donghee Han, Changhwan Youn, Seungkon Hwang, and Jaechul Lee. Power-aware task scheduling for big.little mobile processor. In *2013 International SoC Design Conference (ISOCC)*, pages 208–212, 2013. doi:10.1109/ISOCC.2013.6864009.

[174] Hui Zhao, Mahmut Kandemir, Wei Ding, and Mary Jane Irwin. Exploring heterogeneous noc design space. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 787–793, 2011. doi:10.1109/ICCAD.2011.6105419.

[175] Zhiming Zhou, Jian Shen, Yuxuan Song, Weinan Zhang, and Yong Yu. Towards efficient and unbiased implementation of lipschitz continuity in gans, 2019. arXiv:1904.01184.

[176] E. Zitzler, M. Laumanns, and L. Thiele. Spea2 : Improving the strength pareto evolutionary algorithm. 2001.