POLYPUBLIE

Architecture matérielle logicielle pour l'exécution à latence réduite d'applications de télécommunications émergentes sur centre de données

Michel Gémieux

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

Date	2020
Type	Mémoire ou thèse / Dissertation or Thesis Thèse de doctorat/Ph.D. thesis
URL de PolyPublie PolyPublie URL	https://publications.polymtl.ca/5264/
Conditions d'utilisation Terms of use	© Tous droits réservés / All rights reserved
Référence Citation	Gémieux, M. (2020). Architecture matérielle logicielle pour l'exécution à latence réduite d'applications de télécommunications émergentes sur centre de données [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/5264/

Détails de la thèse
Thesis details

Directeurs de recherche Advisors	Yvon Savaria, Guchuan Zhu, & Jean Pierre David
Programme Program	génie électrique

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal
https://publications.polymtl.ca

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Architecture matérielle logicielle pour l'exécution à latence réduite d'applications de télécommunications émergentes sur centre de données

MICHEL GÉMIEUX
Département de génie électrique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie électrique

Avril 2020

© Michel Gémieux, 2020.

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Architecture matérielle logicielle pour l'exécution à latence réduite d'applications de télécommunications émergentes sur centre de données

présentée par **Michel GÉMIEUX**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Richard GOURDEAU, président
Yvon SAVARIA, membre et directeur de recherche
Guchuan ZHU, membre et codirecteur de recherche
Jean-Pierre DAVID, membre et codirecteur de recherche
Guy BOIS, membre
Bertrand GRANADO, membre externe

DÉDICACE

À tous ceux qui n'ont pas arrêté de me dire :
"Tu es trop proche de la fin pour ne pas finir..."
Merci, vous aviez tous raison...

REMERCIEMENTS

J'aimerais commencer par remercier les Professeurs Yvon Savaria, Guchuan Zhu et Jean-Pierre David qui m'ont accepté sous leur direction pour mener ce projet doctoral dans son entièreté. Je leur serais éternellement reconnaissant de m'avoir permis d'adhérer à un projet avec un partenaire industriel menant des projets de recherche à la pointe de la technologie. Un projet à travers lequel j'ai développé des compétences qui m'ont transformé en une ressource en demande dans l'industrie et qui m'ont valu une place dans une startup à la fine pointe de la technologie avec énormément de potentiel. Je voudrai aussi remercier Patrice Plante et Sébastien Regimbal avec qui ce projet a commencé durant à la maitrise et qui se concrétise en un travail doctoral. Un merci particulier à Richard Goudreau, Guy Bois et Bertrand Granado d'avoir accepté de donner de leurs temps afin de faire partie du jury qui évaluera le travail effectué tout au long de mon doctorat. Puis je tiens à remercier les personnes proches de moi, que ce soit du domaine personnel que professionnel, m'ayant soutenue à travers ce merveilleux, mais aussi tumultueux périple. Particulièrement à :

— Mes collègues et ami(e)s de laboratoire : Mickael Fiorentino, Erika Jolicoeur-Miller et Omar Al Terkawi.
— L'ensemble de l'administration du GRM et de Polytechnique, toujours prêt à aider les étudiants au meilleur de leurs habilités.
— L'ensemble de l'équipe technique du GRM, mais surtout à Rejean Lepage, un collègue et ami qui m'a supporté dans les bons et mauvais moments durant tout mon parcours à Polytechnique.
— Ma famille proche et lointaine ainsi que mes amis, pour leur soutien constant.

J'aimerais terminer en remerciant le conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), Hydro-Quebec, et la Collectivité Territoriale de Martinique (CTM) pour le support financier, ainsi que Huawei Technologie Canada pour le support technique et financier.

RÉSUMÉ

L'industrie des technologies de l'information et des communications fait face à une demande croissante de services sans fil et Internet omniprésents. Cette demande est alimentée par une explosion du nombre d'appareils mobiles riches en multimédia. Il a été estimé qu'à partir de cette année, 2020, le volume de trafic de données mobiles doublera chaque année pour plusieurs années. En conséquence, il en résulte une augmentation significative des dépenses en capital pour les systèmes construits sur les technologies actuelles de réseau d'accès radio qui sont essentiellement basées sur des architectures avec une structure fixe utilisant des plates-formes propriétaires et des mécanismes de contrôle et de gestion de réseau distribués. D'autre part, pour garantir la qualité de service requise, les sous-systèmes sont dimensionnés en fonction des demandes de pointe. Par conséquent, l'extension du réseau aura un impact considérable sur les dépenses d'exploitation. La recherche proposée vise à développer une architecture matérielle et logicielle adaptée à une grappe d'unités de traitement virtualisée pour les signaux en bande de base d'accès radio en nuagique. Ce type d'architecture devra prendre en charge le traitement en temps réel avec des processeurs généralistes sur une plateforme hétérogène. Cela soulève deux défis principaux : la planification des tâches en temps réel et leur exécution d'une manière plus déterministe par rapport aux plates-formes généralistes existantes. Ainsi, les mécanismes d'allocation et de gestion des ressources dans les grappes informatiques doivent être revus. Le deuxième défi est d'obtenir un comportement à faible variance qui implique deux préoccupations majeures : le temps de calcul et le délai de communication. Essentiellement, la variation du temps de calcul est inhérente à tous les processeurs généralistes. Néanmoins, l'infrastructure de communication des grappes informatiques existantes ne fournit aucun soutien pour les communications à faible variance. La recherche proposée est divisée en deux principaux sujets :

— Le calcul dynamique, l'allocation et la gestion des ressources réseau dans une grappe informatique (hétérogène) : les algorithmes d'allocation dynamique des ressources et de planification des tâches en temps réel formeront la fonctionnalité de base prise en charge par le plan de contrôle. Afin de répondre aux fortes contraintes en temps réel de cette classe d'applications, une implémentation matérielle parallèle basée sur circuit logique programmable (FPGA) du plan de contrôle est proposée.

— La prise en charge des calculs et communications à faible variance avec un plan de contrôle implémenté en matériel : Data Plane Development Kit (DPDK) peut être combinée avec la technologie Quick Path Interconnect (QPI) pour fournir une pile de protocoles reposant sur des liaisons physiques haute performance conçues pour des

communications quasi déterministes. Nous montrons qu'en combinant une implémentation matérielle efficace avec la technologie QPI et la bibliothèque DPDK, il existe un fort potentiel pour réaliser des plans de contrôle haute performance.

Afin de résoudre cette problématique, nous découpons la gestion des ressources en trois niveaux, l'ordonancement qui représente la planification des tâches sur une unité de calcul (L1), la gestion des ressources (L2) qui gère les charges de travail à acheminer sur plusieurs noeuds d'unités de calcul, puis la consolidation des ressources (L3) qui s'assure de la gestion de la consommation d'énergie d'une grappe à travers le temps. Cette thèse se concentre et contribue sur les axes suivants :

- Équilibrage de la charge de travail et ordonnancement dynamiques de tâches, où nous démontrons que notre plateforme et sa bibliothèque générique "3D Virtual Fabric Processors" (XDVFP) améliorent l'utilisation des unités de calcul de 80.5 à 99.6%.
- Calculs et communications à faible variance avec un plan de contrôle implémenté en matériel, dans lequel nous démontrons que nous réduisons la complexité de notre moteur de calcul d'allocation de tâche (L2) à une complexité $O(1)$ grâce à l'utilisation d'un accélérateur matériel.
- Tâches à faible variance d'exécution sur un processeur multicœur à l'aide d'un plan de contrôle matériel, où nous utilisons ce même plan de contrôle énoncé plus haut, ce qui permet l'allocation de tâches jusqu'à 760 fois plus vite avec un haut déterminisme au sein de XDVFP.
- Caractérisation et validation du système, où nous explorons les capacités d'une nouvelle plateforme et proposons un ensemble de conseils aux futurs utilisateurs et concepteurs.
- Interconnexion à faible latence et à faible variance pour une grappe informatique en temps réel, où nous proposons et implémentons un nouveau type de communication en utilisant le dernier niveau de cache d'un processeur générique à partir d'un accélérateur matériel (FPGA).

Ce projet de thèse a donc pour principales contributions de :

- Démontrer qu'il est possible de virtualiser un protocole de télécommunication émergent en utilisant que des produits commercialement disponibles, tout en réduisant les coûts.
- Créer une plateforme hétérogène pouvant rencontrer les requis de basse latence et haut débit afin d'assurer la pérénnité du système pour des futures normes de communication.
- Proposer et démonter qu'un nouveau système de communication à faible variance, peu latent à haute bande passante entre les accélérateurs et les processeurs est possible et surtout viable.

ABSTRACT

The Information and Communications Technology industry is facing an increasing demand for ubiquitous wireless and Internet services introduced by an explosion of multimedia-rich mobile devices. It is estimated that starting this year, 2020, the volume of mobile data traffics will double every year. Consequently, it results in significant increases of capital expenditures for systems built on the current Radio Access Network technologies, which are essentially based on architectures with a fixed structure (not reconfigurable) using proprietary platforms with distributed network control and management mechanisms. To ensure the required quality of service, subsystems are dimensioned with respect to the peak demands. Therefore, network expansion will considerably impact on operating expenditures. This thesis aims at developing an architecture at both hardware and software levels suitable for a virtualized Baseband Processing Unit pool in Cloud Radio Acces Network in order to support real-time processing in a General Purpose Processor based platform. This raises two main challenges: scheduling tasks in real-time and executing them in a manner that is reduces variance compared to the existing General Purpose Processor based platforms. Real-time tasks from radio air interface in the Cloud Radio Access Network must be scheduled at a finer grain and must be completed within a given timeslot. Thus, mechanisms for resource allocation and management in computing clusters must be revisited. The second challenge is obtaining a behavior with reduced variability that involves two major concerns: computing time and communication delay. Nevertheless, the communication infrastructure of existing computing clusters does not provide any support for low variance communications. The proposed research is divided into the following main subjects:

— Adaptive computing and network resource allocation and management in (heterogeneous) computing clusters: The algorithms for dynamic resources allocation and real-time task scheduling will form the core functionality that the control plane will support. In order to meet the hard real-time constraints of that class of applications, a parallel Field Programable Gate Array based hardware implementation of the control plane is proposed.

— Supporting low variance computations and communications with a hardware implemented control plane: Intel's Data Plane Development Kit can be combined with its Quick Path Interconnect technology to provide a protocol stack sitting over high performance physical links designed for low variance communications. A notable feature is the availability of a hardware core for the implementation of Quick Path Interconnect and its high performance differential Inputs and Outputs (IOs). By combining

an effective hardware implementation with the Quick Path Interconnect technology (core, physical link, and associated protocol stack) and the Data Plane Development Kit library, there is a strong potential to realize high performance control planes.

In order to solve this problem, we divide the ressource management application into three levels, the scheduling which represents the scheduling of tasks on a compute unit (L1), the load balancing of ressources (L2) which manages the workloads to be routed to one or several nodes of compute units, then ressource consolidation (L3) which manages the energy consumption of a cluster over time. This thesis focuses and contributes on the following topics:

— Dynamic workload balancing and task scheduling, where we demonstrate that our platform and its associated generic library "3D Virtual Processors " (XDVFP) improve the use of computing units by more than 200%.

— Low variance computation and communication enabled by a hardware implemented control plane, in which we demonstrate that we reduce the complexity of our load balancer's task cost engine (L2) to $O(1)$.

— Low variance tasks execution on a multicore processor using a hardware implemented control plane, which allows tasks to be runnning up to 760 times faster with high determinism within XDVFP.

— Sytem characterization and validation, where we explore the capabilities of a new platform and offer insight to future users and designers.

— Low latency and low variance interconnection for real-time computing in a cluster, where we offer and implement a new type of communication using the last level of cache of a generic processor from a hardware accelerator (FPGA).

The main contributions of this thesis are to:

— Prove and demonstrate that it is possible to virtualize an emerging telecommunication protocol using only commercially available products, while reducing costs.

— Create a heterogeneous platform that can meet the requirements of low latency and high speed to ensure the sustainability of a system for future communication standards.

— Propose and demonstrate that a new low variance, low-latency high-bandwidth communication system between accelerators and processors is possible and also viable.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE DES MATIÈRES ix

LISTE DES TABLEAUX xii

LISTE DES FIGURES xiii

LISTE DES SIGLES ET ABRÉVIATIONS xiv

CHAPITRE 1 INTRODUCTION 1
1.1 Cadre théorique 2
1.1.1 *Runtime* 3
1.1.2 Grappe de calcul 3
1.1.3 Allocation de ressources 4
1.2 Problématique et questions de recherche 5
1.3 Objectifs 6
1.4 Contributions 6
1.5 Structure de la thèse 7

CHAPITRE 2 ACCÉLÉRATEURS ET ALLOCATION DE RESSOURCE DANS LE DOMAINE DES TÉLÉCOMMUNICATIONS ÉMERGENTES 9
2.1 Contexte général du projet de recherche 9
2.2 Cloud Radio Access Networks 9
2.3 Accélérateurs en informatique reconfigurable haute performance (HPRC) . . 11
2.3.1 Le paradigme HPC 11
2.3.2 Pertinence de l'utilisation de FPGA dans HPC 13
2.3.3 Accélération d'exécution logicielle 14
2.3.4 L'allocation des ressources 19
2.4 Sommaire de la revue de littérature 20

CHAPITRE 3 DÉMARCHE, HYPOTHÈSES DE RECHERCHE ET OBJECTIFS . 21
3.1 Partenariat Polytechnique-Industrie 21
3.2 Méthodologie et axes de recherche 21
3.3 Hypothèses de recherches 24
3.3.1 Thèmes principaux 24
3.3.2 Thèmes complémentaires 27
3.4 Expérimentation 28

CHAPITRE 4 XDVFP : UNE PLATEFORME POUR LA GESTION DE RESSOURCES RÉSEAU DYNAMIQUE BASÉE SUR DPDK 29
4.1 Plateforme XDVFP 29
4.1.1 Gestion et allocation de ressources au niveau des cabinets de serveurs 29
4.1.2 Architecture générale 30
4.1.3 Modèle d'allocation et gestions de ressources sur XDVFP 31
4.2 XDVFP sur DPDK 34
4.2.1 Bibliothèques de programmation parallèle 34
4.2.2 Modèle de programmation : StreamIt sur DPDK 35
4.2.3 Une application exécutée sur XDVFP 38
4.3 Résultats 41
4.4 Discussion et améliorations possibles 44

CHAPITRE 5 ARTICLE 1 : A CACHE-COHERENT HETEROGENEOUS ARCHITECTURE FOR LOW LATENCY REAL TIME APPLICATIONS 47
5.1 Abstract 47
5.2 Introduction 48
5.3 Related Works 50
5.3.1 FPGAs in HPCs 50
5.3.2 Software runtime 51
5.4 Context and Proposed Architecture 52
5.4.1 Overall System Architecture 52
5.4.2 Proposed Hardware Architecture - The CCHA 55
5.5 A Case Study 57
5.5.1 Max-Min Algorithm 57
5.5.2 Software Tests 57
5.5.3 FPGA Max-Min Implementation 59
5.6 Results and Discussion 61
5.6.1 Resource Utilization 61

5.6.2 Latency Analysis 63
5.7 Conclusion and Future Work 64
5.8 Acknowledgements 65

CHAPITRE 6 ARTICLE 2 : A HYBRID ARCHITECTURE WITH LOW LATENCY INTERFACES ENABLING DYNAMIC CACHE MANAGEMENT 66
6.1 Abstract 66
6.2 Introduction 67
6.3 Background 69
6.3.1 Microarchitectures 69
6.3.2 Real-Time Resource Allocation 72
6.4 Characterization of Low Latency Interfaces on a Hybrid CPU-FPGA Architecture 72
6.4.1 Hardware Configuration 72
6.4.2 Characterization Approach of Low Latency Interfaces 73
6.5 Performance Improvements with Queue-Based Cache Management on an MCP 76
6.6 Validation of Low Latency Communication and Case Study of Cache Management 79
6.6.1 Characterization Results of Low Latency Interfaces 79
6.6.2 A Case Study - Merkle Tree Acceleration 85
6.6.3 Implementation of Merkle Tree and Experiment Results 86
6.7 Conclusion and Future Work 90

CHAPITRE 7 DISCUSSION GÉNÉRALE 92
7.1 Synthèse des travaux 92
7.2 Impact de la recherche 94
7.3 Limitations de la solution proposée 95

CHAPITRE 8 CONCLUSION ET RECOMMANDATIONS 97
8.1 Conclusion 97
8.2 Améliorations futures 97
8.2.1 Interconnexion réseau 98
8.2.2 Consolidation dynamique des ressources 98

RÉFÉRENCES 99

LISTE DES TABLEAUX

Tableau 2.1	Berkeley's 13 dwarfs	12
Tableau 4.1	Résultats de XDVFP avec des systèmes d'Allocation de ressource différents	42
Tableau 5.1	5G expected requirements	49
Tableau 5.2	HRA Area report	61
Tableau 6.1	Communication Interfaces	72
Tableau 6.2	Useful AALSDK API Functions	75
Tableau 6.3	MCP access latency	79
Tableau 6.4	Typical Memory Hierarchy Access times	89

LISTE DES FIGURES

Figure 1.1	Architecture système vue à haut niveau	3
Figure 2.1	Exemples de graphes de flot de traitement parallèles et séquentiels	15
Figure 2.2	Pipelinage logiciel à grosse granularité	16
Figure 3.1	Architecture système multi niveaux	22
Figure 4.1	Architecture système de XDVFP	31
Figure 4.2	Moteur d'allocation de ressources	33
Figure 4.3	Exécution de tâche + données	36
Figure 4.4	Pipeline logiciel à granularité élevée	37
Figure 4.5	Ordonnancement sur 4 coeurs	37
Figure 4.6	Partitionnement des tâches de liaison descendante	41
Figure 4.7	Trace de l'application LTE utilisant XDVFP sur un noeud	43
Figure 4.8	Moteur d'allocation de ressources accéléré	45
Figure 5.1	CCHA Platform Architecture	53
Figure 5.2	CCHA Platform Picture	54
Figure 5.3	Proposed Architecture - Block diagram	55
Figure 5.4	Max-Min block architecture.	59
Figure 5.5	Simulation Results	62
Figure 5.6	Implementation Results	63
Figure 6.1	Accelerator Card	67
Figure 6.2	Socketed FPGA	67
Figure 6.3	MCP	67
Figure 6.4	Accelerator Architectures	67
Figure 6.5	QPI and PCIe connections between CPUs and FPGA.	73
Figure 6.6	The loop back application	74
Figure 6.7	Proposed hybrid architecture composed of CPUs and an FPGA.	77
Figure 6.8	Queue-based cache management.	78
Figure 6.9	Cold cache characterization results	81
Figure 6.10	Warmed cache characterization results	82
Figure 6.11	Merkle Tree in Blockchain applications.	85

LISTE DES SIGLES ET ABRÉVIATIONS

IETF	Internet Engineering Task Force
OSI	Open Systems Interconnection
ACK	Acknowledge
ASIC	Application Specific Integrated Circuit
BBU	Base Band Unit
BRAM	Bloc RAM
BS	Base Station
CAPEX	Capital Expenses
CAPI	Coherent Accelerator Processor Interface
CCI	Cache Coherent Interface
CPRI	Common Public Radio Interface
CPU	Central Processing Unit
CRAN	Cloud Radio Access Network
DAG	Directed Acyclic Graph
DDR	Double Data Rate
DPDK	Data Plane Development Kit
eNodeB	Evolved NodeB
FDMA	Frequency Division Multiple Access
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GDDR	Graphics Double Data Rate
GPP	General Purpose Processor
GSM	Global System/Standard Mobile
HPC	High Performance Computing
IP	Intellectual Property
LLC	Last Level of Cache
LTE	Long Term Evolution
MAC	Media Access Control
MIC	Many Integrated Core
MIMO	Multiple Input, Multiple Output
MKL	Math Kernel Library
NFV	Network Function Virtualization
NIC	Network Interface Controller

OBSAI	Open Base Station Architecture Initiative
OFDM	Orthogonal Frequency-Division Multiplexing
OPEX	Operation Expenses
PCIe	Peripheral Component Interconnect Express
PHY	Physical Layer
QPI	Quick Path Interconnect
RAN	Radio Access Network
RRH	Remote Radio Head
RRU	Remote Radio Unit
SDN	Software Defined Network
SIMD	Single Instruction on Multiple Data
TBB	Threading Building Blocks
WAN	Wide Area Network
WCET	Worst Case Execution Time

CHAPITRE 1 INTRODUCTION

L'industrie des technologies de l'information et des communications (TIC) fait face à une demande croissante de services sans-fil et internet omniprésents, provoquée par l'explosion du nombre d'appareils mobiles intelligents. Il a été estimé qu'à partir de cette année, 2020, le volume des trafics de données mobiles doublera chaque année pour plusieurs années à venir [1]. Pour augmenter la capacité globale du système, il est nécessaire de diviser le réseau avec des cellules plus petites, ce qui nécessite l'installation de nouveaux sites et équipements et, par conséquent, entraîne une augmentation significative des dépenses en capital (CAPEX) pour la mise en oeuvre des réseaux d'accès radio (RAN). Les technologies sous-jacentes reposent essentiellement sur des architectures à structure fixe utilisant des plateformes propriétaires et des mécanismes de contrôle et de gestion des réseaux distribués. D'autre part, pour garantir la qualité de service (QoS) requise, les sous-systèmes (stations de base) sont dimensionnés en fonction des demandes. Par conséquent, l'extension du réseau aura un impact considérable sur les dépenses d'exploitation (OPEX). De plus, la consommation d'énergie électrique représente une part importante des coûts d'exploitation [2]. Par conséquent, l'expansion du réseau dans une architecture RAN traditionnelle avec des ressources sur l'approvisionnement est un fardeau pour les systèmes énergétiques, ce qui augmentera considérablement l'empreinte carbone des TIC. De nos jours, l'industrie des TIC est sous une pression énorme pour développer de nouvelles technologies RAN qui peuvent répondre au besoin d'une densité de capacité sans cesse croissante, tout en réduisant les CAPEX, OPEX, ainsi que l'impact environnemental. Pour relever ce défi, un grand effort a été fait par l'industrie ces dernières années et il est reconnu que l'une des solutions viables consiste à permettre une intégration plus étroite des infrastructures informatiques et des systèmes de télécommunications, traditionnellement séparés, pour former un réseau unifié [1]. Conformément à cette vision, le China Mobile Research Institute a introduit le concept de réseaux d'accès radio basé sur le nuage (C-RAN), renforcé par le traitement centralisé du signal et des données, la radio coopérative, le nuage et une infrastructure propre (plus verte) [3]. Contrairement aux RAN conventionnels, un C-RAN dissocie l'unité de traitement en bande de base (BBU) de la radiocommande (RRH), permettant une opération centralisée des BBU et un déploiement évolutif des RRH sous forme de petites cellules. L'électronique des stations de base numérique conventionnelle peut ensuite être déplacée vers des centres de données hébergeant des processeurs à usage général (GPP) de haute performance, c'est-à-dire le bassin BBU. Grâce à la virtualisation et au contrôle (logique) du réseau centralisé, le C-RAN permet le calcul dynamique et la gestion des ressources réseau en matière de répartition et de gestion active de la consommation

d'énergie, conduisant à des solutions économes en énergie et rentables, avec une utilisation globale améliorée du réseau. En tant que grand fournisseur mondial de solutions TIC, le partenaire industriel du projet investit dans le développement de technologies liées au C-RAN visant à ouvrir la voie vers les réseaux mobiles de prochaine génération (5G). Néanmoins, bien que certains efforts aient été déployés par les secteurs universitaire et industriel pour faire avancer le paradigme C-RAN [4] [5], cette technologie est encore à ses balbutiements et il existe de nombreux problèmes en suspens devant être résolu avant que le réseau de mobilité 5G ne devienne réalité. En ce qui concerne cet ultime objectif, l'une des priorités du partenaire industriel vise le développement de stations de base qui sont déployables avec les technologies actuellement disponibles.

La recherche effectuée dans cette thèse explore des solutions pour l'informatique dynamique et l'allocation des ressources réseau, ainsi que pour la gestion des bassins BBU C-RAN fonctionnant sur une grappe de calcul informatique dans des infrastructures telles que les centres de données. Les solutions explorées comprennent des implémentations exploitant des circuits intégrés composés d'un réseau de cellules programmables (FPGA) pour l'allocation dynamique des ressources et des algorithmes de planification en temps réel. La recherche caractérise également la variabilité et l'amélioration des performances possibles sur un prototype C-RAN assurant la gestion du tampon, le verrouillage de la cache et la gestion des communications entre processeurs, pour permettre la virtualisation de différentes normes de communication sans fil, telles que Long-Term-Evolution (LTE) et ultimement la 5G.

L'objectif principal de cette thèse est de développer des solutions économes en énergie et rentables pour la gestion des ressources informatiques, de stockage et de mise en réseau (allocation des ressources 3D) des grappes informatiques dans le contexte d'applications pour le C-RAN. Les algorithmes, solutions et outils de prototypage développés dans cette thèse sont validés avec la plate-forme *3D Virtual Fabric Processors* développé conjointement avec le partenaire industriel de ce projet, qui a permis une intégration transparente des résultats obtenus avec les processus de développement d'ingénierie de pointe du partenaire industriel. Ce projet a contribué non seulement à la stratégie de développement à long terme du partenaire industriel, mais elle représente également un pas en avant significatif pour l'équipe impliquée dans ce projet en fournissant un contexte à cette recherche.

1.1 Cadre théorique

Afin d'avoir une bonne compréhension du travail effectué, certains concepts doivent être définis. Les concepts élaborés ci-dessous expliquent à haut niveau les thèmes sur lesquels cette thèse s'appuie.

1.1.1 *Runtime*

Pour une meilleure compréhension, nous devons définir le concept de *Runtime* et les systèmes d'exécution. Un système d'exécution est la collection de tous les mécanismes logiciels et matériels qui permettent à un programme de s'exécuter sur une plate-forme donnée. Lorsque l'exécution est mentionnée dans ce projet de thèse, elle fait référence aux systèmes logiciels et matériels, qui servent à la gestion de la mémoire et des ressources, ainsi qu'à l'accélération matérielle pour l'exécution des tâches. Il ne faut pas confondre le système d'exécution proposé avec un environnement d'exécution similaire à JAVA (*Java Runtime Environment*), qui fournit plutôt un espace de développement et d'éxécution.

1.1.2 Grappe de calcul

Afin de comprendre quelle technologie permet les contributions de cette thèse, une description de notre plateforme informatique s'impose. Cette section présente brièvement l'architecture globale d'une grappe de calcul informatique. Comme le montre la Figure 1.1, la grappe est constituée de plusieurs lames de serveur organisées dans une configuration maître-esclave, connectées ensemble avec un commutateur à faible latence et à large bande passante. Une lame sert de nœud maître, tandis que les autres lames esclaves servent de nœuds de calcul. Un tel système pourrait prendre différentes configurations, rendant l'ensemble du système hétérogène.

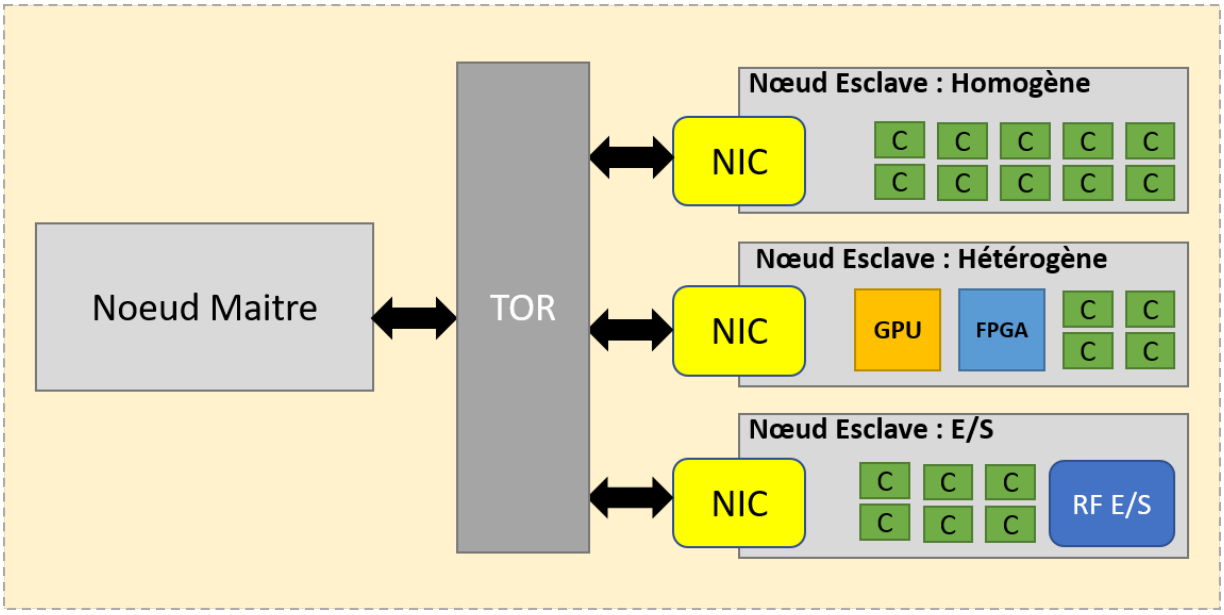


Figure 1.1 Architecture système vue à haut niveau

En raison du contexte d'application, certains esclaves pourraient être dédiés au calcul de l'interface aérienne (*Common Public Radio Interface* pour les applications LTE), tandis que le reste pourrait fournir différents types de ressources de calcul, telles que les unités de traitements graphique à usage général (GPU), les FPGA et différentes saveurs d'unités centrales de traitement x86 (CPU).

1.1.3 Allocation de ressources

L'ordonnancement est la méthode par laquelle on donne accès aux ressources de la machine ainsi qu'aux processus (ensemble de tâches) de calcul d'une application donnée. Le principal but est de pouvoir adéquatement équilibrer l'utilisation des ressources afin que les processus s'effectuent le plus efficacement que possible. De cette façon, un ordonnanceur est le mécanisme qui s'occupe de gérer quel processus peut avoir accès aux ressources de la plateforme et quand. Dans notre cas, nous parlerons d'un ordonnanceur de tâches, aussi nommé "*task scheduler*". Dans le contexte LTE un bon ordonnanceur doit tenir compte des contraintes temps réel, car la norme nous impose une borne stricte sur le temps de traitement par usager de 10ms [6]. Ce dernier doit donc prendre en considération :

— Les échéances pour assurer que chaque processus respecte la plage de temps qui lui est allouée.
— La latence, premièrement le délai d'exécution, c'est-à-dire le temps entre la réception de la requête et sa complétion. Puis le temps de réponse entre l'envoi de la première requête et la réception de la première réponse ("*acknowledge*").
— L'utilisation des CPU, afin ne pas sur-utiliser un processeur, alors que d'autres le sont peu.
— Le débit, qui correspond au nombre d'exécutions de processus par unité de temps.

Afin que l'ordonnanceur puisse gérer les contraintes de temps imposées par le contexte de télécommunication, les tâches seront acheminées avec des descriptifs. Les plus utilisés sont :

— Le temps de relâchement (*Release time*), qui est le temps auquel une tâche est prête à être exécutée.
— Le délai minimum, qui est le minimum de temps permis au début de l'exécution d'une tâche et après sa complétion.
— Le délai maximal, qui est le maximum de temps permis au début de l'exécution d'une tâche et après sa complétion.
— Le WCET (*Worst Case Execution Time*), qui est le plus long temps d'éxécution que peut prendre une tâche.
— Le *Runtime* (Notez que le terme Runtime prend plusieurs sens dans cette thèse. Ces

sens sont clairs par le contexte), qui représente le temps pris pour accomplir une tâche sans interruption.
— Le poids ou priorité, qui est souvent le niveau d'urgence de la tâche.
— La mémoire consommée, cela pourrait être la mémoire nécessaire pour exécuter une tâche ou même les espaces requis pour exécuter une fonction reliée à la tâche.

Une fois toutes ces caractéristiques prises en compte, il faut avoir un ordonnancement adéquat, qui nous permettra d'atteindre la qualité de service voulue. Dans les domaines du "*Cloud Computing*" et du "*High Performance Computing* (HPC)", un ordonnancement efficace des tâches d'un processus est primordial afin d'avoir la plus grande performance possible. Il existe une abondante littérature sur le sujet de l'ordonnancement de tâches, ce que nous pouvons en retirer est qu'il n'existe pas une solution parfaite qui a la capacité de résoudre tous les problèmes. Ainsi, au lieu de s'efforcer à trouver l'ultime solution d'ordonnancement, les concepteurs se fient à des algorithmes ou stratégies, qu'ils choisissent en fonction des caractéristiques de l'application. Il existe de nombreux algorithmes d'ordonnancement disponibles dans la littérature [7].

1.2 Problématique et questions de recherche

La recherche dans ce projet vise à développer une architecture à la fois matérielle et logicielle adaptée à un bassin (*pool*) BBU virtualisé en C-RAN afin de prendre en charge le traitement en temps réel dans une plateforme basée sur GPP. Cela soulève deux défis principaux : la planification des tâches en temps réel et leur exécution d'une manière moins variable par rapport aux plates-formes GPP existantes. La planification habituelle des tâches dans les grappes de calcul ne se fait pas de façon fine et elle n'est en aucun cas spécialisée. D'autre part, les tâches en temps réel à partir de l'interface radioélectrique dans le C-RAN doivent être planifiées à un niveau plus fin et elles doivent être achevées dans un intervalle de temps donné. En outre, la consolidation des tâches (sur les cœurs de CPU) doit être envisagée afin d'optimiser l'utilisation du CPU et de minimiser les frais de communication. Ainsi, les mécanismes d'allocation et de gestion des ressources dans les grappes informatiques doivent être revus. Le deuxième défi est d'obtenir un comportement à faible variance qui implique deux préoccupations majeures : le temps de calcul et le délai de communication. Essentiellement, la variation du temps de calcul est inhérente à tous les GPP. Une étude préliminaire indique que le DPDK [8] semble fournir une infrastructure capable de gérer la variation du temps de calcul à des niveaux acceptables. Néanmoins, l'infrastructure de communication des grappes informatiques existantes ne fournit aucun support pour les communications à faible variance.

1.3 Objectifs

Concrètement, cette thèse vise à développer une architecture matérielle générique pour l'accélération d'applications s'exécutant sur des grappes HPC hétérogènes. L'objectif visé était de réduire la latence du HPC grâce à l'accélération matérielle des systèmes d'allocation et de gestion des ressources informatiques dynamiques à plusieurs échelles de temps. L'une des applications cibles est la plate-forme de calcul pour prendre en charge des normes de communication sans fil telles que LTE et 5G. Une partie essentielle du projet de thèse consiste à développer et caractériser des algorithmes pour répartir les charges de travail sur les lames de serveur de manière équilibrée dans le but de maximiser l'utilisation des processeurs dans la grappe de calcul et de garantir la bande passante pour le transfert de données entre les nœuds de calcul et la mémoire cache réservée pour permettre l'exécution des tâches avec une faible variance. La charge de travail répartie entre les lames de serveur doit ensuite être planifiée à une échelle plus fine afin de minimiser la durée requise pour terminer les exécutions de tâches. Les étapes franchies pour atteindre les objectifs cités ont consisté à :

— Proposer et implémenter un cadre d'allocation des ressources efficace pour les grappes de HPC hétérogènes.
— Proposer et implémenter une architecture matérielle reconfigurable pour l'accélération des services du centre de données.
— Implémenter un allocateur de ressources matérielles multi-domaines (niveaux) pour les grappes hétérogènes.

Les résultats de cette recherche seront précieux pour les praticiens de l'industrie, ainsi que pour les fournisseurs de logiciels associés dans le développement de meilleures pratiques et d'outils pour la gestion des contraintes et la planification prévisionnelle. Les travaux effetués auront un impact sur l'avenir du HPC et, éventuellement, sur la plupart des processeurs à usage général.

1.4 Contributions

L'atteinte des objectifs cités nous amène à revendiquer pour principales contributions :

— Une démonstration qu'il est possible de virtualiser un protocole de télécommunications emergent en n'utilsant que du matériel commercialement disponible (COTS) et ce en réduisant les coûts d'OPEX et CAPEX d'un futur opérateur de télécommunications.
— La création d'une plateforme hétérogène pouvant rencontrer les requis de basse latence et haut débit des normes de communication émergentes.
— La proposition et la démonstration qu'un nouveau système de communication à faible

variance, peu latent à haute bande passante entre accélérateur et processeur est possible et surtout viable.

En plus de ces principales contributions, cette thèse offre aussi une solution viable et expensible à notre partenaire industriel. En effet, la bibliothèque 3D Virtual Fabric Processors (XDVFP) n'est pas seulement un élément essentiel d'une plateforme pour l'implémentation de futures normes de communications, mais elle est aussi une bibliothèque générique permettant l'exécution de tâches avec une faible variabilité et efficace (haute bande passante) sur des plateformes hétérogènes multinoeuds. Nous proposons notre nouvelle approche d'allocation de ressources dynamique dans une grappe de calcul haute performance. Nous nous baserons sur de nouveaux systèmes logiciels couplés avec des améliorations structurelles proposées, menant à la création de notre bibliothèque générique XDVFP. Nous démontrons ensuite que nous pouvons avoir une exécution à faible variance des tâches avec une meilleure utilisation des unités de calcul disponibles, grâce à l'introduction d'accélérateurs matériels. Les résultats de cette contribution ont été publiés dans un article intitulé « A Cache-Coherent Heterogeneous Architecture for Low Latency Real Time Applications ». Puis, nous avons aussi eu l'opportunité d'avoir accès à des plateformes CPU-FPGA d'avant-garde sur lesquelles nous avons pu expérimenter nos résultats de caractérisation et élaborer un ensemble de conseils pour ceux qui voudraient utiliser ou construire ce type de plateforme dans le futur. Nous utilisons ces nouvelles plateformes en socle pour implémenter une nouvelle technique de gestion dynamique et efficace du dernier niveau de cache d'un CPU. Les résultats de la caratérisation et des performances obtenus grâce à notre nouvelle technique de gestion de cache sont exposés dans l'article de revue « A Hybrid Architecture with Low Latency Interfaces Enabling Dynamic Cache Management ». Nous sommes aussi pionniers dans l'exploration et l'élaboration de différentes techniques de communication à travers le dernier niveau de cache d'un CPU à partir d'un accélérateur matériel, rendant cette thèse une des plublications fondatrices d'un nouveau domaine de recherche prometteur.

1.5 Structure de la thèse

Cette thèse est organisée comme suit. Le Chapitre 2 est consacré à l'état de l'art pertinents, incluant les cadres logiciels associés, les techniques et architectures utilisées dans le domaine du HPC appliqué aux télécommunications. Le Chapitre 3 décrit notre méthodologie de recherche et explique le processus de génération de pistes de recherche, d'identification des problèmes, des éléments exploitables, des étapes spécifiques et des résultats éventuels en termes de documents de recherche. Il présente une vue d'ensemble du corps de cette thèse. Nous nous dirigeons ensuite vers le corps central du projet de thèse qui a été présenté comme

un chapitre exploré en profondeur et deux articles.

Nous présentons tout d'abord une architecture logicielle et matérielle, ainsi que les résultats préliminaires de la solution proposés avant l'incorporation d'accélérateurs. Le tout constitue le Chapitre 4 de cette thèse.

Un premier article traite des nouvelles techniques d'accélération d'allocation de ressources sur FPGA dans le cadre de grappe de calcul haute performance pour application nécessitant une basse latence. Cet article est intitulé «*A Cache-Coherent Heterogeneous Architecture for Low Latency Real Time Applications*», exposé dans le cadre du 2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC) et est présenté dans le Chapitre 5 de la thèse.

Notre deuxième article présente une nouvelle technique de gestion dynamique du dernier niveau de cache (LLC) de CPU avec un accélérateur matériel. Cet article est intitulé «*A Hybrid Architecture with Low Latency Interfaces Enabling Dynamic Cache Management*» et il a été présenté dans l'édition du mois d'octobre 2018 de la revue IEEE Access. Cet article est reproduit comme Chapitre 6 dans cette thèse.

Enfin, au Chapitre 7, nous présentons un résumé et une discussion de nos contributions au monde de la recherche, de leur impact sur l'écosystème des HPC et nous formulons des recommandations pour les travaux futurs qui pourraient être menés dans ce domaine.

CHAPITRE 2 ACCÉLÉRATEURS ET ALLOCATION DE RESSOURCE DANS LE DOMAINE DES TÉLÉCOMMUNICATIONS ÉMERGENTES

Ce chapitre servira de fondation sur laquelle se base une bonne partie des concepts utilisés et étudiés dans cette thèse. Il vise à résumer l'état de l'art, ainsi qu'à démontrer le potentiel de recherche de cette thèse.

2.1 Contexte général du projet de recherche

Afin de mieux situer la revue de l'état de l'art, le contexte global du projet de thèse doit être mis en perspective. Ce projet doctoral a commencé au sein d'un projet plus vaste soutenu par un partenaire industriel. La ligne directrice du projet était l'accélération de grappe de calcul pour les technologies émergentes, tel que la 5G. En effet, pour le cas de la 5G, il est estimé que l'architecture soutenant éventuellement le protocole devra avoir 10-100x la bande passante de la technologie précédente, LTE, tout en réduisant de 10x la latence [7]. Une des techniques proposées par l'industrie est connue sous l'appellation Cloud Radio Access Networks (C-RAN), dans le cadre de laquelle une ou un petit ensemble de grappe serviraient de zone de traitement "nuagique" plus proche des utilisateurs pour pallier aux besoins de performances des protocoles. Le principal problème est qu'afin de rencontrer les requis de bande passante, latence et efficacité énergétique dans des petits centres de HPC, les architectures existantes sont inadéquates. Au sein de ce projet, nous visons à introduire une architecture système et une bibliothèque logicielle générique pour satisfaire ces besoins. Nous proposons d'utiliser des accélérateurs au sein de la grappe pour, à la fois augmenter la quantité de traitements possibles tout en diminuant le besoin en énergie. Puis notre bibliothèque logicielle fournira un environnement propice pour l'implémentation de diverses applications nécessitant une accélération. Nous proposons aussi une infrastructure logicielle utilisant les accélérateurs matériels disponibles dans la grappe pour une gestion dynamique des ressources permettant d'améliorer la capacité de calcul tout en réduisant grandement la latence globale de traitement.

2.2 Cloud Radio Access Networks

L'architecture d'un C-RAN se compose de trois éléments clés [5] [6] ; les unités d'accès radio distribuées (RAU) sur le site distant d'une petite cellule ; un bassin de BBU dans un nuage de centre de données, géré par des processeurs généraux à haute performance offrant une

virtualisation en temps réel ; et un réseau de transport optique à large bande passante et à faible latence reliant les BBU et les RAU. La séparation des RAU du traitement en bande de bases permet de migrer le traitement des signaux et des données vers une entité centralisée et, par conséquent, elle embrasse les technologies émergentes des centres de données et du traitement nuagique (*Cloud Computing*) et enfin elle permet des infrastructures plus vertes, une réduction des CAPEX et OPEX, un équilibrage de charge efficient pour soutenir un grand trafic et des services flexibles [9] [10]. Le projet proposé se concentre principalement sur les questions liées aux BBU, en particulier le concept d'Infrastructure-as-a-Service (IaaS) activé par la technologie des centres de données.

L'un des sujets centraux liés aux nuages IaaS est de savoir comment allouer dynamiquement des ressources aux grappes de calcul pour minimiser les coûts d'exploitation tout en respectant les contraintes de QoS. Dans la littérature, l'une des solutions les plus populaires pour trouver le meilleur compromis entre ces deux objectifs contradictoires modélise l'allocation des ressources en tant que problème de bien-être social ("*social welfare*") exprimé par une fonction d'utilité appropriée [11]. Ces dernières années, ce paradigme a été largement adopté dans l'ingénierie du trafic sous le nom de *Network Utility Maximization* (NUM) [12] et la gestion de l'énergie dans le *Smart Grid* [13]. De nombreux modèles et algorithmes ont été développés. Ils peuvent être facilement appliqués aux affectations de ressource dynamiques dans les grappes informatiques avec des adaptations adéquates. Une autre solution viable est l'application de technologies des systèmes de commande, en particulier le *Model Predictive Control (MPC)* [14], ce qui revient à trouver une solution optimale compromettant les performances souhaitées et le coût requis. L'allocation dynamique des ressources peut également être formulée comme un problème de contrôle par rétroaction [15], auquel des outils de la théorie des systèmes de commande peuvent être appliqués à la conception d'algorithmes de commande en temps réel. Néanmoins, une formulation appropriée des problèmes de commande permettant une maximisation de l'utilité conjointe de la qualité de service et des coûts de consommation d'énergie est encore nécessaire [16]. En effet, l'allocation dynamique des ressources à différents niveaux du traitement nuagique est un sujet de recherche très actif dans la littérature [17].

Pour réduire davantage les coûts d'exploitation dus à la consommation d'énergie, les solutions au niveau du cabinet d'équipements informatique consistent à mettre hors tension les serveurs inactifs à l'aide de composants matériels de bas niveau pour prendre en charge une mise à l'échelle des performances proportionnelle à la consommation d'énergie. La réduction du sourcoût introduit par le basculement entre différents états d'alimentation et les ressources physiques (par exemple, les serveurs, les configurations de base, etc.) peut également faire progresser considérablement la gestion des ressources écoénergétiques [18].

L'intérêt de la mise en œuvre de systèmes C-RAN sur les nuages informatiques était clairement compris. Par contre, même si le maximum de puissance de traitement qui est disponible dans un grand centre de données permet en principe des solutions par la force brute, les récentes tentatives industrielles de le faire de façon économiquement acceptables n'ont pas donné les résultats souhaités. Un facteur clé qui a entravé ces initiatives est la latence entre les fils de coopération rencontrés avec les systèmes d'exploitation de centre de données gérés par un Linux typique. Une solution à ce problème a été élaborée par Intel, le fournisseur de la grande majorité des processeurs au coeur des centres de données. Le Data Plane Development Kit (DPDK) [19] a été spécifiquement proposé à cet effet.

2.3 Accélérateurs en informatique reconfigurable haute performance (HPRC)

L'informatique reconfigurable haute performance est considérée comme la prochaine étape du HPC [20]. Actuellement à l'ère de l'informatique hétérogène, l'ajout de matériel reconfigurable [21], similaire aux FPGA, révolutionne la façon dont nous faisons des calculs, avec l'arrivée de "FPGA intégrés" [22]. Les sections suivantes explorent l'état actuel du HPC et l'évolution vers l'informatique hétérogène pour finir avec un argument pour les FPGA dans les environnements HPC.

2.3.1 Le paradigme HPC

Afin de mettre en place correctement cette recherche, nous devons comprendre le but et les principaux algorithmes exécutant une plate-forme HPC donnée. Le calcul haute performance consiste à utiliser plusieurs unités de calcul parallèles, dans le cas de nos serveurs lames, pour exécuter et résoudre des problèmes scientifiques. L'unité informatique peut s'étendre de la petite lame à processeur unique aux serveurs multiprocesseurs haute performance connectés via des liaisons à haut débit. Une grappe HPC typique contient des millions de cœurs actifs, comme le plus grand système connu qui contient 10 649 600 cœurs [23]. Les applications typiques roulant sur des HPC sont la résolution d'équations complexe, la simulation large (génome) et le rendu de calculs hautement techniques. Ces types d'applications à grande échelle mettent l'accent sur l'analyse massive de données [24]. Jusqu'à présent, la plupart des puissants systèmes HPC étaient basés sur un processeur. Cependant, l'analyse massive des données étant la nouvelle application principale des systèmes HPC, l'homogénéité des systèmes est coûteuse pour un propriétaire de centre HPC. En effet, la consommation électrique et la génération de chaleur sont proportionnelles à la vitesse d'horloge des processeurs. Cette combinaison a un impact négatif sur le coût de possession d'un système HPC [25]. Afin d'obtenir un meilleur coût de performance, l'alternative viable aux HPC conventionnels est

la migration vers des architectures hétérogènes de HPC. L'informatique hétérogène n'est pas nouvelle et à de nombreuses architectures différentes. Dans le domaine CPU, vous pouvez trouver une plate-forme multicœur asymétrique [26] avec la même architecture, deux unités de calcul cadencées différentes, elle est appelée architecture Big.LITTLE. L'informatique hétérogène est lorsque plusieurs architectures ou matériels informatiques sont utilisés dans un système donné pour exécuter des applications spécifiques ou très spécifiques. Habituellement, le HPC hétérogène implique l'utilisation d'accélérateurs matériels différents du conventionnel (CPU) utilisés dans les lames de serveur. Cependant, différents accélérateurs ont des forces différentes. Pour pouvoir répondre aux besoins d'un type d'accélérateur spécifique, nous devons mieux définir les applications fonctionnant sur les systèmes HPC. Afin de comparer précisément un système, D. Patterson de l'Université de Berkley a classé les principaux types d'algorithmes qui fonctionnent sur les HPC, il les a nommés, les 13 nains [27]. Le tableau ci-dessous définit chaque nain. Nous observons 4 groupes principaux du Tableau 5.2 :

— calculs arithmétiques complexes combinés à un accès régulier à la mémoire est plus efficace sur les **GPU** ;
— calculs arithmétiques complexes combinés à un accès irrégulier à la mémoire est plus efficace sur les **FPGA** ;
— calculs arithmétiques simple nécessitant une gestion complexe de la mémoire, favorisent les architectures **Multicoeurs** ;
— calculs arithmétiques simple utilisant des opérations matérielles spécifiques, favorise les **FPGA**.

Tableau 2.1 Berkeley's 13 dwarfs

Dwarf	Application	Accelerator
DLA : Dense Linear Algebra	dense matrices	GPU
SLA : Sparse Linear Algebra	sparse matrice	FPGA
SM : Spectral Methods	FFT-based methods	GPU
NBM : N-Body Methods	Particle interactions	GPU
SG : Structured Grids	Fluid dynamics	FPGA
UG : Unstructured Grids	Adaptive mesh FEM	FPGA
MR : Map Reduce	Monte Carlo integration	Multicore
CL : Combinational Logic	Logic gates	FPGA
GT : Graph Traversal	Searching, selection	Multicore
DP : Dynamic Programming	Tower of Hanoi problem	Multicore
BB : Backtrack and Branch-n-Bound	Global optimization	Multicore
GM : Graphical Models	Probabilistic networks	Multicore
FSM : Finite State Machines	TTL counter	FPGA

La principale information à tirer du Tableau 5.2 est que la plupart des applications HPC

impliquent le traitement d'une grande quantité d'informations. La latence semble être une pensée d'après-coup. C'est là que nos recherches revendiquent l'une de ses contributions, à savoir améliorer la latence des HPC hétérogènes, en ce qui a trait à la satisfaction de contraintes de latence et de variabilité réduite d'exécution. Cette proposition vise à résoudre le problème avec l'utilisation de la prise en charge FPGA (*offload*). La section suivante expliquera plus en détail la viabilité du FPGA dans HPC.

2.3.2 Pertinence de l'utilisation de FPGA dans HPC

Nous pouvons déduire du Tableau 5.2 que le FPGA est mieux adapté à l'application de diffusion. Des travaux similaires à Inta et al dans [28] exposent les FPGA comme accélérateurs de fonctions. Dans le cas d'une application ou une fonction donnée s'exécute sur le FPGA et renvoie ses résultats à une unité de traitement principale.

Deux problèmes principaux peuvent être dérivés de l'utilisation des FPGA. Le premier, c'est sa courbe d'apprentissage. Il est bien connu que pour utiliser la majeure partie de la puissance de traitement FPGA exposée par Escobar et al dans [29], le développeur doit également avoir une bonne connaissance de l'architecture matérielle. Pour le rendre facile à utiliser, des cadres et outils de programmation de haut niveau ont été introduits, communément appelés outils HLS [30], [31]. La plupart des HPC contenant des FPGA connectent tous les accélérateurs via PCIe. PCIe est principalement conçu pour favoriser la bande passante par rapport à la latence.

De nombreuses études et systèmes utilisent un protocole de communication basé sur PCIe [32] [33]. L'utilisation d'accélérateurs FPGA via la communication PCIe n'est pas nouvelle dans le domaine HPC [34]. De grands acteurs comme Intel et Microsoft [35] ont réussi à intégrer l'accélération FPGA dans leur flux de traitement pour une analyse massive des données de leur moteur de recherche Bing. Peu d'attention a été accordée aux problèmes de latence dans HPC pour favoriser le traitement massif des données. Nous explorerons le problème de latence dans les applications de flux *streaming* dans la prochaine section de cette revue de la littérature. C'est là que les FPGA à fiches (*in-socket*) entrent en action. Considérés comme la prochaine étape importante du HPRC, comme le montre le professeur Chow dans [36] le professeur Ling dans [37], ils permettent des communications à faible latence entre l'unité de traitement. Certains diront que ce n'est pas nouveau en raison des "processeurs logiciels" déjà disponibles sur les plates-formes FPGA, comme Nios [38], Microblazes [39] et Zynqs avec processeurs ARM [40] "dur". Le problème avec ces implémentations est toujours la difficulté de les intégrer dans une grappe HPC tout en utilisant un protocole à latence plus faible. Pour résoudre ce problème, le FPGA en prise (*in-socket*) a réduit la latence de communication avec

l'utilisation du bus frontal [41]. La communication point à point entre le CPU et le FPGA a permis aux développeurs d'avoir un meilleur accès à la mémoire système [22]. Même si elle était 6 fois moins latente que la communication PCI-X, elle est encore trop élevée pour compenser la virtualisation 5G. Afin de répondre aux exigences de la 5G les implémentations de [41] et [22] manquaient d'une fonctionnalité spécifique, une interface cohérente de cache (CCI).

La première communication FPGA / CPU cohérente en cache a été introduite par Intel, avec le bus Quick Path Interconnect [22]. L'utilisation de ces avancées permet à cette recherche de contribuer au domaine d'étude. Une CCI permet des communications via la mise en cache. La latence des communications serait équivalente au temps requis pour écrire et lire un point de données dans le cache. Dans les fiches (*socket*), les FPGA permettent un traitement à faible latence et à haut débit d'avoir un répartiteur de ressources multiniveaux efficaces. Notre travail étant à la jonction entre deux domaines, nous comparerons les gains d'efficacité au HPC avec les accélérateurs FPGA et les allocateurs de ressources matérielles. Malheureusement, peu de travaux ont été publiés dans notre créneau, en raison de la faible disponibilité de la plate-forme FPGA QPI en fiche. Nous sommes quelques-uns à avoir cette opportunité.

Notre architecture matérielle permettra le déploiement des algorithmes de programmation complexes [42] et d'allocation des ressources. Ce qui accélérera finalement les temps d'exécutions d'un HPC hétérogène. La section suivante explique en détail ce que nous appelons un "*Runtime*".

2.3.3 Accélération d'exécution logicielle

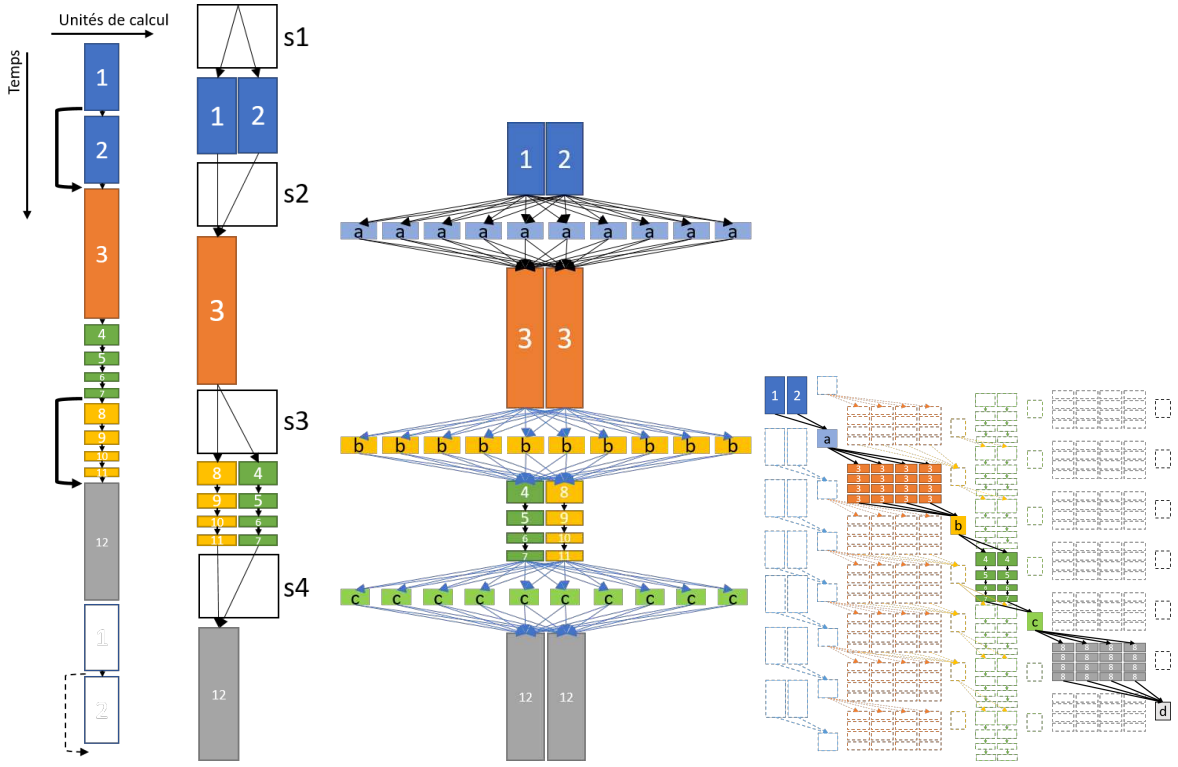
La manière typique d'obtenir un système moins latent consiste à utiliser des "systèmes d'exploitation en temps réel (RTOS)", tels que FreeRTOS [43]. Laplante et al dans [44] expliquent comment ils réduisent la latence induite par la gestion des ressources avec des insertions d'assemblages intelligents au niveau du noyau. Cependant, un problème associé a RTOS est leur rigidité et le manque de support pour les nouvelles bibliothèques de calcul. Les rendant moins adaptés aux nouvelles applications, en particulier les applications de type HPC. En dehors de l'utilisation de RTOS, les moyens courants pour accélérer l'exécution sont les suivants :

- une meilleure gestion de la mémoire ;
- un algorithme de planification spécifique à l'application et un schéma d'allocation des ressources ;
- une nouvelle méthodologie et l'architecture.

Pour les systèmes hétérogènes, il existe des alternatives logicielles qui utilisent des ensembles

de bibliothèques pour des applications spécifiques. Pour les applications CPU, le cadre d'exécution OPENCL [45] est fréquemment utilisé pour obtenir une utilisation plus efficace des processeurs multicœurs et comme langage pour intégrer des accélérateurs dans le système (language pour plateforme hétérogènes). Plus récemment, Augonnet et al [46] ont présenté StarPU une bibliothèque de programmation de tâches pour les architectures hybrides, une bibliothèque qui prend en charge les problèmes d'exécution pour les architectures hétérogènes. StarPU facilite l'utilisation d'accélérateurs et simplifie la mise en œuvre de communications, de transferts de données et de gestion de mémoire plus efficaces entre les multiples unités de traitement d'une grappe (plates-formes hybrides).

Ces bibliothèques exploitent généralement un système de programmation par tâches afin d'obtenir des exécutions parallèles. Examinons donc les différents modèles d'exécution parallèle (aussi appelé modèle de synchronisation) étudiés dans cette thèse en référant à la Figure 2.1. Dans cette figure, (a) représente un programme séquentiel tandis que (b) exécute des tâches en parallèle. En (b), certaines tâches nécessitent l'achèvement de leurs prédécesseurs afin de préserver la cohérence séquentielle du programme. La première synchronisation prise en

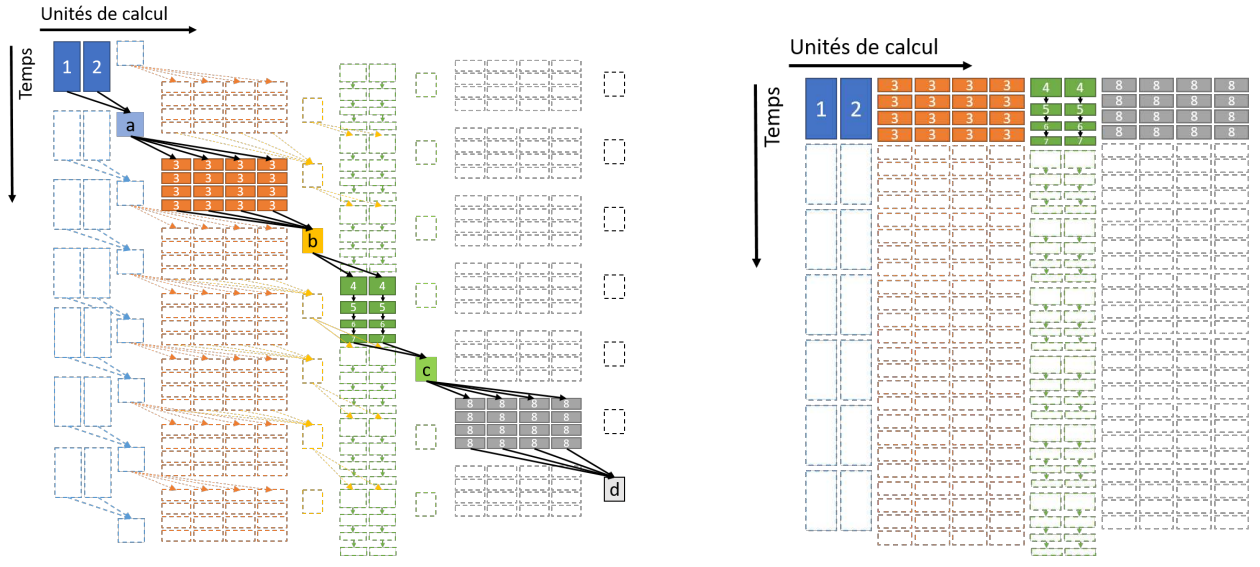


(a) Séquentiel (b) Parallèle (c) Tâches et données parallèles (d) Tâches, données et pipeline parallèles

Figure 2.1 Exemples de graphes de flot de traitement parallèles et séquentiels

charge par nos premiers résultats exposé au Chapitre 4 est : **Parallélisme explicite des tâches :** dans ce mode, les dépendances ne sont pas gérées par le *runtime* et l'application attendra explicitement l'achèvement d'un ensemble de tâches avant d'envoyer une autre charge de travail qui consomme des données produites par une charge de travail précédente. Le cas présenté en (c) correspond à un parallélisme au niveau des données. Le runtime XDVFP que nous proposons ne fait rien à ce niveau. Cependant, cela est largement utilisé dans les tâches LTE par l'utilisation d'instructions *Intel Single Instruction Multiple Data* (SIMD). (d) présente comment les tâches dépendantes et indépendantes peuvent être canalisées dans un système lorsque les dépendances sont gérées par le *runtime*. La diagonale illustrée en (d) correspond au cas du traitement d'un bloc de transport LTE. Puis dans le Chapitre 4 nous explorons le mode de synchronisation du **Parallélisme des tâches piloté par les données**. Dans ce mode, nous pourrions répartir les tâches d'une sous-trame LTE complète et nous synchronisons au niveau de la sous-trame. Le niveau suivant consiste à briser la limite de la sous-trame pour permettre le traitement des sous-trames en parallèle. Nous appelons ce mode **Pipelining logiciel à gros grains :** la Figure 2.2 aide à visualiser la différence entre le pipelining matériel et logiciel.

Les différents *runtime* comme XDVFP peuvent exploiter ces différentes techniques de programmation parallèle, afin d'exécuter une application sur une grappe de calcul. Un des avantages de la programmation parallèle basée sur les tâches est qu'un pipeline logiciel peut avoir des ressources presque illimitées. Ainsi un pipeline logiciel n'est limité que par le nombre



(a) Pipeline matériel (b) Pipeline logiciel

Figure 2.2 Pipelinage logiciel à grosse granularité

de tampons virtuels (quantité de mémoire accessible disponible). Les tâches sont donc des fonctions sans état qui consomment et produisent dans ces tampons. Ne laissant aux bibliothèques d'exécution parallèles que gérer les tampons de mémoire plutôt que des tâches individuelles.

Cela mène aux améliorations de l'exécution grâce à la gestion des ressources mémoires, un domaine très étudié, car elles sont considérées comme le goulot d'étranglement du système d'exécution. Il existe deux niveaux principaux auxquels la gestion de la mémoire est critique, la cache et la Dynamic random-access memory (DRAM). [47] propose un cadre de gestion de cache complet pour les systèmes en temps réel afin de réduire le nombre de ratés de cache, ce qui réduira les accès à la DRAM et réduira ultimement la latence d'exécution du *runtime*. De même, le contrôleur Heracles [48] exploite le verrouillage du cache du processeur et le mécanisme de localisation des données dans la DRAM pour réduire la latence et maximiser l'utilisation des services de recherche "Google". Dans cette thèse, nous utiliserons des mécanismes similaires, plus spécifiquement le verrouillage de cache, afin de maximiser les accès à cet espace mémoire. Ainsi, les techniques de localisation des données assureront une communication plus fiable entre le CPU et le FPGA. En effet, la réduction des éventuels ratés d'accès au cache et à la mémoire, dont les techniques de coloration et de verrouillage, améliore la fiabilité de l'architecture matérielle que nous proposons. Il faut savoir que la localisation des ressources pose des problèmes de programmation linéaire (LP) ou de programmation nonlinéaire (NLP) pour lesquels des solveurs peuvent être implémentés dans des GPP multicœurs, en raison de la nature séquentielle des algorithmes qui peuvent être facilement programmés et implémentés dans des plates-formes informatiques basées sur GPP. Néanmoins, il existe une littérature très riche visant à exploiter le parallélisme dans [49]. Avec le développement récent de la technologie informatique, le calcul de différents algorithmes d'optimisation peut être accéléré sur des plates-formes multiprocesseurs, telles que les GPU [50] et les FGPA [51].

Dans [52], les auteurs étudient l'efficacité de l'algorithme d'ordonnancement et de gestion des ressources pour les systèmes hétérogènes. Ils trouvent qu'ils peuvent réduire la surcharge des algorithmes pour maximiser l'utilisation des ressources, avec des insertions de code via leurs systèmes d'exécution pour les plates-formes hybrides. De la même manière, nos recherches exploiteront différents algorithmes pour garantir la meilleure solution pour chaque application de l'accélérateur d'exécution matérielle.

Au mieux de nos capacités, nous n'avons pas réussi à trouver d'accélération d'exécution matérielle dans les HPC avec pour but d'améliorer l'invariance d'exécution de tâche sous des contraintes de latence. La plupart des accélérations matérielles dans HPC sont pertinentes pour le calcul de données massives et non sensibles à la latence.

Cependant, de nombreuses recherches ont été effectuées sur la mise en œuvre matérielle des algorithmes de planification. Dans des travaux précédents [7], nous avons démontré que l'utilisation de StarPU avec l'algorithme de planification adéquat permet la virtualisation d'une application avide de latence, telle que LTE sur une plateforme donnée. Sur la base de nos travaux, nous pouvons aller plus loin et voir les avantages possibles des algorithmes de planification implémentés par le matériel. [42] démontre les gains de latence des algorithmes de programmation mis en œuvre par le matériel afin de réduire la latence d'un RTOS. Nous prévoyons de tirer parti de l'accélération matérielle pour améliorer la latence à une échelle beaucoup plus grande que [42]. Ils utilisent un RTOS sur Xilinx Zynq, et nous utiliserons un FPGA dans une fiche sur un système d'exploitation compatible HPC dans une grappe. Le document de Mancuso [42] est celui auquel nous comparerons nos résultats.

Pour le cas de la méthodologie et de l'architecture, Mavroidis et al dans leur récente publication [53] ont exposé un calculateur reconfigurable et des systèmes d'exécution pour les HPC (ECOSCALE). Il propose un environnement de programmation et une architecture matérielle pour réduire le trafic de données et la latence dans une grappe, en utilisant des accélérateurs matériels reconfigurables communiquant avec le protocole MPI. En raison du domaine de recherche "de niche" dans lequel nous nous trouvons, le papier Mavroidis est un document auquel nous comparerons cette proposition de recherche. Cependant, l'ECOSCALE n'est qu'une implémentation théorique et reste à prouver empiriquement. L'ECOSCALE est prometteur, mais implique de nombreux changements dans une grappe. Notre approche vise à être une solution plus prête à l'emploi au problème de latence et de maximisation de l'utilisation des HPC hétérogènes.

La gestion des ressources matérielles à faible latence dans HPC n'est pas courante en raison de la latence induite par la communication, comme PCIe ou Ethernet pour envoyer des informations à un accélérateur matériel. La plateforme sur laquelle nous travaillons nous donne l'opportunité de contribuer et d'être pionniers dans un nouveau domaine qui gagne en importance.

Notre recherche vise à réduire le temps d'attente d'exécution, avec l'accélération matérielle. Nous proposons une méthodologie et une architecture matérielle pour l'accélération d'exécution dans des environnements HPC hétérogènes. Une fois l'architecture générique éprouvée, nous approfondirons le côté algorithme du problème. Le premier algorithme d'ordonnancement utilisé dans la preuve de concept est l'équité Max-Min. Il a été choisi pour ses capacités d'équilibrage de charge tout en maintenant la durée d'exécution des tâches, mais plus précisément pour sa facilité d'accélération matérielle [54]. Max-Min fournit une bonne base pour les calculs parallèles. La sous-section suivante présente plus en détail l'algorithme.

2.3.4 L'allocation des ressources

La recherche étant plus orientée architecture qu'algorithme, le problème d'allocation des ressources sera brièvement discuté. Nous proposons d'utiliser l'algorithme Max-Min comme point de départ pour la recherche.

L'équilibrage de charge "*Max-Min Fairness*" (MMF) est une méthode couramment utilisée dans le domaine de la mise en réseau [55]. Elle résoud le problème du partage des ressources entre plusieurs travailleurs, où chacun possède un droit égal aux ressources, mais où un travailleur ne peut pas traiter une charge de le même manière ou efficacité que les autres. La juste part max-min alloue intuitivement les bonnes ressources à chaque travailleur. Par conséquent, un travailleur avec une faible puissance de traitement recevrait une petite charge, tandis qu'un travailleur avec une puissance de traitement élevée recevrait des charges plus importantes. Tout serait fait de manière uniforme, ce qui signifie qu'un travailleur ne recevrait jamais plus qu'il ne peut traiter à un moment donné [56].

La définition mathématique de l'équité Max-Min serait : [57] "Une répartition possible des taux, $\vec{x}$, est"max-min équitable "si et seulement si une augmentation de tous les taux dans le domaine d'allocations doivent se faire au prix d'une diminution de certains taux. Formellement, pour toute autre allocation possible $\vec{y}$, si $y_s > x_s$ alors il doit exister des s' tels que $x_{s'} \leq x_s$ et $y_{s'} < x_{s'}$."

Pour un problème donné, un schéma d'allocation Max-Min peut ou non être possible. Cependant, si un tel schéma existe, il est unique. La preuve mathématique du point de vue du réseau est exposée dans [54]. L'équilibrage de la charge de travail peut être formulé comme un problème de maximisation de l'utilité. Les algorithmes équitables Min-Max et Max-Min sont pris en compte dans la plate-forme de l'industriel. Ces algorithmes ont été choisis principalement en raison de leur compatibilité avec une exécution massivement parallèle, qui peut être obtenue avec différentes technologies, telles que FPGA, GPGPU et DSP [58]. Comme première étape de cette thèse, nous implémentons une version simple et efficace de l'algorithme, capable d'allouer des charges de travail à des lames hétérogènes via une grappe qui maximise la durée de vie avec un schéma d'allocation équitable. Dans les chapitres suivants, nous explorons en profondeur le contexte, la théorie et l'implémentation RTL d'un algorithme d'équilibrage de charge de travail MMF. Dans les étapes suivantes, nous testerons et optimiserons le schéma d'allocation avec une accélération matérielle d'une planification équitable du sac de tâches démontrées dans [59]. L'objectif est de mettre en œuvre des solutions d'équilibrage de charge de travail robuste et efficace.

2.4 Sommaire de la revue de littérature

Cette section résume comment notre projet se base sur littérature exposée ci-dessus. Premièrement, du point de vue de l'architecture matérielle, nous nous comparerons directement à l'ECOSCALE [53], étant donné qu'il s'agit de l'un des rares articles de notre créneau. Comme indiqué dans les sections ci-dessus, nous voulons fournir à la communauté HPC hétérogène une solution plus simple et qui soit directement applicable. Nous comparerons notre architecture de grappe de calcul à la "Chimera" [28] et à la "Catapult" de Microsoft [60] [61]. Nous ne serons pas en mesure de comparer directement l'efficacité, le résultat du débit avec le papier précédemment indiqué, en raison des différences entre les architectures matérielles et des applications visées. Cependant, nous avons quelques similitudes sur lesquelles nous allons exposer les avantages de notre solution, telle que les améliorations de latence par rapport aux solutions standard. Un domaine avec lequel nous avons beaucoup à comparer est celui des RTOS. Nous serons capables de comparer les améliorations de la latence RTOS [42] et de la planification.

Notre recherche vise à réduire la latence d'exécution avec l'accélération matérielle. Nos CPU-FPGA communiquant par QPI [62] qui permet une communication basse latence grâce à la cohérence du cache. Nous comparons les résultats de latence avec les implémentations précédentes de FPGA en fiche, comparable à [41]. Nous proposons d'introduire une architecture matérielle à faible latence pour accélérer la prise de décision des tâches. En fin de compte, nous voulons fournir une architecture générique adaptée pour améliorer le temps d'exécution du processeur à usage général. Pour atteindre notre objectif, nous utiliserons plusieurs techniques en cours de route, comme le verrouillage de la cache afin d'améliorer la fiabilité des communications, similaires à [47] et [33]. Nous comparerons aussi notre solution à la solution d'accélération d'exécution logicielle existante adoptée dans StarPU [46].

Deuxièmement, nous proposerons des algorithmes d'ordonnancement et d'allocation des ressources basés sur des modèles stochastiques pour obtenir un système plus efficace pour notre application donnée, tout en restant générique afin d'avoir une conception évolutive et plus polyvalente. Le but n'est pas de se limiter à la virtualisation du protocole sans fil, mais de pouvoir résoudre le même type de problèmes dans les HPC et ultimement dans tous les processeurs à usage général.

CHAPITRE 3 DÉMARCHE, HYPOTHÈSES DE RECHERCHE ET OBJECTIFS

Le présent chapitre énonce la démarche de l'ensemble du travail. Nous expliquons notamment l'implication technique du partenaire industriel, en passant par la démarche et la méthode, pour finir avec le lien entre les différents chapitres de cette thèse.

3.1 Partenariat Polytechnique-Industrie

Comme expliqué brièvement dans la section 1, le projet initial était basé sur une collaboration entre Polytechnique Montréal et un partenaire industriel. Elle visait à développer une architecture adéquate et une solution complète pour une dynamique écoénergétique et économique du calcul, nécessaire pour l'allocation et la gestion dynamiques des ressources dans des systèmes hétérogènes. L'une des applications visées est une grappe de calcul supportant les normes de communication sans fil telles que LTE et les technologies 5G à venir. Un des principaux objectifs du projet de recherche était de développer et de caractériser les algorithmes d'allocation des charges de travail (un ensemble de tâches) distribuées aux lames de serveur de manière équilibrée.

Un accent particulier a été mis sur la mise en place et la validation des algorithmes de programmation dynamique sur la plateforme conjointe du partenaire et de Polytechnique appelée, "3D Virtual Fabric Processors", une grappe de calcul hétérogène applicable au contexte des applications LTE. L'architecture système de la grappe, au Chapitre 4, a été pensée et évaluée de conjointement avec le partenaire, cependant l'équipement de test, ainsi que l'infrastructure réseau, étaient un prêt de l'industriel, accessible à leur bureau seulement.

Une fois l'étape essentielle de l'analyse de l'ordonnancement effectuée pour tous les algorithmes d'allocation et les performances évaluées via des simulations numériques, une intégration système a été faite pour des tests empiriques. Nous avons donc testé notre API avec des données représentant des trames de télécommunication fictives, puis avec une vraie implémentation de la norme LTE. Certaines valeurs et informations critiques ont été remplacées par de l'information représentative, afin de protéger les intérêts du partenaire industriel.

3.2 Méthodologie et axes de recherche

La méthodologie afin d'atteindre les objectifs définis dans l'introduction est détaillée ci-dessous. En continuant sur la lancée de la section précédente, la méthodologie utilisée dans

cette thèse est analogue à celle utilisée par le partenaire industriel et les autres étudiants impliqués dans ce programme. En effet afin d'atteindre l'objectif global du projet, nous avons divisé les travaux de recherche effectués en trois sous-problèmes, pour éventuellement les rassembler dans une solution adéquate.

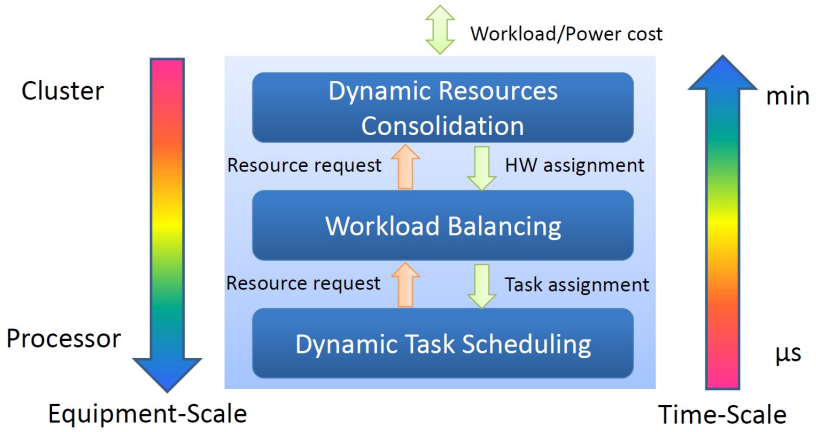


Figure 3.1 Architecture système multi niveaux

L'architecture représentée à la Figure 3.1 est proposée pour atteindre une solution optimale pour le fonctionnement global d'un système, telle une grappe, impliquant un grand nombre de sous-systèmes présentant des comportements différents en termes de dimension, d'échelle de temps et de mécanismes opérationnels (par exemple, tâches déclenchées par un évènement ou sur des intervalles de temps fixes). Cette architecture permet l'amélioration de propriétés connexes telles qu'une facilité d'intégration, une interopérabilité, la modularité, l'évolutivité et l'extensibilité plus grande que d'autres méthodes plus conventionnelles. L'architecture système proposée se compose de trois couches principales. La première couche (la plus basse sur l'échelle de temps et équipement) est au sommet de la pile logicielle habituelle afin de gérer les tâches qui ont des exigences variantes dans le temps en termes de charge de calcul. La deuxième couche se concentre sur la gestion au niveau du serveur via l'équilibrage de la charge de travail. Enfin, la couche la plus élevée sur l'échelle de temps et équipement se préoccupe de l'allocation et de la gestion des ressources globales, l'accent étant mis sur la maximisation de l'utilisation des ressources et de l'efficacité énergétique. En général, la couche supérieure générera des points de consigne pour le fonctionnement de la couche inférieure. Les demandes de ressources peuvent également être émises par les couches inférieures pour déclencher l'ajustement des schémas d'allocation des ressources au niveau des couches supérieures. Cela permettra d'atteindre une solution globale optimale et un fonctionnement stable. Deux axes de recherches sont envisagés concernant l'élaboration et la mise en œuvre du système proposé d'allocation et de gestion des ressources.

Axe 1 : Calcul dynamique, allocation et gestion des ressources réseau dans les grappes de calcul informatiques

Cette orientation vise à développer une architecture adéquate et une solution complète pour l'allocation et la gestion dynamiques des ressources informatiques économes en énergie et rentables pour les systèmes hétérogènes agissants selon multiples échelles de temps. Cette architecture devrait permettre une intégration transparente de diverses techniques dans le fonctionnement des centres de données. L'accent sera mis sur les couches inférieures de l'infrastructure. Les solutions développées fourniront des moyens de réduire les CAPEX, les OPEX et la consommation d'énergie. L'une des applications cibles est une plate-forme de calcul de prise en charge de la norme LTE, qui nécessite une planification dynamique à une échelle de temps de quelques millisecondes. Les algorithmes d'allocation dynamique des ressources et de planification des tâches en temps réel formeront la fonctionnalité de base prise en charge par le plan de contrôle. Afin de répondre aux fortes contraintes en temps réel de cette classe d'application, une implémentation matérielle parallèle basée sur FPGA du plan de contrôle est proposée.

Axe 2 : Prise en charge des calculs et communications à faible variance avec un plan de contrôle implémenté en matériel

L'objectif de cette portion des travaux est de développer des solutions permettant le traitement et la communication à faible variance dans une plate-forme basée sur GPP. Les technologies DPDK et QPI d'Intel semblent adaptées au développement de telles applications. Cependant, DPDK est limité, car il fonctionne sur un noeud à la fois et, par conséquent, les communications entre les ordinateurs ne sont pas gérées de manière à permettre des temps de communication proches d'être déterministes. Comme DPDK est un outil ouvert, il peut être amélioré dans le cadre de ce projet. Le DPDK peut être combiné avec la technologie QPI pour fournir une pile de protocoles reposant sur des liens physiques de haute performance conçus pour les communications quasi déterministes. Une caractéristique notable est la disponibilité d'un noyau matériel pour la mise en œuvre de QPI à l'aide de la structure du FPGA et de ses E/S différentielles haute performance. En combinant une implémentation matérielle efficace avec la technologie QPI (noyau, liaison physique et pile de protocoles associée) et la bibliothèque DPDK, il existe un fort potentiel pour réaliser des plans de contrôle hautes performances. De plus, la compatibilité et l'aptitude de ces technologies à fournir une solution globale au problème considéré seront évaluées dans le cadre de cette thèse.

3.3 Hypothèses de recherches

Les chapitres ci-dessus nous mènent donc aux hypothèses de recherches suivantes :

— Nous pouvons avoir une grappe de calcul hétérogène utilisant des COTS rencontrant les spécifications des protocoles de télécommunication émergents, comme la 5G.
— Il est possible d'avoir une bibliothèque générique d'allocation et gestion de ressource adéquate pour améliorer le traitement des tâches dans une grappe pour la virtualisation de normes de télécommuniations.
— Nous pouvons satisfaire les besoins de faible latence, de bande passante élevée et d'utilisation des unités de calcul grâce à l'accélération matérielle.
— Nous pouvons réduire le CAPEX et OPEX de C-RAN grâce à des améliorations matérielles, logicielles et architecturales de grappe utilisée dans le domaine du HPC dans les TIC.

Ainsi comme l'explique la Figure 3.1 de la Section 3.2, nous avons exploré les différentes couches du sytème, pour finalement arriver à une amélioration système multi-niveaux. Nous explorons donc les hypothèses de recherches définies à travers chaque couche du modèle dans chacun des thèmes suivants.

3.3.1 Thèmes principaux

Les deux axes présentés dans la Section 3.2, se découpent en plusieurs sous-thèmes que nous pouvons considérer comme les principaux thèmes explorés dans chacun des chapitres suivants.

Chapitre 4 :

L'équilibrage de la charge de travail (couche 2) et ordonnancement dynamique des tâches (couche 1)

Cette tâche se concentre d'abord sur le développement d'algorithmes au niveau de la deuxième couche pour une distribution équilibrée des charges de travail aux processeurs. Nous démontrons que nous pouvons maximiser l'utilisation du processeur dans une grappe de calcul homogène. Ceci se fait en fournissant une bande passante garantie pour le transfert de données entre les nœuds de calcul et une mémoire cache réservée pour permettre l'exécution à faible variance des tâches. Nous créons une bibliothèque générique XDVFP basée sur DPDK, en utilisant différents concepts de programmation parallèle, jusqu'à ce jour utilisée en silos, afin de créer un environnement propice à la simulation et à l'expérimentation. Au sein de cette bibliothèque, nous implémentons des algorithmes de planification dynamique au niveau

de la première couche pour gérer le traitement des tâches dans un environnement variant dans le temps. Comme dans des applications telles que le LTE, les ressources informatiques requises varient en fonction des demandes réelles (par exemple, appel téléphonique, vidéo-streaming), la charge de travail attribuée aux processeurs doit être gérée à une échelle de temps plus petite afin de respecter les contraintes de temps. Dans ce but, nous exploitons la technique de la planification dynamique, combinée à une planification basée sur la priorité statique, afin d'obtenir des algorithmes de planification rapides et simples pour répondre aux contraintes de synchronisation strictes. Une analyse d'ordonnancement est réalisée pour tous les algorithmes d'ordonnancement et techniques de programmation parallèles considérés et leurs performances ont été évaluées via des simulations numériques, puis dans l'intégration du système et l'évaluation globale des performances confirmées par des expérimentations. Nous observons que grâce à l'utilisation de la méthode de programmation proposée ainsi que le système d'allocation sur la plateforme XDVFP, nous pouvons réduire le temps moyen d'exécution d'une application basée sur les tâches par un facteur de 2.27.

Chapitre 5 :

La prise en charge des calculs et communications à faible variance avec un plan de contrôle implémenté par matériel

Dans ce chapitre nous trouvons des solutions pour améliorer le calcul et la communication à faible variance dans une grappe de calcul. Les algorithmes d'allocation des ressources développés dans le premier axe fourniront les moyens d'améliorer les performances en optimisant l'utilisation des ressources. Cependant, dans les applications avec contraintes de temps critiques, comme LTE et d'autres normes de communication, des exigences strictes de synchronisation doivent être respectées afin de fournir une qualité de service (QoS) adéquate telle qu'imposée par les normes. Par conséquent, dans notre grappe de calcul hétérogène agissant sur de multiples échelles de temps, nous implémentons des mécanismes d'allocation des ressources respectant les contraintes sur le temps d'exécution du pire cas (WCET) pour le calcul des tâches et la latence de communication du pire cas (WCCL) des transferts de données entre les nœuds de calcul. Par conséquent, nous développons un plan de contrôle accéléré par le matériel (FPGA) pour combler ces deux contraintes, prouvant que nous avons un impact direct sur les performances de la grappe. Nous développerons aussi un traceur matériel distribué en appui pour des raisons de profilage. En effet notre allocateur de ressources matériel (HRA) offre un facteur d'accélération de 760 par rapport à la solution logicielle. Nous déterminons ensuite les interconnexions de la grappe les plus appropriées pour les applications C-RAN LTE. Enfin, des solutions permettant une réduction significative du WCET

sont exposées.

Le calcul de tâches à faible variance d'exécution sur un processeur multicœur à l'aide d'un plan de contrôle matériel

Dans ce chapitre nous améliorons aussi la limite supérieure de l'exécution, la latence et la variance d'éxécution du calcul de la tâche. Il est bien connu que WCET est considérablement augmenté dans les processeurs multicœurs modernes en raison des interférences causées par les ressources partagées dans les logiciels et le matériel. Le cas évident en logiciel est un planificateur de système d'exploitation préemptif qui pourrait interrompre l'exécution des tâches. Côté matériel, la cache partagée, les unités de traitement partagées (telles que la technologie Hyper-Threading d'Intel) ou le mode d'économie d'énergie automatique (par exemple, la technologie Turbo Boost d'Intel) sont des fonctionnalités matérielles qui ont également un impact sur le WCET. Cependant, la plupart des problèmes peuvent être éliminés en utilisant le bon logiciel *backend* (basé par exemple sur DPDK) et en désactivant la fonctionnalité matérielle qui introduit de l'incertitude. Le principal problème restant est lié au dernier niveau de cache partagé (LLC). Une interférence intra tâche se produit lorsqu'une tâche expulse ses propres blocs mis en cache et une interférence inter tâche se produit chaque fois qu'une tâche s'exécutant sur un noyau expulse les blocs mis en cache d'une autre tâche. Il est connu qu'une coloration des pages (partitionnement de cache) et une stratégie de mise en cache avec une approche combinée avec un mécanisme de verrouillage avec contraintes peuvent améliorer le WCET de la tâche. Ce projet propose un cadre permettant l'exécution dans des blocs de cache réservés (verrouillés) et afin de minimiser les échecs de cache dans la zone de mémoire cache réservée. Nous utilisons des techniques de compilation afin d'allouer les données de tâche dans des blocs de cache réservés pour obtenir ainsi une utilisation d'un plan de contrôle matériel pour pré-extraire les données dans le LLC. Nous utilisons une plateforme hétérogène exploitant des processeurs Xeon d'Intel, où les FPGA sont dans un socle directement connecté au CPU par QPI. Un mécanisme de verrouillage de la cache matérielle est disponible dans de nombreux processeurs multicœurs, mais il n'etait pas encore disponible sur le processeur serveur d'Intel intégré au système utilisé dans nos travaux. Grâce à notre plan de contrôle matériel sur cette plateforme hétérogène, nous pouvons utiliser les blocs de données de cache comme une ressource allouable par les algorithmes d'allocation de ressources développés dans l'axe 1.

Chapitre 6 :

Caractérisation et validation du système

La caractérisation des solutions sur différentes couches, y compris les composants développés

dans le cadre de l'axe 2, est réalisée dans une troisième étape du projet. Les tests au début du projet ont été réalisés au moyen de simulations numériques avec certains composants simulés à l'aide de notre bibliothèque. L'évaluation de la performance des solutions développées est effectuée sur la plate-forme expérimentale en utilisant les sorties du traceur distribué en temps réel développé en soutient dans le chapitre précédent. L'application ciblée est un système émulant les exigences LTE. Cette tâche a été exécutée en étroite collaboration avec l'équipe de recherche de notre partenaire qui nous a fourni les spécifications sur la virtualisation de l'interface aérienne LTE. Nous développons de concours avec l'industriel une implémentation logicielle de la norme LTE permettant les expérimentations, les mécanismes pour émuler les charges de travail transférable dans un réseau C-RAN, les spécifications de configuration système et les exigences de traitement du planificateur sans fil LTE. Ceci a permis de définir les mesures de virtualisation LTE nécessaires à l'évaluation des performances du prototype C-RAN. À la fin de l'expérimentation des exigences LTE, nous abordons la possibilité de valider la plate-forme en ce qui concerne les exigences pour les applications 5G.

Interconnexion à faible latence pour une grappe de calcul

Dans ce chapitre, nous contribuons sur les mécanismes d'interconnexion appropriés pour une grappe de calcul exécutant des applications LTE dans le contexte de C-RAN. Nous proposons une nouvelle architecture d'interconnexion basée sur QPI qui minimise la latence et peut être utilisée comme une ressource contrôlée par l'allocateur de ressources développé dans l'axe 1. Nous utilisons les techniques et bibliothèques décrites dans les chapitres précédents pour créer une interconnexion plus déterministe avec une latence équivalente au temps d'accès d'une ligne de cache dans le dernier niveau d'un CPU. Un modèle logiciel de cette interconnexion est développé ainsi qu'un prototype matériel en collaboration avec l'équipe de recherche de l'industriel. Un prototype matériel a également été défini pour valider l'architecture proposée.

3.3.2 Thèmes complémentaires

En plus des cinq différents thèmes de recherches exposés ci-dessus, de nombreux thèmes connexes ont été étudiés et implémentés afin de soutenir le filon principal de cette thèse. Succinctement cette thèse a dû explorer :

— Les traceurs temps réel.
— Différents algorithmes d'allocation de ressources.
— Plusieurs correctifs et amélioration du noyau Linux pour améliorer la dynamicité du système d'exploitation, pour accommoder le traitement temps réel.
— Des systèmes (*framework*) de programmation parallèle.

3.4 Expérimentation

En ce qui concerne la mise en œuvre de chaque conception matérielle, nous avons suivi plusieurs étapes pour ne pas nuire à la validité et au besoin d'une architecture accélérée par matériel pour un problème donné. La méthode d'implémentation est la suivante :

— Étudier et trouver une solution théorique viable.
— Faire une simulation et une implémentation de la solution (C ++).
— Faire une implémentation comportementale, qui est testée en simulation (avant et après synthèse).
— Terminer avec une implémentation RTL et un test empirique de la dite implémentation RTL.

Les prochains chapitres iront dans le détail des multiples axes définis dans les sections précédentes. Le Chapitre 4 détaille directement la base du corps de la recherche sous la forme d'un développé, puis les chapitres suivants iront dans le détail des améliorations proposé sous la forme d'articles publiés au sein de journaux et conférences.

CHAPITRE 4 XDVFP : UNE PLATEFORME POUR LA GESTION DE RESSOURCES RÉSEAU DYNAMIQUE BASÉE SUR DPDK

4.1 Plateforme XDVFP

4.1.1 Gestion et allocation de ressources au niveau des cabinets de serveurs

Comme la dynamique des algorithmes de gestion et allocation de ressources au niveau des cabinets de serveurs fonctionnera à une échelle de temps rapide (ordre des millisecondes pour l'équilibrage de la charge de travail, microsecondes pour la tâche ordonnancement), l'accent de ce chapitre sera mis sur l'efficacité informatique des algorithmes. Étant un cas particulier du problème de la maximisation de l'utilité, la formulation des algorithmes Max-Min, Min-Max ou équité seront prises en considération. La formulation de l'algorithme, l'analyse de la complexité et de la convergence prendront en compte que nous pourrions avoir accès à de multiples technologies pour accélérer leur exécution (FPGA, GPGPU ou DSP). Un accent particulier sera donné sur la mise en place et la validation de l'ordonnancement dynamique des algorithmes sur les processeurs virtuels 3D Fabric (XDVFP) du partenaire, une plateforme de calcul hétérogène applicable au contexte d'applications de télécommunications sans-fil émergentes. Une analyse de l'ordonnancement sera effectuée pour chaque algorithme de planification et les performances seront évaluées via des simulations numériques, qui seront une étape essentielle avant l'intégration du système et l'évaluation globale des performances avec expérimentations.

Ce chapitre présente les travaux suivants :

- Recherche de différents cadres de calcul parallèle à diverses échelles de temps et de modèles de programmation qui a conduit à l'identification des concepts pouvant convenir aux applications ciblées.
- Développement d'un modèle de simulation utile pour confirmer l'évolutivité des solutions proposées.
- Proposer, implémenter et simuler des solutions.
- Soutenir la mise en œuvre et l'intégration accélérée du matériel sur la plateforme XDVFP.

La bibliothèque exploite les meilleures fonctionnalités de différentes bibliothèques de programmation parallèles telles que TBB et Streamit. Cependant, la majorité de ce qui rend XDVFP une bibliothèque unique offrant de hautes performances est le fait qu'elle soit entièrement

bâtie comme une extension de la trousse DPDK. Le DPDK est communément utilisée pour arimer les cartes réseaux d'Intels au "user-space" d'une application. Nous étendons cette bibliothèque pour la réservation et l'isolation des coeurs de calcul au sein de notre bibliothèque en plus de combiner la trousse avec StreamIt pour créer un environnement pour le transfert de données et l'exécution de tâches de façon moins variable à travers les multiples noeuds d'une grappe hétérogène. Ces fonctionalités rendent la grappe plus adéquate pour supporter une architecture d'HPC alors qu'elle était contrainte à une utilisation monoserveur.

XDVFP est à la fois une architecture, un modèle de gestion de ressources et une bibliothèque (API) générique d'exécution pour un environement de grappe hétérogène. XDVFP est une des premières bibliothèques de calcul parallèle multinoeuds acceptant des accélateurs faciles d'intégration, elle peut être considérée comme telle, grâce aux contributions suivantes :

- Nouveau concept de programmation parallèle utilisant des solutions existantes exploitées différemment,
- Le protocole de cohérence de données XDVFP, qui tire le plus possible de DPDK afin d'augmenter le transfert de données au sein d'une grappe de façon moins variable et cohérente.
- Bibliothèque de calcul parallèle multinoeuds acceptant des accélérateurs "out of the box", grâce à notre protocole internoeuds XDVFP.
- Implémentation d'un système d'allocation de ressource multi niveaux et multi noeuds dynamiques.

Le but de XDVFP est donc d'introduire la plateforme, ainsi que de démontrer son utilité par le biais d'une analyse avant accélération.

4.1.2 Architecture générale

La bibliothèque XDVFP a été conçue pour l'architecture logique d'une grappe de calcul hétérogène avec différents noeuds prêts à accepter différents types d'accélérateurs. À titre d'introduction et d'évaluation de la bliblothèque, nous visons l'implémentation dans ce chapitre d'une grappe de calcul homogène. La grappe est donc constituée d'un ensemble de lames de serveur interconnectées par un tissu réseau typique, avec un commutateur maitre (spine) et mutli commutateurs comportant plusieurs esclaves (leaves).

Comme présenté dans la vue logique d'une grappe à la Figure 4.1, la plate-forme XDVFP utilise 2 configurations de lame de serveur, le maître et les esclaves. Au sein des différences configurations de serveurs, maitre (M) et esclave (S) certains moteurs et fonctionnalité sont répliquées. Bien qu'ils aient des utilités similaires, certaines subtilités sont de mises. Nous passerons donc à travers les divers blocs des différentes configurations exposés plus haut.

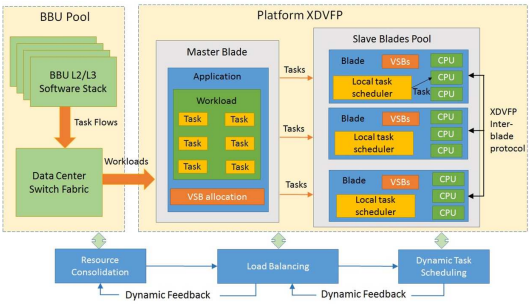


Figure 4.1 Architecture système de XDVFP

Certains blocs de la figure ne seront pas discutés avec plus de détails, car leur fonctionnalité implicite peut être facilement définie par le nom de cette dernière.

XDVFP introduit la gestion des ressources multi-niveaux au sein même de la bibliothèque. Plus précisément sur les niveaux L1, l'ordonnancement des tâches, et L2 sur la gestion des ressources de façon équilibrée, voir sur la Figure 4.1. Sur Esclave nous implémentons un allocateur de tâches au niveau L1, il est responsable de l'ordonnancement des tâches à être exécutées sur le noeud. Nous retrouvons aussi un allocateur de tâches sur Maître, cependant il vise à ordonnancer les tâches associées au moteur de gestion de ressources L2. En effet, notre allocateur génère des tâches pour le calcul de la décision d'allocation des tâches au sein de la grappe. Cela nous mène au gestionnaire de tâches sur M, qui assigne des tâches et charges de travail aux noeuds S. Cette allocation multi-niveaux n'est possible que grâce à une boucle de rétroaction qu'implémente XDVFP entre les niveaux L1 et L2, mais aussi grâce à un système de notification entre noeuds, supportés par le "XDFVP Inter-Blade Protocol". Pour soutenir les fonctionnalités de XDVFP, le modèle de communication initial est un nœud maître-esclave connecté en utilisant une topologie en étoile, où le nœud central est un commutateur à faible latence de haute performance qu'on retrouve dans des systèmes dits de "*High Performance Computing*" (HPC). Nous supposons que tous les nœuds sont équipés d'un adaptateur réseau compatible offrant *Remote direct memory access* (RDMA).

4.1.3 Modèle d'allocation et gestions de ressources sur XDVFP

À travers les sous-sections suivantes, nous décrivons en détail la solution proposée pour l'implémentation d'un système d'allocations de ressources multi échelle. Commençons par la formulation de différents concepts et définitions permettant de comprendre en détail les contributions de XDVFP. Ils serviront de base pour lancer la discussion de notre solution.

Une application est un programme exécuté sur les unités de calcul du noeud maitre. Pour un

processeur x86, tel que les Xeon d'Intel supportant le "Hyper-Threading" utilisés dans cette thèse, une unité de calcul, communément référée comme coeur, soutient deux fils permettant l'exécution en parallèle d'une ou plusieurs fonctions. Souvent une fonction correspond à une ou des tâches à exécuter. Dans le contexte de XDVFP, une tâche est définie comme une fonction prenant comme entrée un ensemble de données et qui en produit un autre. Nous utilisons des tampons de flux virtuel (VSB) pour définir ces ensembles de données d'entrée et sortie. Un VSB représente un tampon synchronisé avec une ligne de cache de Linux. Pour finir, une charge de travail est un ensemble ordonné de tâches.

Ainsi la bibliothèque a pour contribution principale d'offrir une infrastructure permettant à un moteur d'allocation de ressources de répartir la charge de travail entre les lames de serveur pour une meilleure utilisation des ressources de calcul et de communication de la grappe. Au sein de cette thèse, nous travaillons avec des tâches qui s'exécutent à une échelle de temps dans l'ordre de la microseconde. Cela signifie que l'exécution des fonctions associées au gestionnaire de ressource (L2) sera ajoutée à l'exécution totale de l'application, pour ainsi augmenter la latence totale du programme, de même pour l'ordonanceur de tâche (L1). Par conséquent, les algorithmes d'allocation des ressources doivent être rapides et aussi déterministes que possible afin de minimiser l'impact sur l'application. C'est la principale motivation de l'évaluation d'un gestionnaire de ressources accéléré dans le Chapitre 5.

L'allocation et l'ordonnancement des tâches d'un graph acyclique dirigé (DAG) est un problème NP-difficile qu'il n'est pas possible de résoudre en utilisant l'algèbre linéaire, même si le problème est relativement petit. Ainsi, afin de réduire la complexité du problème, nous divisons les fonctions d'ordonnancement (L1) et de gestion des ressources (L2) sur différents noeuds de calcul agissant sur des niveaux d'opération différents. Ces niveaux définis dans l'architecture logique de XDVFP se reflètent sur la Figure 4.2 :

Dans l'architecture d'exécution XDVFP, L2 est associé au gestionnaire de ressources et L1 est associé à l'ordonnanceur local. En partant des différents types de programmation parallèles énoncés dans le Chapitre 2, XDVFP utilise une implémentation de **parallélisme des tâches pilotée par les données**. Ainsi, la localité et la quantité de données devant être utilisées dictent l'allocation d'une tâche, à la fois au niveau de L1 et de L2. Prenons le cas d'une trame LTE. Dans ce mode, nous répartissons les tâches de la trame complète (L2) et nous synchronisons au niveau de la sous-trame (L1). Le niveau suivant consisterait à briser la limite de la trame pour permettre le traitement des sous-trames en parallèle. L'exécution parallèle des sous-trames serait possible grâce à l'utilisation d'un **pipeline logiciel à gros grains**. Dans le cas du *runtime* XDVFP, le pipeline logiciel n'est limité que par le nombre de tampons de flux virtuels (VSB) qu'il peut allouer. Les tâches sont donc des fonctions sans états qui

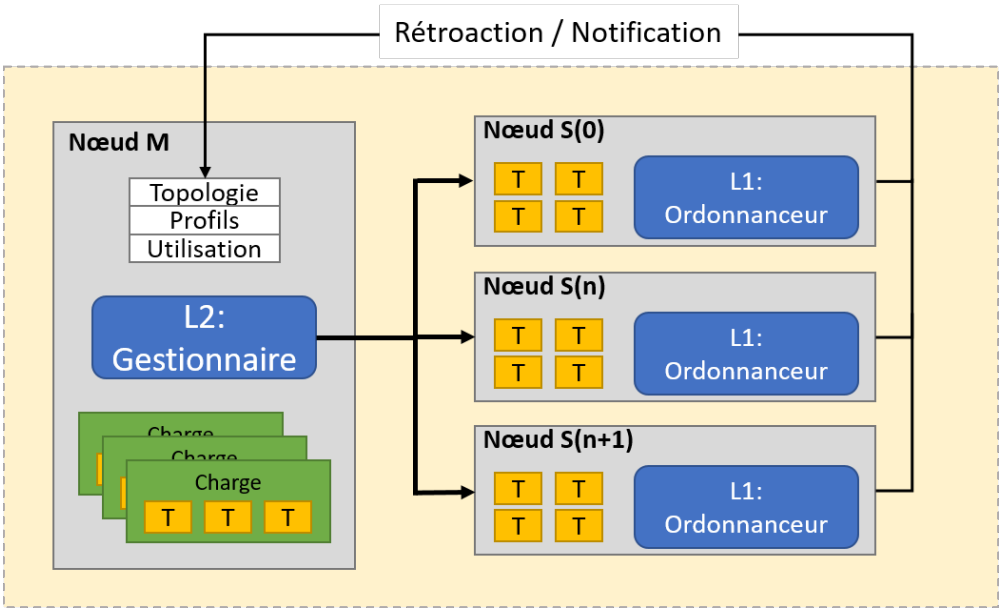


Figure 4.2 Moteur d'allocation de ressources

consomment et produisent des données retirées et injectées de ces tampons. Ainsi, le *runtime* XDVFP gère ces tampons à travers le système au niveau des gestionnaires de ressources de M et S. Le nombre de VSB allouable dans un système est fixe et ce nombre total est connu de XDVFP. Cela nous ramène donc avec un gestionnaire de ressources au niveau L2 qui génère des charges de travail contenant toutes les informations nécessaires pour une exécution sur un noeud de calcul. Le gestionnaire utilise donc un moteur d'allocation qui calcule le coût d'exécution d'une charge de travail sur les noeuds disponibles en se basant sur l'état courant de la grappe. Notons que l'algorithme de gestion de ressources au niveau L2 peut constituer un projet de recherche à part entière. Nous avons choisi d'utiliser un algorithme communément utilisé dans le monde de la gestion des ressources dans le domaine des réseaux, le Max-Min fairness. L'explication théorique de l'algorithme est donnée au Chapitre 2 et une implémentation logicielle matérielle est faite au Chapitre 5. Il calcule le coût d'exécution de la charge de travail basée sur le profil des tâches dans la charge, qu'il compare avec l'utilisation courante de la grappe, information fournie par les boucles de rétroactions, notre traceur haute précision, ainsi que la topologie des accélérateurs disponibles dans chaque noeud. Alors le moteur de calcul de coût assigne la charge de travail au noeud qui permettra l'exécution la plus efficace de la charge. Une fois la charge accumulée (ses VSB et tâches), l'ordonnanceur assigne les différentes tâches aux unités de calculs disponibles (coeurs) basés sur les dépendances inter tâche (donc de données) et en fonction de l'état actuel des coeurs. À noter que XDVFP est une bibliothèque permettant l'utilisation et la simulation de différents

algorithmes de gestion et d'allocation de ressources, L1 et L2. Les choix d'algorithmes faits sont les nôtres, mais la bibliothèque permet aux développeurs de créer leur propre architecture de gestion de ressources avec les algorithmes de leur choix.

Les sections suivantes rentreront dans le détail des techniques de programmation parallèles implémentés dans XDVFP sur DPDK, pour finir par une description du comportement d'une application sur XDVFP.

4.2 XDVFP sur DPDK

4.2.1 Bibliothèques de programmation parallèle

Pour ce faire, XDVFP cumule et adapte différentes bibliothèques logicielles et techniques de programmation pour atteindre le but escompté. En effet, afin d'obtenir le meilleur rendement possible des processeurs multifilaires, les systèmes d'exécution seuls ne font pas l'affaire. Ils doivent être couplés avec un type de programmation parallèle qui profite de l'architecture du matériel. Il s'agit d'une programmation basée sur les tâches, le format de tâches et d'un ordonnanceur de tâches. Le projet de recherche est principalement construit sur la boite à outils pour le développement de plan de données (DPDK) [19] d'Intel. XDVFP s'inspire et bâtit sur de nombreux logiciels, mais surtout : l'*Intel's threading building blocks* (TBB) et StreamIt.

Intel's Threading Building Blocks (TBB) [63] est une bibliothèque d'exécution qui soutient la programmation parallèle en C++. Elle permet au programmeur de faire abstraction des fils matériels et de laisser l'ordonnanceur s'occuper de gérer l'assignation des tâches au matériel. TBB est constitué de deux bibliothèques dynamiques : une pour le support général et l'autre pour l'allocation de mémoire et la gestion des entêtes correspondants. L'ordonnanceur de la bibliothèque dynamique se base sur la méthode du "*task stealing*" [7].

StreamIt [64] est un système comprenant un langage de streaming haute performance et un compilateur. StreamIt est construit sur un modèle synchrone avec profil de calcul continu (*streaming*). Ce modèle permet aux programmeurs d'implémenter des acteurs indépendants, appelés sous StreamIt comme filtres, qui interagissent sur les données des canaux d'entrée et de sortie.

Puis nous bâtirons nos solutions par-dessus la trousse DPDK. DPDK est un ensemble de librairies intergiciel qui peuvent être utilisées pour les paquets avec un nombre minimum de cycles CPU (généralement moins de 80 cycles, de l'ordre de 500ps chacun selon Intel).

Il peut être utilisé pour développer des algorithmes de capture de paquets rapides et pour exécuter des piles de raccourcis tiers. Selon Intel, les fonctions de traitement des paquets ont été évaluées comme fonctionnant jusqu'à 160 Mfps (millions de trames par seconde, en utilisant des paquets de 64 octets) avec une carte d'interface réseau (NIC) PCIe Gen-2. Ce niveau de performance peut permettre des applications de type LTE sur CRAN dans les centres de données.

DPDK est une bibliothèque de communication qui peut être ajoutée au noyau du système d'exploitation Linux. Il fonctionne naturellement sur les ressources de bas niveau offertes par les processeurs tels que Quick Path Interconnect (QPI) qui est la deuxième clé permettant la technologie promue par Intel [65]. Il s'agit d'une technologie de protocole multicouche et la solution proposée par Intel pour interconnecter physiquement ses processeurs récents. Les liens QPI sont des liens rapides, mais relativement étroits qui peuvent assembler les processeurs dans une architecture de plateforme de style mémoire partagée distribuée. L'absence de contrôleurs centralisés permet de prendre en charge une bande passante beaucoup plus élevée avec une latence plus faible que les bus larges conventionnels. Il dispose d'un protocole *snoop* optimisé pour une faible latence et une grande évolutivité, ainsi que des paquets et des structures planifiées permettant de conclure rapidement les transactions. Les débits des liens QPI sont 4.8 GT/s et 6.4 GT/s. Les unités de données transférées à la couche physique (Phit) ont une largeur de 20 bits. Ces Phit sont regroupés en FLIT 80 bits (*Flow control unit*). QPI est organisé comme un protocole à 5 couches censé être largement transparent pour les utilisateurs. QPI et DPDK sont au cœur de la plate-forme du partenaire.

Ces projets tentent tous d'exploiter les parallélismes de tâches au sein d'un processeur multi-cœur ou dans un système qui comprend un grand nombre de ces processeurs. Nous explorons plus en détails comment chacune de ces bibliothèques a été exploitée au sein dans ce projet dans la Section 4.2.2. Notre solution proposée dans ce chapitre inclura donc notre propre bibliothèque développée par dessus ces trois projets, couplée à de l'accélération matérielle. Dans ce contexte, ces accélérateurs agiront comme support à XDVFP afin d'augmenter les performances de nos calculs de coût d'exécution de tâches.

4.2.2 Modèle de programmation : StreamIt sur DPDK

On sait que le modèle de programmation basé sur les tâches est souvent un meilleur modèle de programmation multifils pour de nombreuses raisons, tel qu'expliqué dans [66]. En quelques mots, le modèle de programmation multifilaire laisse le contrôle (allocation et synchronisation) du traitement des ressources (cœurs de processeur) entre les mains du programmeur. Pour le modèle de programmation par tâches, un *runtime* logiciel est nécessaire pour gérer une

application spécifiée avec des tâches. Les *runtimes* logiciels fournissent une API pour capturer les tâches et un *backend* peut être utilisé pour ; allouer, acheminer et exécuter des tâches sur les ressources de traitement disponibles basées sur diverses approches algorithmiques. Elles permettent parfois de mesurer les performances pour alimenter l'allocation de ressources et résoudre les dépendances entre les tâches si ce modèle est pris en charge. Ainsi que déplacer les données entre les nœuds de calcul lorsque le *runtime* prend en charge plusieurs nœuds.

Une autre contribution de XDVFP est son amélioration de l'exécution parallèle de tâches en exploitant la programmation de flux (*stream*) implémentée par dessus DPDK. Prenons une application (5G) comprenant de nombreux DAG interdépendants où chaque sommet représente une tâche, Figure 4.3 reflète ce genre d'application. On sait que le problème d'allocation et de planification des tâches est un problème NP-difficile [67]. Le système actuel cible les applications qui peuvent être représentées comme plusieurs programmes de flux parallèle. La Figure 4.4 expose le pipeline logiciel à grain grossier des programmes de flux dans le contexte d'une architecture multicœurs. Nous présentons un exemple exécuté sur un noeud avec 4 cœurs de travails. Typiquement le DAG est exécuté sur plusieurs cœurs et l'utilisation des cœurs n'est pas optimale en raison des dépendances.

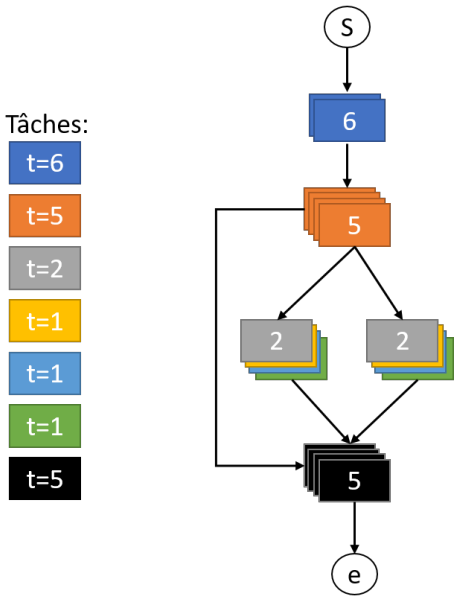


Figure 4.3 Exécution de tâche + données

Cependant, avec plusieurs DAG en parallèle, nous pouvons atteindre un état stable où il y a toujours des tâches exemptées de dépendances exposé par la Figure 4.4. Il s'agit donc d'un pipeline logiciel.

L'utilisation des unitées de calcul pourrait atteindre près de 100% avec de nombreux pro-

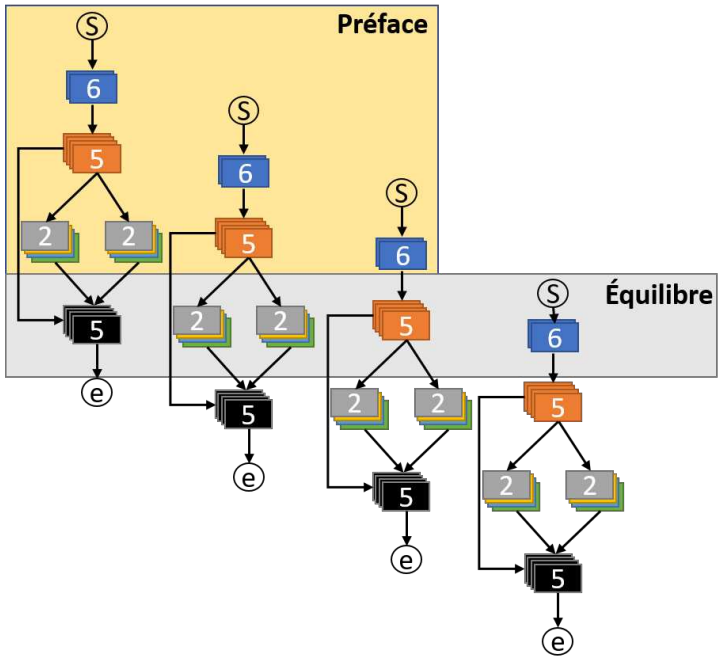


Figure 4.4 Pipeline logiciel à granularité élevée

grammes parallèles comme le montre la Figure 4.5.

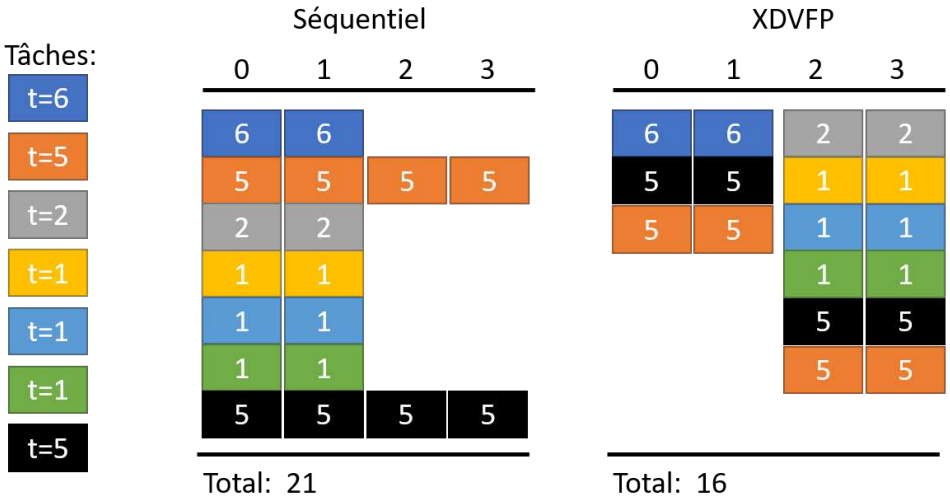


Figure 4.5 Ordonnancement sur 4 coeurs

Donc, si nous capturons les dépendances, évitons la synchronisation explicite au niveau de l'application et laissons la gestion des dépendances dans un *runtime* optimisé, nous pourrions atteindre un état stable où il y a toujours une tâche à exécuter. Par conséquent, pour un pro-

gramme de flux où les tâches ont à peu près les mêmes temps d'exécution, les algorithmes de planification des tâches peuvent ignorer les dépendances et utilisent un algorithme gourmand pour remplir les les unités de calcul sur une architecture à plusieurs coeurs (sur un noeud). Dans le cas de la structure d'allocation de ressources XDVFP, c'est le niveau de planification de tâche dynamique L1 qui implémente un algorithme gourmand pour garder occupé ses cœurs. Les travaux de Gordon & al. [68] sur le pipeline de logiciels de tâches à grain grossier était uniquement destiné à une architecture multicœur et ne tenaient pas compte des communications entre les serveurs lames. Un objectif du processeur virtuel XD est d'étendre ce concept à une architecture à plusieurs noeuds (mettre toutes les lames de serveur dans cet état stable où elles ont toujours des tâches à exécuter). Ceci est accompli par les niveaux d'allocation des ressources L2 et L3. Nous avons pu atteindre cet objectif au sein de XDVFP grâce à l'utilisation de notre modèle de programmation combiné avec "StreamIt" par dessus DPDK. En effet, l'utilisation de la trousse nous permet de passer au dessus la couche noyau du système d'exploitation et nous permet de réserver la bande passante des cartes réseaux. Nous pouvons donc "binder" ces cartes pour l'usage unique de notre application, réserver et isoler les unités de calcul nécessaires pour assurer des transferts constants et fiables tout en contraignant notre application dans le "user-space".

4.2.3 Une application exécutée sur XDVFP

En vue de tout ce qui a été dit dans les sections précédentes, XDVFP est une bibliothèque efficace avec des mécanismes de gestion dynamique des ressources d'une grappe de calcul. Afin de mieux comprendre comment les différents blocs se joignent et interagissent, nous explorons le fonctionnement d'une application sur XDVFP.

L'application alloue un nombre fini de tampons de flux virtuels pour l'ensemble système et envoie des tâches au *runtime* XDVFP en spécifiant des tampons d'entrée et de sortie pour ces tâches. Les tâches distribuées à l'exécution sont accumulées dans une charge de travail (Workload) au lieu d'être exécutées immédiatement dans le système. L'application décide explicitement quand envoyer la charge de travail (placement de la charge de travail) au contrôleur de gestion des ressources. Puis, ces tampons sont gérés par le *runtime* XDVFP. Plus précisément par le moteur de gestion de données global (maître) et local (esclave). L'application définit tous les VSB utilisés dans le système, ainsi l'ensemble de VSB dans le système est statique, mais il est reconfigurable dynamiquement. La mémoire virtuelle de Linux est allouée à tous les tampons sur tous les nœuds de calcul. Le *runtime* XDVFP prend en charge deux types de VSB : scalaire et vectoriel. Un VSB scalaire est simplement un tampon d'une taille maximale X alors qu'un vecteur VSB est défini par un certain nombre

de blocs X de taille Y. Le vecteur VSB permet un RDMA semblable à la dispersion entre les noeuds. Puis une tâche envoyée au *runtime* (et accumulée dans une charge de travail), entraine que l'application doive reconfigurer explicitement ses VSB consommés et produits. Cette étape est nécessaire pour optimiser les transferts de données entre les noeuds. Cela signifie qu'une tâche mise en file d'attente dans une charge de travail comprend 2 ensembles d'informations, son descripteur et son profil. Le descripteur de tâche est principalement ce qui est nécessaire pour exécuter la tâche dans le système (pointeur sur une fonction C, pointeurs vers les VSB *inout*). Le profil de tâche représente les caractéristiques d'une tâche, telles que la quantité de données consommées et produites par la tâche et le temps d'exécution du pire cas (WCET).

Lorsque l'application place la charge de travail dans un système, un ID de contexte d'exécution unique est associé à la charge de travail et il est placé en fonction d'une méthode de placement spécifiée par l'application. Ainsi lorsqu'une charge de travail est placée dans le système, le *runtime* retourne un ID de contexte d'exécution associé à la charge de travail placée. L'application peut éventuellement attendre la fin de cette charge de travail ou poursuivre la génération d'une nouvelle charge de travail. Au niveau du noeud de calcul recevant la charge de travail, les fils de travail tirent perpétuellement sur une mémoire tampon partagée contrôlée par le moteur de l'ordonnanceur de tâches local afin obtenir une tâche à exécuter. Quand l'exécution de la tâche est effectuée, il avertit l'ordonnanceur de tâches local. Cela nous mène à la méthode de placement du moteur de gestion de ressource. Il utilise un algorithme afin de déterminer où la charge de travail doit être exécutée. Une charge de travail peut être exécutée sur un seul nœud de calcul ou fractionnée sur différents nœuds de calcul. Nous nous sommes d'abord basé sur l'algorithme LPM puis Max-Min comme méthode de placement utilisant des profils de tâches pour déterminer comment partitionner une charge de travail. Une méthode plus simple consiste à spécifier explicitement sur quel nœud de calcul la charge de travail doit être réalisée. Ici, un développeur voulant utiliser XDVFP pourrait implanter l'algorithme de son choix. Ce paragraphe explique le fonctionnement en ce qui a trait au mécanisme d'allocation de ressources multinoeuds multi niveaux.

Afin que le mécanisme décrit plus haut fonctionne adéquatement, nous avons proposé et implémenté le **Protocole Inter-Noeuds**, aussi appelé le XDVFP *Inter-Blade Protocol* qui constitue la base du protocole de communication utilisé pour échanger des données entre les noeuds. Il prend en charge les types de paquets suivants :

— transfert de charge de travail du nœud maître au nœud esclave ;
— notification de fin d'exécution de la charge de travail du nœud esclave vers le nœud maître ;

— mise à jour du descripteur de données du nœud maître vers le nœud propriétaire des données ;
— mise à jour de la destination des données du nœud maître au nœud propriétaire des données ;
— transfert de tampon de données d'un nœud à un autre.

Une fois les données acheminées, nous devons nous assurer que la cohérence entre données est respectée lors de l'exécution des charges de travail. Le **Protocole de cohérence des données** XDVFP garantit que la cohérence séquentielle de l'application est conservée lorsque nous exécutons toutes les tâches en séquence (un coeur) et lorsque nous l'exécutons dans un environnement multicoeurs/multi noeuds. Cela signifie que si nous expédions une tâche $N+1$ qui produit un tampon X puis une autre tâche $N+2$ qui consomme ce tampon X, la tâche $N+2$ est verrouillée jusqu'à la tâche $N+1$ se termine. En d'autres termes, le *runtime* XDVFP gère les dépendances entre tâches et également le transfert de données entre les nœuds de calcul. Alors, si la tâche $N+2$ a été distribué sur un autre nœud de calcul que la tâche $N+1$, le protocole impose la règle suivante : un tampon appartient à un calcul unique, le nœud et le tampon ne peuvent être écrits (produits) que par leur nœud propriétaire. Plusieurs tâches peuvent écrire dans le même tampon, mais ces tâches doivent être exécutées sur le même nœud de calcul. Cette règle simplifie grandement le développement de la bibliothèque. Comme nous l'avons mentionné précédemment, le *runtime* XDVFP alloue toute la mémoire VSBs Linux sur chaque nœud informatique. Le moteur de gestion des données (sur les nœuds maître et esclave) gère un descripteur de données pour chaque VSB. Ce descripteur comprend la description de la mise en page du tampon, un état local et un état global. La description de la disposition est utilisée pour le transfert de données et elle est reconfigurable par rapport à utilisation de tampon (changer la taille d'un tampon scalaire réduira le transfert de données). L'état local n'est valide que pour le nœud propriétaire. Il comprend un nombre de sémaphores, de producteurs et de consommateurs, ainsi qu'un masque de destination distante. L'état global inclut le nœud qui est le propriétaire si le tampon local est valide. Dans notre modèle de programmation, tous les noeuds de serveurs sont des nœuds de calcul, ce qui signifie qu'il fournit des cœurs de processeur qui pourraient être utilisés pour exécuter des tâches. Un noeud du système sera le nœud maître et les autres seront considérés comme des nœuds esclaves. Le nœud maître exécute une ou plusieurs applications et gère les ressources du système. Les nœuds esclaves exécutent principalement les tâches envoyées par le nœud maître. Le *runtime* XDVFP (*X-Dimensions Virtual Fabric Processors*) est donc une bibliothèque de programmation expérimentale destinée à gérer l'exécution des tâches dans un système hétérogène. Le *runtime* XDVFP est la couche logicielle utilisée pour programmer le prototype de plateforme hétérogène présenté dans les premiers chapitres.

Les résultats de la section suivante refléteront le gain de performances d'un système lors de l'exécution d'une application utilisant notre bibliothèque XDVFP. Cette expérimentation utilisera tous les blocs exposés au-dessus.

4.3 Résultats

En nous basant sur notre bibliothèque, nous pouvons tester la validité de notre proposition. L'API de notre bibliothèque permet la simulation de n'importe quelle application basée sur tâches voulant utiliser notre système (*framework*) d'allocation de ressources et de calcul. Dans cette section résultats, nous utiliserons des vraies données fournies LTE par notre partenaire. Les résultats ci-dessous exposent donc l'efficacité des différentes solutions proposées de façon expérimentale utilisant du vrai "*signaling* LTE". Le signaling de l'application testé est composé d'un flux de programmes (DAG) pour la liaison descendante LTE (*downlink*) représentée sur la Figure 4.6.

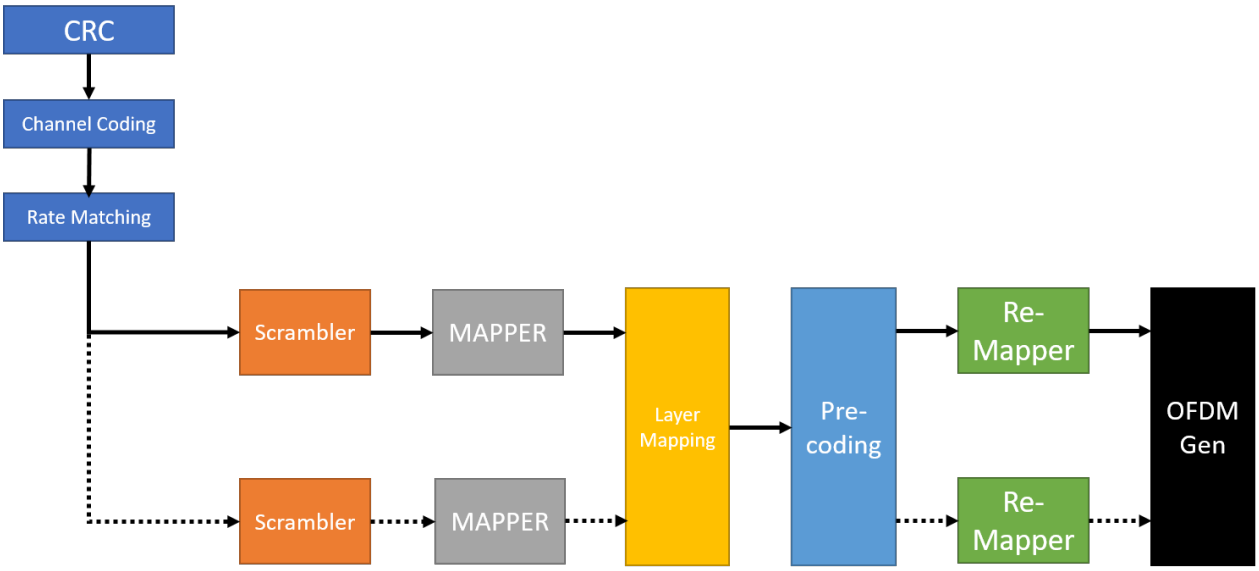


Figure 4.6 Partitionnement des tâches de liaison descendante

L'application XDVFP est un programme C (API) implémenté en utilisant ces quatre étapes :

- définition des tampons ;
- définition des tâches ;
- définir la ou les fonctions qui répartissent les tâches, reconfigurer les VSB et synchronisation à la fin de la charge de travail ;
- assemblez toutes ces choses ensemble.

Les tests empiriques seront faits sur un noeud de grappe avec 14 coeurs utilisant la technologie d'*Hyper Threading* d'Intel, nous donnant essentiellement 28 unités de calcul. Afin d'avoir une application ininterrompue lors des tests, nous avons modifié le "*grub*" du serveur pour allouer 1 des coeurs (donc deux unités de calcul) pour le système d'exploitation (CentOS-Linux). Cela nous permet de nous assurer qu'aucune tâche de notre application ne sera interrompue par une tâche du noyau Linux. Il reste donc au test 13 coeurs et 26 unités de calculs distinctes. Le nombre de VSB allouable par application est variable et peut être défini par le développeur de l'application. Pour les résultats exposés ci-dessous, le nombre de VSB est celui nous donnant les meilleurs résultats pour chacune des implémentations.

Le *runtime* XDVFP a été entièrement modélisé en logiciel et la mise en œuvre d'accélérateurs matériels sera démontrée dans les Chapitres 5 et 6 . À cette phase du projet de thèse, nous montrons comment le système d'exécution XDVFP est efficace pour planifier des tâches sur un seul noeud sans accélérateur matériel. Les accélérateurs matériels sont principalement utilisés pour déplacer efficacement les données entre les noeuds et pour équilibrer la charge de travail entre les noeuds. La démonstration de noeuds multiples avec une exécution XDVFP accélérée matériellement est la prochaine phase de ce projet.

L'expérimentation représente l'implémentation d'une application de liaison descendante LTE, sur la même plateforme de XDVFP, mais sans nos améliorations proposées. Ce test traite en permanence des blocs de transports L1. Pour la configuration du test, nous avons le nombre de cœurs de travail qui est un paramètre évident, le nombre de Virtual Stream Buffer utilisé par l'application et le mode de synchronisation sont présentés dans le Tableau 4.1.

Le code initial reçu de l'équipe du partenaire fonctionnait dans un mode équivalent au parallélisme des tâches explicites. Bien que cette technique ait été optimisée elle reste non-optimale, car nous obtenons 80% d'utilisation des coeurs de travail dans ce mode. Les meilleurs résultats sont obtenus en utilisant le logiciel de pipeline à gros grains mode avec 83430 tampons de flux virtuels.

Tableau 4.1 Résultats de XDVFP avec des systèmes d'Allocation de ressource différents

Type	VSB	Temp Moyen	Exec	Recup Donnée	Recup Tâche	Avert
Explicite	9270	207682280 us	80.5%	19.20%	0.1%	0.13%
Gros Grain	83430	82045662 us	99.59%	0.05%	0.09%	0.27%
Amélioration	N/A	2.28	1.24	384	1.11	0.48

Nous observons clairement dans les résultats exposés au Tableau 4.1, que notre approche d'exécution de tâche en utilisant un parallélisme de gros grain donne une utilisation des coeurs quasi maximale avec une valeur d'utilisation des unités de calcul qui approche le 99%. L'amélioration des performances est due à l'allocation des tâches de façons grossières qui limite le besoin que chaque coeur demande pour une nouvelle tâche dans une queue. Comme le montre le Tableau 4.1, où la majorité des pertes sont engendrées par l'action "Récupération de données *pull*".

Nous observons aussi que nous avons une moyenne de 2.24 ms par VSB pour une bande passante de 5.6Gbs dans l'implémentation explicite contre 0.983 ms avec une bande passante de 6.29 Gbs avec l'allocation à granularité grossière. Nous voyons donc que nous obtenons une amélioration avec l'utilisation du parallélisme à plusieurs niveaux. Cependant, toutes les exigences ne peuvent pas être remplies, nous sommes sous le requis de 10Gbs de bande passante (*backhaul* de la grappe) et le traitement de taches (et VSB) reste élevé autour de la milliseconde. Nous pouvons aussi évaluer les résultats en utilisant notre traceur distribué basé sur Lttng, comme le montre la Figure 4.7.

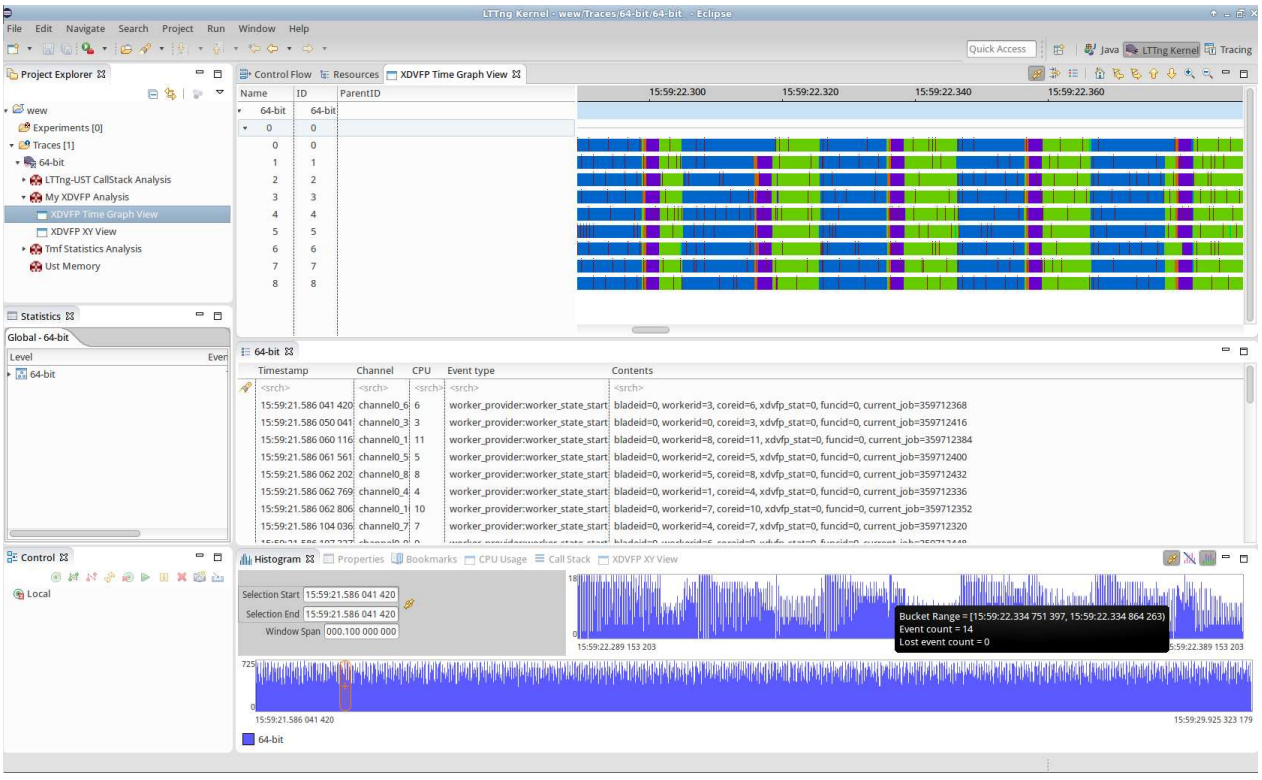


Figure 4.7 Trace de l'application LTE utilisant XDVFP sur un noeud

Grâce à notre nouvelle technique de programmation parallèle et notre technique d'allocation

de tâches à granularité grossière, la majorité de l'application se déroule dans des fonctions d'exécution, tel que le montrent les différents codes de couleur autre que le rouge. Une couleur correspond à un type de tâche (fonction). Les fins d'intervalles, rouges, représentent le temps où le coeur de calcul est inactif, en attente d'une tache à exécuter ou en manque de données pour exécuter une tâche.

Il faut noter que dans un environnement avec de multiples nœuds de calcul, ces performances seront dégradées par les communications entre les noeuds et par la façon dont les charges de travail sont placées entre les noeuds. L'accélération matérielle d'exécution XDVFP (qui comprend l'allocateur de ressource XD) en combinaison avec une carte d'interface réseau / matrice de commutation basse de latences optimisées limitera cette dégradation.

4.4 Discussion et améliorations possibles

Bien que la section résultat semble démontrer que notre implémentation est adéquate pour la virtualisation de la liaison descendante de LTE, nous observons que les résultats de temps d'éxécution sont encore élevés. Nous pensons qu'il peut être réduit car il inclus l'exécution des tâches associées à notre calculateur de coût de charges de travail. En effet, bien que nous avons une amélioration de 11% de la bande passante, 24% sur l'utilisation des coeurs de calculs et d'un facteur 2.28 sur le temps d'exécution moyen par tâche. Nous pouvons constater que les sections bleues, sur la Figure 4.7, correspondant aux tâches associées au calcul des coûts de charges de travail par le gestionnaire, utilisent pour une durée non négligeable les ressources de calcul disponible. Nous pensons que nous pouvons améliorer les résultats exposés ci-dessus en simplifiant le problème d'allocation et en utilisant les avantages des accélérateurs de la grappe de calcul. Nous proposons de simplifier le problème de l'allocation à un ensemble de tâches indépendantes. L'attribution d'une tâche indépendante est un problème plus simple ; cette classe de problème est appelée sac des tâches (BoT). Cependant, nous verrons plus loin que nous n'ignorons pas complètement les dépendances entre les tâches.

La façon dont nous pouvons transformer un programme de flux (une application comme la liaison descendante LTE) en un problème de sac de tâches consiste à répartir les tâches parallèles à chaque phase du programme de flux. Pour le cas de la liaison descendante, nous pouvons assembler une charge de travail avec des tâches indépendantes pour chaque phase, puis laisser le moteur d'allocation décider des nœuds esclaves que chaque tâche exécutera. La réduction du problème à un problème BoT ne signifie pas que nous ignorons complètement les dépendances entre les tâches ; nous les considérons à la granularité de la charge de travail. Une tâche est une fonction nécessitant des ressources informatiques sur un noeud de calcul et elle consomme et produit des tampons. Par exemple, si une tâche B consomme un tampon

qui a été produit par la tâche A, logiquement, la tâche A a été allouée à un nœud esclave dans la charge de travail $W1$ avant l'allocation de la charge de travail $W2$ qui contient la tâche B. Par conséquent, lors de l'allocation de la charge de travail $W2$, nous savons sur quel nœud esclave le tampon produit par la tâche A et requis par la tâche B a été alloué. Fabricio & al. [69] ont démontré que l'allocation de l'application BoT peut être améliorée en considérant les affinités d'entrée du fichier de tâche (en d'autres termes, essayez d'allouer la tâche où se trouvent leurs tampons d'entrée).

L'allocation des ressources a de nombreux objectifs, principalement de maximiser l'utilisation des ressources tout en minimisant le temps d'exécution des tâches (charge de travail) dans le système, ainsi que son impact sur les performances de l'application (faible latence). Nous visons une latence maximale de 50 microsecondes et un impact plus déterministe sur l'application. Ces objectifs nous ont été donnés par le partenaire industriel, comme une valeur adéquate pour assurer la pérennité de la solution et rencontrer les requis des technologies courantes. Le premier objectif est couvert par l'algorithme multidimensionnel proposé pour l'allocation des ressources. Le deuxième objectif est couvert par une implémentation matérielle accélérée massivement parallèle dans le Chapitre 5 définit sur la Figure 4.8. Ainsi, l'approche d'optimisation initialement explorée pour le problème d'allocation des ressources

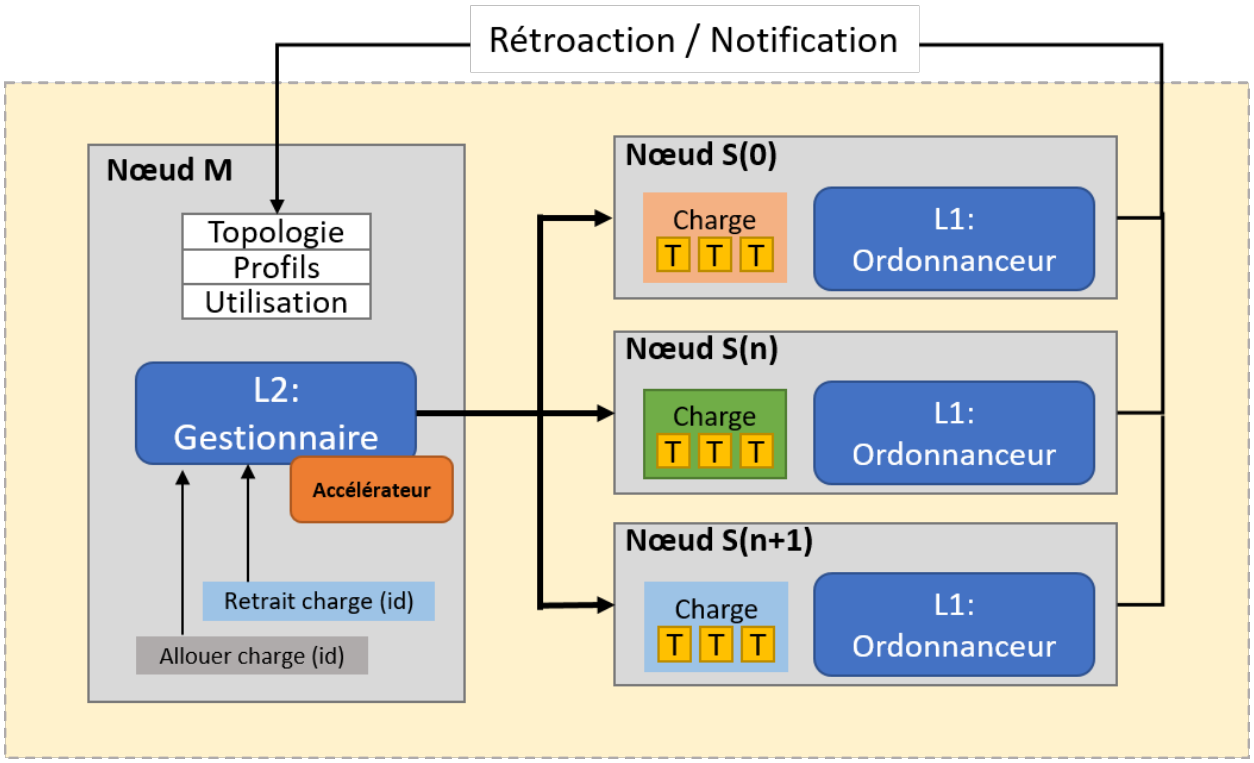


Figure 4.8 Moteur d'allocation de ressources accéléré

est l'approche de programmation dynamique. La programmation dynamique est une approche d'optimisation qui transforme un problème complexe en une séquence de problèmes plus simples ; sa caractéristique essentielle est sa formulation multi-étapes de la procédure d'optimisation. Dans ce cadre, une variété de techniques d'optimisation peuvent être utilisées pour résoudre des aspects particuliers d'une formulation plus générale. Pour chaque problème (ou étape) plus simple, sur la base de l'initiale, nous déterminons une solution optimale.

La prochaine étape de cette recherche vise à définir un cadre où nous trouvons les descriptifs adéquats pour la création d'un algorithme de calcul de coût de charge de travail. Puis, en se basant sur ces métriques, créer un algorithme pouvant être accéléré par une implémentation matérielle. En effet, avec l'implémentation matérielle du gestionnaire des ressources, où la latence ajoutée à l'application est minimisée et prévisible, nous proposons d'accélérer l'algorithme de calcul de coût. L'accélérateur matériel (FPGA) parallélisera massivement l'évaluation des coûts d'une charge de travail dans le système. En supposant que l'évaluation des tâches devient une opération atomique, la complexité de l'algorithme est réduite à $O(1)$, temps constant, où l'algorithme s'exécutera en N itérations où N est le nombre de tâches. La latence de cet accélérateur dépendra du nombre de tâches qu'il pourrait gérer en parallèle et également de la topologie statique du système. Le chapitre suivant entrera plus en détail sur la suite du thème, avec un moteur de calcul de coût accéléré.

CHAPITRE 5 ARTICLE 1 : A CACHE-COHERENT HETEROGENEOUS ARCHITECTURE FOR LOW LATENCY REAL TIME APPLICATIONS

Authors
Michel Gémieux, Yvon Savaria, Jean-Pierre David and Guchuan Zhu Department of Electrical and Computer Engineering at Polytechnique Montréal, Canada E-mail : {michel.gemieux, yvon.savaria, jpdavid, guchuan.zhu}@polymtl.ca

Published in 2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)

Reference as Gemieux, Michel & Savaria, Yvon & David, Jean-Pierre & Zhu, Guchuan. (2017). A Cache-Coherent Heterogeneous Architecture for Low Latency Real Time Applications. 176-184. 10.1109/ISORC.2017.1.

5.1 Abstract

This paper proposes a generic hardware architecture for runtime acceleration of heterogeneous high performance computing (HPC) clusters. This runtime accelerator performs real time resource allocation and management of HPC systems with low latency on multiple time scales. One of the target applications is to perform the signal processing in wireless communication systems such as LTE and 5G over the cloud. A core part of this work is to develop and characterize algorithms that can distribute workloads to server blades in a balanced manner with the aim of maximizing processor utilization in computing clusters. Resources are also managed to guarantee bandwidth for data transfer between computing nodes and reserved cache memories to enable deterministic task execution. This paper shows how a workload distributed among several server blades can be scheduled at a finer time scale than what a normal software implementation would allow in order to minimize the makespan required to complete execution of sets of tasks. A case study is conducted on the implementation of a resource allocator for the proposed platform. A 760-time acceleration factor of the resource allocation process has been achieved compared to a pure software implementation, whilst enabling data transfers at the nanosecond scale. It stands as a proof of concept that confirms the viability of CPU-FPGA platforms for wireless standards virtualization.

5.2 Introduction

Nowadays the amount of data used and transferred by information systems keeps increasing, and the need for faster computing follows the trend [70]. The stagnation of the processors operating frequency, related to power constraints and leakage current problems, pushed hardware vendors to increase the amount of cores in their commercial offering. This boosted the interest for heterogeneous computing with very parallel accelerators paired with x86 compatible processors that are modestly parallel. Mirroring processor evolution, computing clusters kept increasing the amount of action units (server blades) pushing them towards the era of High Performance Computing (HPC) [71].

The same need motivated the development of heterogeneous clusters in order to fulfill the demand [72], using accelerators (CPU/GPU/DSP) to do specific types of computing work [73]. The so-called “Big Data" puts emphasis on the amount of data being processed at a given time. Moreover, several applications have requirements for very low latency and guaranteed performance, while simultaneously requiring high throughput. If the need for HPC with low latency is clear now, it has not been studied as extensively as mainstream HPC in the last years.

These requirements motivated mainstream compute hardware vendors to develop and offer commercially socketed FPGAs. This market is still in its infancy, but with big players like Intel [74] leading the way to integrate FPGAs in HPC settings, the situation could change rapidly. Promising as they are, it remains to be seen how and how much these socketed FPGA processors will deliver on their promises. Thus the work reported in this paper will contribute to validate the viability of FPGA accelerated HPC platforms.

This paper aims to develop one of the first generic hardware low latency runtime accelerators. The hardware architecture is a complete solution for dynamic computing resource allocation and management in heterogeneous computing clusters in real time. The main goal is to reduce the additional latency introduced by the runtime while maximizing the processing units utilization.

To deal with the challenges of constantly increasing information exchange, the Information and Communications Technology (ITC) industry is pushing for the centralization of cellular standards processing that is performed according to established standards, such as Long Term Evolution (LTE) and those that are emerging in relation to 5G [75], through the use of techniques such as Software Defined Radio (SDR), Software Defined Networking (SDN), and Network Function Virtualization (NFV).

An FPGA-multiprocessor heterogeneous architecture is proposed in this work. This archi-

tecture enables the virtualization of wireless protocols on “commodity” type servers. In line with this technological trend, this work is at the intersection of the cloud computing and telecommunication systems. It is proposed in [6] to use a cloud-based infrastructure, called Cloud Radio Access Network (C-RAN), in order to have an energy-efficient wireless network for cellular signal processing. However, this kind of architecture requires a high bandwidth low latency platform, in order to meet expected deadlines of the new cellular standards. One of the target applications is the computing platform designed to support wireless communication standards, such as LTE and upcoming 5G technologies. Table 5.1 exposes the latency that future 5G standards are expected to meet.

Applications related to LTE and 5G are the primary focus of this research. However, this paper targets the fundamental problem of Runtime latency reduction to improve data centers efficiency. The proposed hardware architecture for runtime acceleration is generic. Considered as a major step in High Performance Reconfigurable Computing (HPRC), Intel is already providing Integrated FPGAs in CPU packages for data centers use, as explained in [74]. Intel predicts that 30% of all data center will use FPGAs by 2020. Our research is thus a precursor to runtime acceleration for data centers, and eventually for consumer computers, when the CPU-FPGA platforms become mainstream for their versatility. The proposed generic hardware architecture could have a significant impact on the way FPGA acceleration will be used.

The rest of this paper is organized as follows. Section II goes over some related works and reviews important recent efforts that summarize the heterogeneous HPC cultures in relation to which we can better expose our contributions. Section III dives into the block architecture of the proposed Cache Coherent Heterogeneous Architecture. Section IV presents a case study, where we implement a resource allocator for 5G virtualization on the proposed accelerator platform. Section V exposes the results and analysis confirming the feasibility and validity of the proposed hardware resource allocator (HRA). Finally, section VI concludes and offers some suggestions for future work.

Tableau 5.1 5G expected requirements

Parameter	Value
Transmission Bandwidth	20 MHz
Latency in the air link	under 1ms
Latency end-to-end (device to core)	under 10 ms
Energy efficiency	over 90% improvement over LTE

5.3 Related Works

5.3.1 FPGAs in HPCs

This section presents a brief analysis of the low latency accelerators in HPC settings, more specifically FPGAs. Different accelerators have different strengths. The main types of algorithms which run on HPCs are classified as the 13 dwarfs [27]. We observe that heavy arithmetic computation combined with irregular memory access is more efficient on **FPGAs**. We can deduce that FPGAs are inherently better for streaming type applications. The work reported in [28], exposes FPGAs as function accelerators. Where a given application or function runs on it and returns the information to a main processing unit. Our proposed architecture interacts in a similar way with a multi-core CPU. In order to improve the latency bottleneck, communication between CPUs and accelerator needs to improve. Most HPCs containing FPGAs connect all their accelerators through PCIe, that was mainly designed to favor bandwidth over latency. Numerous studies and systems use a PCIe based communication protocol [32].

Using FPGA accelerators through PCIe communication is not new in the HPC realm, Intel and Microsoft [35] successfully integrated FPGA acceleration in their processing flow for massive data analysis for their Bing research engine. Little attention was put to latency issues in HPC to favor massive data processing. We will explore the latency problem in streaming application in the next section of this paper. This is where socketed FPGAs come into action. Considered as the next important step of HPRC, as discussed in [36] and [37], FPGAs enable lower latency communications between processing units. Indeed, the concept of "Soft processors" is already available on FPGA platforms, such as Nios, Microblaze and Zynq with ARM processors [40]. These processors remain hard to integrate with low latency in an HPC cluster. Socketed FPGAs can solve this issue by reducing the communication latency with the use of the Front Side Bus [41]. The point-to-point communication between the CPU and the FPGA provides developers with the opportunity to have better access to system memory [22]. Although it offers 6 times lower latency than PCI-X communication, the latency is still too high for 5G virtualization. In order to meet 5G requirements, [41] and [22] implementations lacked a specific feature, a cache coherent interface (CCI).

The first cache coherent FPGA/CPU communication was introduced by Intel, with the Quick Path Interconnect bus [22]. The contribution of the present work is based on the use of such a feature. The communication latency is the time required to write and read data in cache. In-socket FPGAs enable low latency and high throughput processing, useful for implementating efficient multi-level resource allocation algorithms. As our work is at the junction between two fields, we will compare the efficiency gains to HPC with FPGA accelerators and hardware

resources allocators. Unfortunately, little work has been published in our niche, due to low availability of any in-socket QPI FPGA platform.

Our hardware architecture will enable the deployment of complex scheduling and resource allocation algorithms, which will ultimately accelerate the runtime of a heterogeneous HPCs.

5.3.2 Software runtime

Our contribution in this paper relates to the so-called "runtime" and runtime systems. The term runtime, as used in this paper refers to the software and hardware systems, such as memory and resources management, and hardware acceleration for task execution, in a heterogeneous system.

Runtime acceleration is mostly studied in the realm of operating systems (OS), where resource management is critical for an efficient system. The runtime efficiency is tipically measured by its latency and resource utilization.

The typical way to get a lower latency system is to use aReal Time Operation systems (RTOS), such as FreeRTOS [43]. It is explained in [44] how to reduce the latency imposed by resource management with smart assembly code insertions at the kernel level. However, a problem associated with RTOS is their rigidity and the lack of support for new computation libraries, making them less suitable for newer applications, especially for HPC applications. Apart from the use of RTOS, the common ways to accelerate a runtime are through better **memory management** mechanisms, suitable **application specific** scheduling algorithms and resource allocation schemes, and **new architecture**.

For heterogeneous systems, there exist software alternatives which use sets of libraries for specific applications. More recently, StarPU, a task programming library for hybrid architectures, was introduced in [46] to tackle runtime concerns for heterogeneous architectures.

Runtime improvements through memory resources management are heavily investigated, because it is considered as a main bottleneck of runtime systems. There are two main levels at which memory management is critical, namely cache and DRAM. A complete cache management framework is proposed in [47] for real-time systems in order to reduce cache misses, which reduces the access time to DRAM and ultimately reduces the runtime execution latency.

We did not find hardware runtime acceleration methods in HPCs. It is worth noting that most of the solutions for hardware acceleration in HPCs are relevant to massive data computation, but not latency awareness.

However, a lot of research has been done on the hardware implementation of scheduling

algorithms. We demonstrated in our previous work [7] that using StarPU with adequate scheduling algorithms enables the virtualization of latency hungry applications, such as LTE on a given platform.

We aim in this work at going a step further and address the possible advantages of hardware implemented scheduling algorithms. Specifically, we plan to leverage hardware acceleration to improve the latency on a much larger scale than what is reported in [42] where a RTOS on Xilinx Zynq is used.

In terms of architecture, a reconfigurable computing and runtime systems for HPCs (ECOSCALE) are exposed in [53]. This system consists of a programming environment and a hardware architecture aimed at reducing data traffic and latency in a cluster, by using reconfigurable hardware accelerators communicating with MPI protocol. Due to the “niche" research field we are in, the results reported in [53] are the ones with which we will compare to. However, the ECOSCALE [53] is just a theoretical implementation, and it remains to be empirically validated.

Hardware acceleration of resource management function is not common in HPC due to the latency induced by the accelerator's communication (i.e. PCIe). The platform on which we work provides us with the opportunity to contribute to the new field that is gaining in importance.

We propose a software approach and a hardware architecture for runtime acceleration in heterogeneous HPC environments. Once the generic architecture exposed, we will dive into a case study implementing a hardware resource allocator on the proposed architecture.

5.4 Context and Proposed Architecture

This section introduces the proposed Cache Coherent Heterogeneous Architecture (CCHA). It begins by describing its overall hardware and software aspects. Then Section 5.4.2, presents the essential blocks of the CCHA.

5.4.1 Overall System Architecture

CCHA aims at resolving latency shortcomings in HPRCs. Its overall architecture is shown in Figure 5.1. While at that level of abstraction, the architecture is similar to some existing systems, it differs as shown later due to the specifics of how it is implemented. Figure 5.2 is a photograph of the actual prototype platform. The system has an in-socket FPGA. Other in-socket FPGA architectures have been reported [76]. However, in the proposed FPGA-



Figure 5.1 CCHA Platform Architecture

CPU subsystems implementation, these components are connected with Intel's QPI bus [77]. Such connection enables cache coherent communication between CPU and FPGA entities. Coupling the CPU to the FPGA through QPI enables low overhead communication. More specifically, QPI reduces latency and interrupts overhead compared to other communication mechanisms such as PCIe. In a nutshell, coupling a CPU to an FPGA through QPI enables communicating with relatively low overhead.

As Figure 5.1 shows, the accelerator is interfaced with a multi-core general purpose processor running a standard version of Linux. The selection of a generic version of Linux is motivated by having the opportunity to use the latest software libraries for any application. A generic Linux is more flexible than a RTOS that typically offers low compatibility with the most recent libraries and techniques. This allows using the Accelerator Abstraction Layer SDK (AALSDK) from Intel. It also allows using standard APIs to interface with hardware accelerators that can be implemented in the FPGA. In our case, we used a combination of multiple libraries coupled with AALSDK version 3.3. One end goal of the proposed architecture is to use the FPGA as a service. The FPGA being a massively parallel architecture, it should be used accordingly to accelerate system functions and supported applications. A simple, yet efficient way to use the FPGA as a service will be briefly explained in Section 5.4.2.

On the software side of the implementation, we consider a simplified version of the task allocation problem where tasks are independent. This class of problem is called bag-of-tasks (BoT) management [78]. However, even if tasks are partly treated as if they are independent from each other, as shown later, dependencies between them are not completely ignored.

Streaming type programs (such as LTE/5G) are transformed into bag of tasks problems by

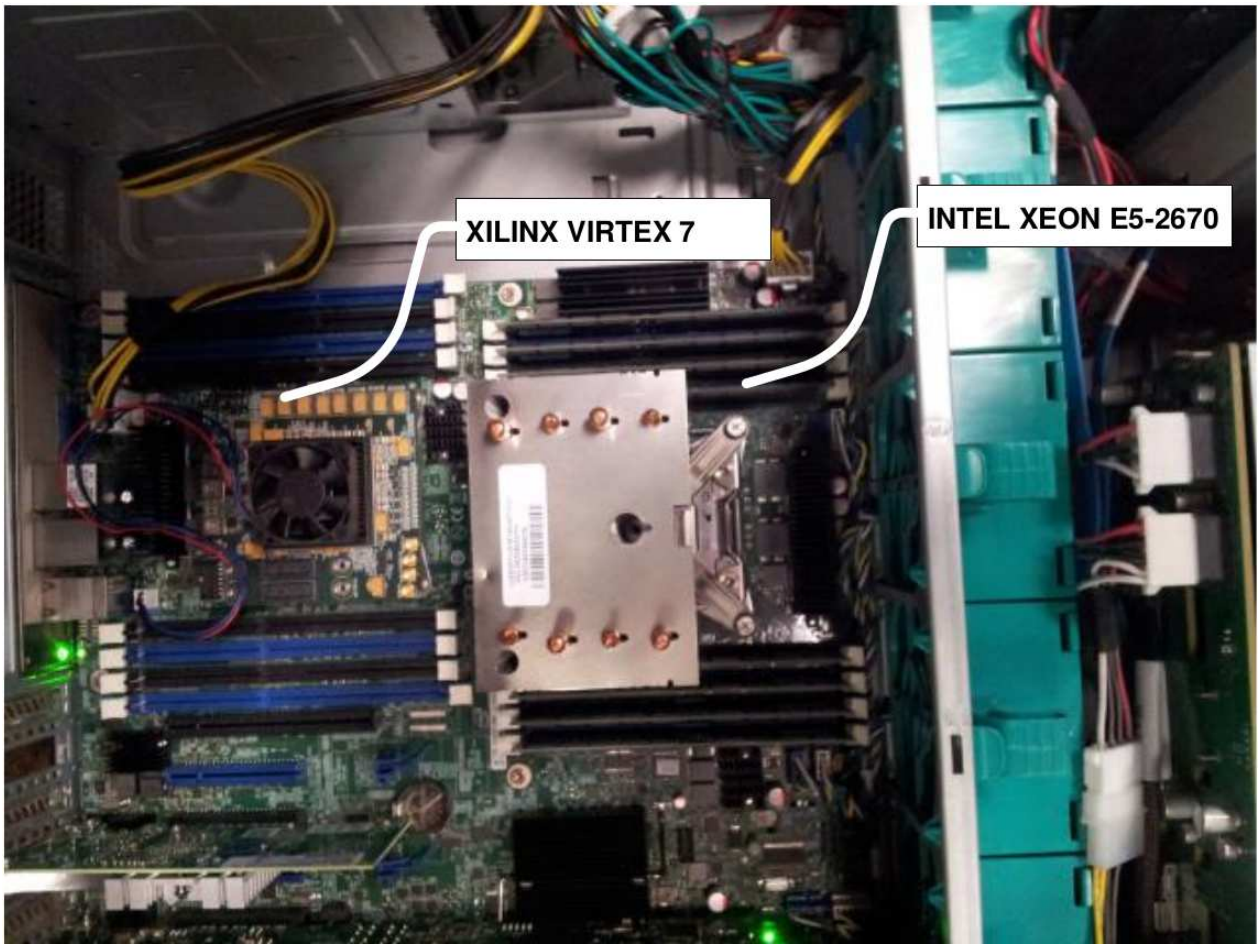


Figure 5.2 CCHA Platform Picture

dispatching several parallel tasks in parallel at each phase of the streamed program.

A task is a function requiring computing resources on a compute engine that consumes and produces data buffers. For example, assume Task B consumes a buffer that has been produced by task A. Logically, Task A has been allocated to a slave node in workload W1 before the allocation of workload W2 that contains Task B. Consequently, at the allocation of workload W2, the system knows which slave node holds the data buffer produced by Task A that is required by Task B. It is shown in [78] how the execution of a BoT application can be improved by considering task file input affinities (in other words, tasks should preferably be allocated where their input buffers are located, if possible).

In order to enable an effective communication between the CPU and FPGA subsystems, the Intel's QPI Endpoint is used. It implements the physical layer of the QPI bus on the FPGA side. The following section explains in more details the hardware architecture proposed in

this paper.

5.4.2 Proposed Hardware Architecture - The CCHA

The proposed CCHA is a hardware architecture for real time low latency applications. The generic architecture offers application acceleration at a fine granularity, with the use of "active system tracking" and low latency cache coherent communication. This method, paired with a high throughput low latency bus, allows for real time function acceleration. CCHA is first described as the combination of the following elements called : Cache Coherent Interface, Requestor, Arbiter, Information Matrix (Memory), Tracer and Accelerator. The relationship between these blocks is partly exposed in Figure 5.3.

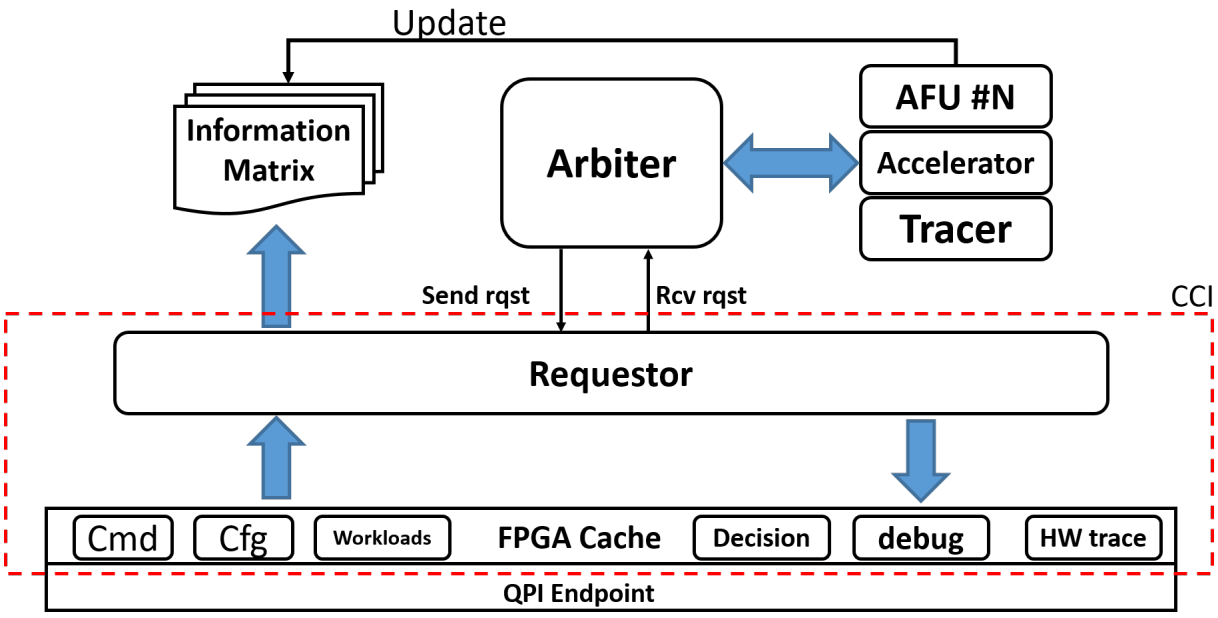


Figure 5.3 Proposed Architecture - Block diagram

Cache Coherent Interface (CCI) : It is a generic synchronous cache interface that manages communication between a QPI Endpoint on the FPGA and compatible drivers on the processor.

Requestor : It acts as a controller from and to the CCI module. It takes care of the handshaking between the two systems. It sends and receives address indexes from the arbiter block and sends requests to the CCI block.

Arbiter : It partly acts as a demultiplexer. It is also a decision layer, interacting with the Requestor to send and/or receive the right information to the right entity on the HW

architecture. Each Acceleration Function Unit (AFU) is assumed to have a specific address to which information is read and written.

Information Matrix : This memory module stores all the information needed by the CCHA. It is a multidimensional data array that captures the important system characteristics for active tracking of an application and how it is executed. The Information Matrix is divided into three main parts called : Topology, Utilization Matrix and Task Profile Matrix that will be further detailed in the following paragraphs.

Topology : It is a vector representing a static view of the whole system. Thus it is a multi-dimensional data array, ordered as a master/slave structure. This vector specifies computing nodes properties, the type of workers (flavor), the type of tasks they can execute, the interconnect topology, and the respective speed of each module.

Utilization Matrix : It represents the most up-to-date state of the overall system. The Utilization Matrix is updated in real time, reflecting the use of all the workers in the system. For demanding real-time applications acceleration, a good tracking of the cluster utilization is mandatory to produce effective task assignment to each worker.

Task Profile Matrix : It receives workloads from the multi-core subsystem. A workload has a given amount of tasks, specific to the application. A typical profile vector contains a task ID as well as all the information representing a task, such as its Worst Case Execution Time (WCET) for each compatible type of worker.

Tracer : It is a non-intrusive hardware implemented monitoring instance. It snoops the CCHA behavior on the FPGA side, without interfering with other modules.

Accelerator : This block is application specific. It implements with FPGA fabrics the function to be accelerated on the CCHA.

The CCHA is designed to offer low-latency hardware acceleration. Our architecture coupled with a multi-core processor, allows to use the FPGA as a service. Its request based communication gives the CCHA the ability to handle multiple clients, as long as the addressing is correct. The typical flow of a CCHA accelerated application would be as follows. The software application on the multi-core subsystem sends a request to be connected with modules on the FPGA. The Handshake Controller acknowledges the request and points the application to a predefined address space in cache. Then the software portion on an application sends a cache line sized command to configure the addresses for all the AFUs. Once the configuration is done, the FPGA receives tagged workloads. At that point the arbiter dispatches the right workload to the right Accelerator for processing. The result, or decision produced by the accelerator is then written in cache for the SW to read. Writing back the solution is done

in a similar manner, by sending requests to the Arbiter, then the Requestor can access the cache.

The efficiency of the CCHA is demonstrated through a case study developed in Section 5.5.

5.5 A Case Study

This section demonstrates the efficiency of the CCHA on a real-time, latency aware application inspired by C-RAN. In this application, a bag of tasks must be allocated with low latency. This is done with a hardware implemented Resource Allocator, which is based on the Max-Min Fairness (MMF) algorithm. The rest of this section compares a software based resource allocator with its hardware implementation mapped on the CCHA. This case study stands as a demonstration of the efficiency and viability of the CCHA.

5.5.1 Max-Min Algorithm

Max-Min Fairness load balancing is commonly used for network resource allocation problems [55]. It stands as a solution to achieve a fair resource utilization when a set of tasks needs to be allocated to multiple workers. MMF assumes that each worker has an equal right to the resources, but different workers can have a different load. MMF results in higher average throughput and tends to an efficient utilization of the workers. Therefore, the utility of a worker with low processing power will be maximized before that of the workers with higher processing power. Successful load balancing leads to an allocation such that each worker never receives more than what it can process at a given time.

This algorithm was chosen mainly because it allows highly parallel implementations. It is also of interest that MMF is compatible with different technologies, such as FPGAs, GPGPUs, and DSPs [58]. For this case study, we implement a simple, yet efficient version of the MMF algorithm. It allocates workloads onto heterogeneous blades in a cluster to minimize the makespan with a fair allocation scheme. The following subsections explore in some depth the context, theory and RTL implementation of an MMF workload balancing algorithm.

5.5.2 Software Tests

In order to quantify the speedup given by the resource allocator on the CCHA, a software model of the Max-Min resource allocator was developed. The model was built on top of the following libraries :

— Intel's Data Plane Development Kit (DPDK), for memory management

— StarPU, for the preset API controlling the scheduling,

— Streamit, for its streaming APIs.

The pseudo-code of the application is listed in Algorithm 1 :

Algorithm 1: Max-Min pseudo-code

```
1  W : Workload with N tasks in t_i index by i;
2  ws_j : Computed workload for M nodes in the system;
3  T : Topology Vector;
4  R : Vector of current Resource Utilization;
5  P : Task Profile;
6  while W not empty do
7  |  foreach task t_k in W do
8  |  |  (ws_idx[k], min_cost[k]) = min( task_cost (t_k, P, R, T));
9  |  end
10 |  (slave_id,selected_task) = max(min_cost);
11 |  remove selected_task from W;
12 |  add selected_task to worker_id ws;
13 |  Add consumed resource from selected_task to R;
14 end
```

To the best of our abilities, we followed the pseudo code above and implemented the resource allocator in C. The code was tested on a system populated and configured with :

— One Intel's Xeon E5-2670v3, a 12 core 2.3GHz CPU ;

— 64 Gb of DDR4 RAM ;

— Disabled hyper threading to maximize determinism ;

— Ubuntu 14.04 LTS ;

— A specific *grub config* (isolcpu) to maximize the workers availability while restricting the OS to one core. This is done with the snippet of code presented below.

```
BOOT\_IMAGE=/vmlinuz-3.2.0-67-generic
root=/dev/mapper/INTEL--FPGA-root
ro crashkernel=384M-2G:64M,2G-:128M
nomce isolcpus=1,2,3,4,5,6,7,8,9,10,11
```

All the data required for the resource allocation engine is stored in memory. Using DPDK and the Grub configuration, one isolated CPU core runs Linux has a host, sets the memory pools in which the data would be placed and then cleared. Using an MPI call, the software implementation of the resource allocator application (Max-Min) is started on the remaining cores. With this software implementation, a run time of 6.4 milliseconds is needed to execute the algorithm.

5.5.3 FPGA Max-Min Implementation

As mentioned in Section 5.5.1 the Min-Max algorithm is used because it allows massively parallel implementations. A RTL implementation equivalent to the algorithm listed in the pseudo-code 1 is sketched in Figure 5.4.

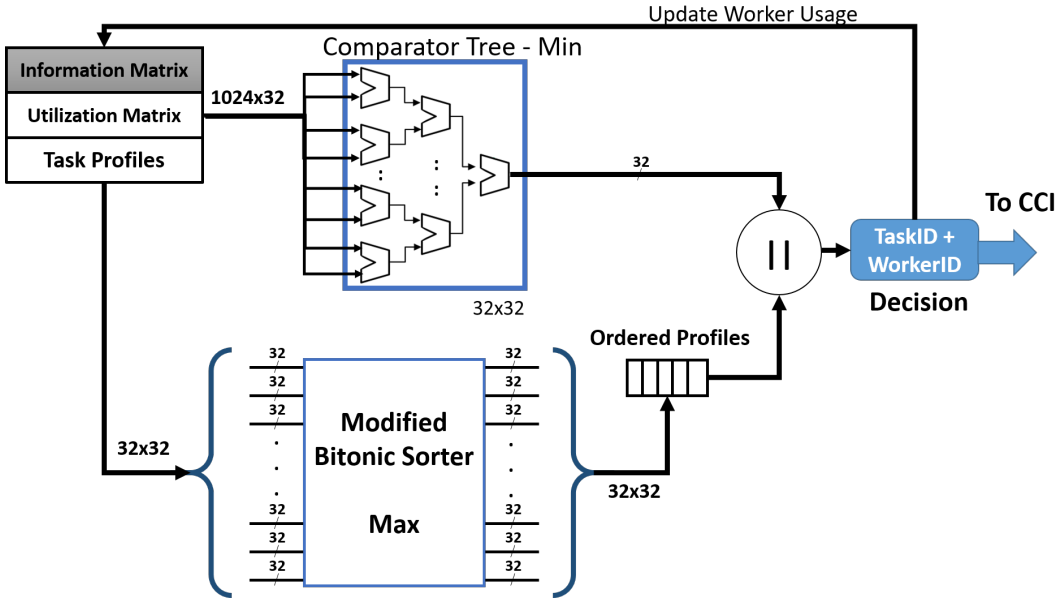


Figure 5.4 Max-Min block architecture.

The datapath computes, for each task the optimal worker in a multi-CPU cluster that should execute that task. In fact, it computes the impact of that assignment on the cluster. The resource allocator reported here is designed to support a cluster of 16 double socketed blades, each socket populated with a 16-core double threaded CPU. It gives each task 1024 assignable workers (one thread on one core), at any given moment. The assignment of one task to a specific worker is called a decision. To make a decision, the current datapath passes the task profiles through a Bitonic sorter, and the workers utilization vector through a comparator tree. The data format of the vectors processed to make a decision are (B :byte, b :bit) :

- Task Profile Vector (19B) : TaskID/16b - FunctionID/4b - inout data/116b - WCET/16b
- A Workload(19 456B) : 1024 Task Profiles
- Utilization Vector (one Worker) : BladeID/4b - SocketID/1b - WorkerID/5b - CurrentUsage/16b
- Decision Vector : TaskID/16b - BladeId/4b - SocketID/1b - WorkerID/5b - Expected_Exec_Time/16b

So a Decision is a pair composed of a task ID from the max sorter and the IDs of the fastest

current execution worker for a given task.

The Max-Min resource allocator is best described as the combination of its following parts called : Information Matrix, Modified Bitonic Sorter, Decision Tree (Comparator Tree).

The **Information Matrix**, as explained in Section 5.4.2 is registered data array, readily available in one clock cycle (5 ns in the current implementation). It contains a 1600 bytes Topology, an equivalent Utilization Matrix and a 19 456 bytes Workload (1024 Task Profiles). It was chosen to register all the Information, to be able to access multiple vectors as fast as possible.

The **Modified SPIRAL Bitonic Sorter** [79] is a 32 input and 32 output block. It takes as input 32 Task Profiles per clock cycle. It orders the 1024 reduced Task Profiles (TaskID + WCET = 32b) in 318 cycles (latency), and outputs the ordered Task Profiles 32 at a time.

The **Decision Tree** is a typical 11 stage comparator tree that reduces the 1024 assignable worker to one. So, the Decision Tree has an 11-cycle latency. It inputs 1024 values and outputs one per clock cycle.

Once the pairs of task and workers with minimum impacts (total execution time) are found, the algorithm dispatches the tasks in an orderly way, from those that have highest to the lowest impact on the system. The metric considered so far is additive. It measures the cumulative impact of a task's WCET on a given worker (current usage).

The complexity of the Hardware Resources Allocator algorithm is well represented by the fact that for each workload, it needs to compute and find the minimums of 1024 guaranteed completion times (based on WCET) of tasks to be executed on 1024 assignable workers. For each Workload, a minimum of 1,048,576 possible values (WCET + Current Worker usage pair) could be evaluated. However, using a combination of a Max sorter and a Min reduces the complexity to two sorted queues, where their respective 1024 minimum completion times must be ordered, starting by the maximum impact value.

A simple way to implement the behavior previously described is with a sorter and a Decision tree. More specifically the Max-Min FPGA implementation is mostly made of two parts : the **Max** and the **Min**.

The **Max** module is a SPIRAL Bitonic Sorter [79] modified for this paper. It orders Task Profiles from the longest to execute to the fastest, while keeping the profile information of each task (Modification for persistent TaskID while sorting per WCET). The modification allows separating the task ids from the actual values of the tasks. Once through the Max bitonic sorter, the datapath sorts the list of task IDs from maximum to minimum.

The **Min** is implemented with a comparator tree. The tree determines at any given moment

the lowest impact worker to which a task should be associated. It fetches 1024 Utilization Vectors from the updated registered Utilization Matrix in one cycle. The Utilization Matrix is updated with each new Decision (with the expected execution time of the task assigned to the worker).

The datapath outputs one Decision per cycle, after its basic latency. The value of that latency is evaluated below. Section 5.6 summarizes the results obtained with a Max-Min implementation on the CCHA.

5.6 Results and Discussion

This section reports various significant results related to complexity, throughput and latency. All the experimental results were obtained on the platform described in section 5.4. The specifics of the system on which the tests were made are :

— CPU : Intel's Xeon E5-2670
— FPGA : Xilinx's Virtex 7 XC7VX485T
— OS : Ubuntu LTS 14.04
— Grub config as defined in Section 5.5.2.

5.6.1 Resource Utilization

Table 5.2, summarizes the Virtex 7 resource utilization used by the Max-Min resource allocator implementation. The first line, called Virtex 7, summarizes the total available resources on the platform that was used. Those resources are usually reported in 5 categories, Flip-Flops(FFs), Look-Up-Tables (LUTs), Block RAMs(BRAMs), Gigabit Transceivers (GTs) and DSPs (DSP48) not used in the current implementation. The last line is the utilization percentage of each type of resource.

Tableau 5.2 HRA Area report

Logic block	FFs	LUTs	BRAM	GTs
Virtex 7 XC7VX485T	607 200	303 600	2 060	56
QPI Core	55 300	40 575	185	21
Bitonic Sorter (Max)	109 599	112 839	80	0
Comparator tree (Min)	3 233	2 727	0	0
Remaining	439 068	147 459	1 795	35
Utilization (%)	27.69	51.42	12.86	37.5

As shown in Table 5.2, most of the resources required by the MMF FPGA implementation are taken by the modified SPIRAL Bitonic sorter. Depending on the amount of tasks sorted

at once, the size of the bitonic sorter will either decrease or rise. Reducing the input set size in conjunction with the streaming width greatly impacts the resource required to implement the sorter.

Figure 5.5 shows a successful logic simulation of the resources allocator. It shows results obtained at different stages of the algorithm. Two zones can be distinguished, the top left represents the configuration of the HRA, and the middle is the bitonic sorter in action. The dense lines under middle are the Decisions made at each cycle. Note that the complexity strongly depends on the size of the bitonic sorter and the amount of data to be processed.

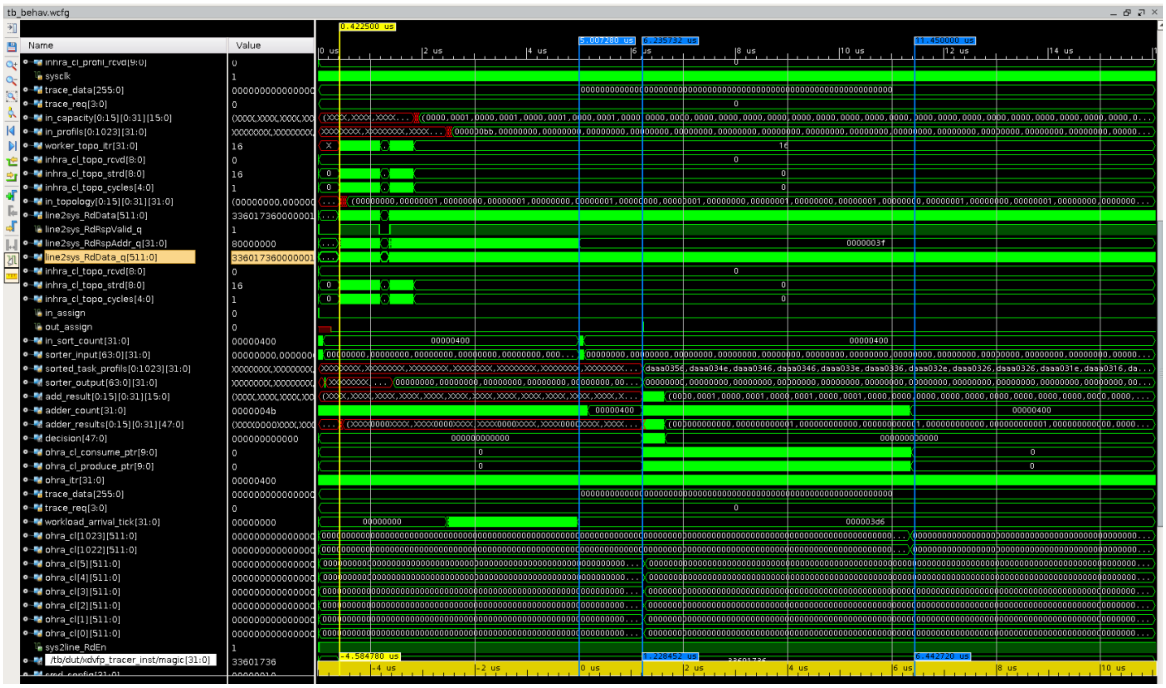


Figure 5.5 Simulation Results

The result of placement and routing is shown in Figure 5.6. This layout confirms the success of our efforts to keep the footprint of the implementation as small as possible. Another objective that was pursued successfully as confirmed by Figure 5.6 was to obtain, reproducible density and performance with the use of strategically placed “pblocks”. An important part of the implementation is the phy_block location. It must be placed on the edge with 20 accessible high-speed transceivers to meet the QPI latency and throughput requirements. By contrast, a random mapping of the QPI reserved transceivers would significantly affect the overall performance, and would have resulted in timing discrepancies of various signals in the QPI bus.

According to Table 5.2 and Figure 5.6, the combination of the QPI interface and HRA

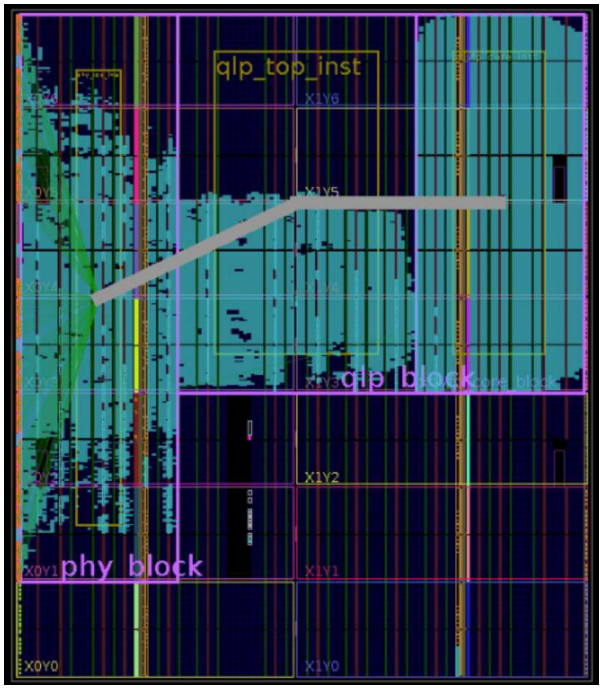


Figure 5.6 Implementation Results

respectively occupy 28%, 52% and 13% of the FFs, LUTs and BRAMs, leaving plenty of resources for implementing other AFUs in the same FPGA. It is remarkable that a good part of the reported complexity come from the QPI core itself and could not be compressed.

5.6.2 Latency Analysis

The total latency induced by the system depends on communication and logic latencies. The following analysis is based on a 200 MHz base clock that was successfully obtained with the implementation shown in Figure 5.6. Based on this clock rate the performance of the HRA-CCHA combination can be evaluated.

First, communication latency on the CPU side is due to cache line access. The Xeon E5 uses Intel's Quick Path Interconnect that yields a doubly pumped data rate of 6.4 GigaTransfers per second (GT/s). The typical latency of a 64-byte packet sent through QPI is 5.4 ns [77]. Note that newer versions of the Xeon E5 chips have a higher data rate at 9.6 GT/s, which would reduce the latency overhead of the CPU-FPGA communication through the QPI link.

Second, the configuration latency needs to be evaluated. The first block of information to transfer is the Topology. It is 1600 bytes long. At 64 bytes transfers per cycle with QPI, 25 clock cycles are needed to transfer the Topology. Similarly, 304 cycles are required for a Workload transfer through the QPI bus. Thus, the configuration latency is given by :

$$L_{Cfg} = L_{Topology} + L_{Workload} = 25 + 304 = 329\ cycles \tag{5.1}$$

The total architecture latency can be evaluated in the same manner. To get the minimum latency of the system, the Bitonic Sorter and Comparator tree latencies are added to the equation. It is as follows :

$$L_{min} = L_{Topology} + L_{Workload} + L_{Sort} + L_{Comp} + L_{Decision} \tag{5.2a}$$

$$L_{min} = 25 + 304 + 318 + 11 + 1 = 659\ cycles. \tag{5.2b}$$

Equation 5.2 shows that the basic system latency (to get the first decision) is 3.295 microseconds with 1 new decision per 5 ns cycle for 1024 cycles. To have the 1024 sorted decisions at the end of the HRA, the whole application would take 1683 cycles, or about 8.415 microseconds at 200 MHz.

Compared to the software implementation reported in 5.5.2, the HRA offers an acceleration factor of **760.5**.

Note that the current operating clock is limited at 200 MHz by Intel's QPI SoftCore. By contrast, the rest of the HRA logic can operate at 450 MHz. Using that frequency would improve the overall system latency **2.25x**.

5.7 Conclusion and Future Work

This paper proposed a generic heterogeneous hardware architecture for low latency real-time applications. Such an architecture allows accelerating latency sensitive applications with stringent real time latency constraints.

The reported results demonstrated that : (1) it is possible to have a usable hardware acceleration framework in a commodity server, whilst keeping low latency communication ; (2) a heterogeneous FPGA system appears to be a viable technological option for the future.

A case study has been carried out to validate the claims regarding the effectiveness of the proposed architecture, in which an implementation of an MMF hardware resource allocator is put on the CCHA. The HRA is able to allocate a workload (1024 tasks) to 16 blades containing a total of 1024 workers in less than 8.5 microseconds. Finally, it is shown that the hardware implementation is **760x** faster than a functionally equivalent software implementation.

As part of our future work, it is planned to experiment with OS integration of the proposed architecture for low latency scheduling purposes. Furthermore, the suitability of the resource allocator as means to virtualize a 5G stack using a heterogeneous will be investigated.

The results of this research will be used as a foundation to investigate application acceleration with heterogeneous HPC clusters and effective means to perform scheduling and workload management. This paper offers starting points toward runtime acceleration for data centers, and eventually for consumer computers, if CPU-FPGA platforms become mainstream.

5.8 Acknowledgements

Thanks to Natural Sciences and Engineering Research Council of Canada for financial support and Huawei Technologies for technical guidance and for financial support.

CHAPITRE 6 ARTICLE 2 : A HYBRID ARCHITECTURE WITH LOW LATENCY INTERFACES ENABLING DYNAMIC CACHE MANAGEMENT

Authors
Michel Gémieux, Yvon Savaria, Jean-Pierre David and Guchuan Zhu Department of Electrical and Computer Engineering at Polytechnique Montréal, Canada E-mail : {michel.gemieux, yvon.savaria, jpdavid, guchuan.zhu}@polymtl.ca

Published in IEEE Access

Reference as Gemieux, Michel & Li, Meng & Savaria, Yvon & David, Jean-Pierre & Zhu, Guchuan. (2018). A Hybrid Architecture with Low Latency Interfaces Enabling Dynamic Cache Management. IEEE Access. PP. 1-1. 10.1109/ACCESS.2018.2876597.

6.1 Abstract

The main focus of the dominant technologies in the high performance computation (HPC) market, such as GPU and multicore systems, is put on processing power, while much less attention has been paid to communication delays inside hybrid architectures. To fill this gap, this paper presents an experimental study on Intel's Broadwell Xeon multicore processor with integrated Arria 10 FPGA capabilities to characterize the communication delays between CPUs and the FPGA, using both the low latency cache coherent interface, and the two PCIe links offered by this platform. The obtained results show that an FPGA cache access latency can be as low as 25 cycles at 400 MHz and that the platform is capable of reaching a bandwidth over 20 GB/s using an aggregate of the three available links. Furthermore, an FPGA-based cache management mechanism is proposed and implemented in this work. A case study on a Merkle tree hash function shows that a hardware accelerator can achieve a 5-fold data access acceleration in the worst case scenario. This scheme takes advantage of the QPI cache coherency and queuing theory to achieve a low latency and efficient memory management. In addition, design recommendations regarding the use of the CPU-FPGA platform for the implementation of fine-grained memory management schemes are suggested.

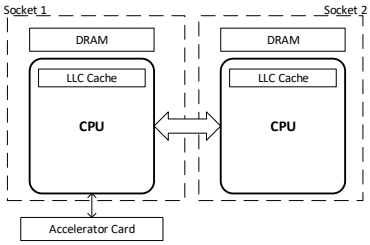


Figure 6.1 Accelerator Card

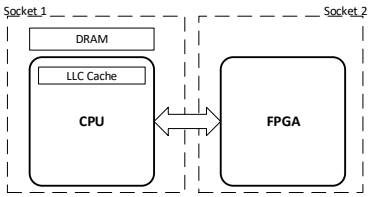


Figure 6.2 Socketed FPGA

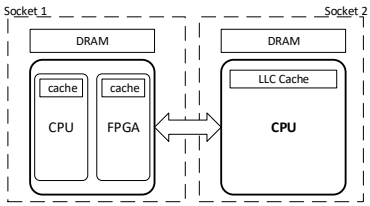


Figure 6.3 MCP

Figure 6.4 Accelerator Architectures

6.2 Introduction

The advent of "Big Data" increased the demand for high performance computing systems to handle an overwhelming amount of information circulating in data centres around the world. Due to the volume of the data consumed at any time, throughput aware acceleration remains a bottleneck in HPCs [70]. This need motivated the development of heterogeneous computing clusters, which are mostly populated by accelerator cards, powered by GPUs, DSPs, and more recently add-on cards with large Field Programmable Gate Arrays (FPGAs). Some mainstream systems have been deployed by leading industrial companies, such as Amazon, which provides FPGA-accelerated clusters [80] for energy efficient and cost effective data processing.

Meanwhile, new applications may introduce new concerns that need to be addressed. Deep learning, for instance, stresses two prevalent requirements, namely the need for high me-

mory bandwidth during the training of a neural network [81] and the need for low latency calculations for the inference of a trained application [82]. Other applications, such as High Frequency Trading (HFT) and 5G communication systems, impose also strict latency constraints [83]. There is no doubt that one of the key technological bottlenecks to solve is to guarantee low-latency and high-performance communications, while providing a high throughput.

Considered as a major step in High Performance Reconfigurable Computing (HPRC), Intel provides a platform with integrated FPGAs in CPU packages for data-center-based applications, called Multi-Chip Package (MCP) [84]. Specifically, this platform features a cache coherent interface between the CPU cache and a memory structure on the FPGA used as cache, which gives the opportunity for low latency hardware acceleration through its proprietary links. The cache coherent MCP allows for a cache-based communication with the FPGA. Such an interface enables cache manipulations of x86 processor caches, which is usually reserved to inaccessible hardware system interfaces embedded in the CPU. This feature provides the end-users with more control over the processor's cache through the use of an FPGA, which can ultimately reduce the latency, increase the throughput, and enhance the determinism of co-designed applications.

By exploiting the new capabilities mentioned above, the aim of the present work is put on providing a solution for low-latency resource allocation acceleration for a class of hybrid architectures, especially cache management schemes, and an enhanced execution determinism. Due to the versatility and the parallelism provided by FPGAs, the present work explores some widely adopted CPU-FPGA architectures. Note that communication interfaces can significantly affect the transfer efficiency of a given architecture. Therefore, we investigate two prevalent CPU-FPGA interfaces from the literature, namely the Peripheral Component Interconnect Express (PCIe) [85] and the Quick Path Interconnect (QPI) [77].

The contributions of the present work include :

— the development of an accelerator-controlled cache management scheme that allows for a more efficient use of CPU cores for computations ;
— an in-depth empirical characterization of the HARPv2 CPU-FPGA platform, which enables low latency resource allocation and application acceleration ;
— the implementation of an accelerator for the hashing of a blockchain application to validate the proposed cache management scheme.

The experience gained in this work suggested guidelines on how to efficiently use socketed FPGA architectures, in order to identify means of manipulating CPU caches in a deterministic manner, with low latency accesses to shared memory in a multi-system architecture.

The rest of the paper is organized as follows : Section 6.3 reviews some communication standards widely adopted in hybrid architectures, as well as the architectures in which they are embedded. It also states the problem of real-time resource allocation on hybrid architectures. Section 6.4 presents the methods used to characterize low latency interfaces. In Section 6.5, we introduce a proposed queue-based cache management scheme. Section 6.6 presents the experimental characterization results of the MCP, along with a case study of a blockchain application conducted to validate the proposed cache management mechanism. Section 8 concludes the paper and elaborates on future work.

6.3 Background

This section presents some popular CPU-FPGA microarchitectures and makes a brief comparison of some most prevalent interfaces associated with memory models (hierarchy). It also presents a review of related work on latency optimization for hardware acceleration of real-time applications.

6.3.1 Microarchitectures

In high performance computing, device locality and fast interconnects between the host and its accelerator are performance critical. To a certain extent, regardless of the interface used in CPU-FPGA platforms, accelerator locality has a significant impact on communication delays, for both computation acceleration and data transmission.

A typical FPGA-accelerated server is composed of a daughter card in a PCIe slot on the motherboard, as shown in Figure 6.1. This configuration induces usually a high latency. In addition, the complexity of the CPU's root complex can add extra overhead in cases where the data coming from the accelerator have to go through bridges, switches or chipset to reach the processing units. The accelerator's bandwidth is usually bounded by the physical implementation of its communication interfaces. Important improvements have been made in recent years by hardware platforms providers, such as Intel FPGA (previously Altera) and Xilinx, to enable the use of FPGA acceleration in HPC for "Big data" type workloads. Microsoft has successfully integrated Virtex 6 FPGAs in its first launch of their Catapult hardware [61], and then outperformed it with its new upgrade using Stratix D5 FPGAs [86]. Bing also used a hardware-accelerated search engine in its data centers for massive data analysis. The data flow is then highly optimized to favour massive processing over latency as explained above.

Socketed FPGAs, as shown in Figure 6.2, aim at offering low latency, high throughput hard-

ware acceleration. As discussed in [37] and [36], socketed FPGAs can offer low latency communication using the Front Side Bus [41], or its new upgrade, the QPI bus. The earlier socketed FPGAs came as custom-made adapter cards fitting a specific socket type (i.e. LGA2011) [87]. In our previous work [87], we implemented a dynamic resource allocator using a QPI-based CPU-FPGA platform equipped with a Virtex 7 socketed FPGA. The introduction of QPI allows for a cache coherent interface with the accelerator. Using the accelerator system combined with an efficient parallel algorithm can achieve a speedup of two orders of magnitude, while reducing the latency overhead. With the same configuration, Intel launched the Heterogeneous Accelerator Research Program (HARP), through which the first QPI-based CPU-FPGA architecture using Stratix FPGAs was introduced [88].

A year later, Intel introduced in the second iteration of the HARP platform a Multi-Chip Package containing a Xeon-CPU and an FPGA [89], as described in Figure 6.3. This new iteration coming out from the HARP offers improvements on the QPI management of the accelerator side and software development kits (SDKs), as well as the addition of two new communication interfaces exploring the PCIe Gen3 x8 standard. For a given implementation on the MCP, the interfaces can be used separately or combined. To the best of our knowledge, no latency bounded application exploiting the HARP MCP platform has been reported, while most publications using the HARP platform are related to machine learning. Unlike the work reported in [90], which provides information on processing performance per watt and/or accuracy for given models on specific configurations, the work reported in this paper focuses on latency metrics and characterization of the pre-market MCP.

Remarkably, most of the aforementioned platforms follow similar design flows for software and hardware co-design. The typical flow is to use APIs provided with a SDK to program the application on the CPU side, the Accelerator Abstraction layer (AALSDK) [91] and the Open Programmable Acceleration Engine (OPAE) [92] for the new Intel MCP. The FPGA part can be designed at various abstraction levels, ranging from low-level Hardware Description Languages (HDLs), such as VHDL and Verilog, to higher level languages, such as OpenCL. The OpenCL design environment varies according to the FPGA manufacturer, e.g., SDAccel for Xilinx [93] and FPGA Runtime Environment for Intel [94].

An important bottleneck in such systems is the interface between the host CPU and the FPGA. In this subsection we focus on the two types of interfaces available on the MCP : PCIe and QPI. The throughput of QPI is proportional to the processor clock frequency. A QPI has unidirectional send and receive link pairs, which can be activated simultaneously or independently. For a 2.4 GHz CPU, a QPI link has a 2 bit/Hz **rate** (double data rate), 20 data bits per **direction**, and **duplex** operation, with 8 **bits**/byte, which yields a theoretical

throughput of 19 GB/s. An interesting feature of QPI is that the link comes with a cache coherency mechanism allowing for a unified memory space between the accelerator and the processor. Such mechanism allows for cache based communication between the host and the FPGA. It typically provides lower latency than other link types, and is equivalent to a 64B cache line access on x86 platforms. Given the information above, we can calculate the theoretical bandwidth of the QPI link as follows :

$$Bandwidth = \frac{Freq \times Rate \times Directions \times Duplex}{Bits} \tag{6.1}$$

The typical implementation of PCIe for FPGA accelerators is PCIe 8x Generation 3, which gives a link running at up to 8 GB/s. A PCIe link typically has a larger overhead than a transfer induced by QPI. As PCIe interfaces are throughput efficient, they are usually recommended for data transfers larger than 4 KB. However, PCIe is not cache coherent, even if it is allowed to access the unified address space of a process in the case of a CPU-FPGA MCP. Moreover PCIe links do not always deliver their theoretical performance. The actual bandwidth depends on the connection with the processor. It was shown in [76] that, in a topology similar to the one presented in Figure 6.1, PCIe under-performed. However, in this work, a study of PCIe transfers in various scenarios allowed us to observe a close to the maximum theoretical performance. The main differences between PCIe and QPI are summarized in Table 6.1.

Cache coherent interfaces enabled by **unified memory spaces** in, e.g., QPI and CAPI interfaces provided by IBM [95] allow for a better memory management in accelerator architectures. Whereas non-coherent memory spaces imply that the host and/or the accelerator need to explicitly copy and allocate the data. In the case of message transmission acceleration towards the host with a topology similar to Figure 6.2 or Figure 6.3, data copy and allocation are needed only on the host side. Simpler data structures will result in lower data allocation latency (new/alloc). For the case of **private memory spaces**, typically with a topology similar to Figure 6.1, data allocation needs to be performed (copied) on both the host and accelerator main memories.

The main advantage of coherent unified memory spaces is that it allows for a device to directly access and consume the data without reallocation and duplication, as both the accelerator and the host have a direct access to the memory space. The MCP is one of the first platforms to support a unified memory-mapped I/O (MMIO) space between the host and the accelerator for x86 high performance server-based processors, allowing for zero-copy transactions. In this paper we use the combination of the cache coherent MCP and low latency interfaces for real-time hardware acceleration of resource allocation.

Tableau 6.1 Communication Interfaces

	QPI	PCIe Gen3 x16
Topology	Link	Link
Bus Width (bits)	20	16
Bi-directional Bus	Yes	No
Rate (GTransfers/s)	12.6	16
Theoretical Bandwidth (GB/s)	25.6	16
Memory	Coherent/MESIF	No

6.3.2 Real-Time Resource Allocation

This subsection reviews some notable results on accelerators for realtime resource allocation, among which latency reduction is one of the most attractive subjects. A typical way to get a reduced latency is to use a Real Time Operating system (RTOS), such as FreeRTOS [96]. It is explained in [44] how to reduce the latency inferred by resource management with smart assembly insertions at the kernel level. However, a problem associated with RTOSs is their rigidity and their lack of support for new computation libraries. A solution consists in the use of a new method and architecture. Indeed, there exist many solutions for hardware-base implementation of scheduling algorithms, such as [97], [98], [99], and [100]. In a previous work [7], we demonstrated that using StarPU with an adequate scheduling algorithm enabled the virtualization of a latency sensitive application (LTE) on a computing platform. It is demonstrated in [42] that gains can be obtained in terms of latency reduction for a RTOS supported by hardware implemented scheduling algorithms. Compared to the work reported in [42], where a RTOS on Xilinx Zynq is used, we plan in the present work to leverage hardware acceleration on a much larger scale based on a MCP FPGA on an HPC compliant OS.

6.4 Characterization of Low Latency Interfaces on a Hybrid CPU-FPGA Architecture

In this section, we provide a detailed description on the method used to characterize low latency interfaces, including the specifications of the platform used, the test environment, and the means used to control the state of the memory hierarchy during the experiments.

6.4.1 Hardware Configuration

The detailed configuration of the hybrid architecture is built on a multichip package (MCP), as presented in Section 6.3.1. This MCP contains a Broadwell 14 core 2.4 GHz Xeon processor

integrated with an Arria 10 GX 1150 FPGA, both are server grade processing units. The FPGA has one hardware single precision floating point unit embedded in each DSP block (1518 blocks) and 54000 M20k memory blocks for high performance applications. Such a large FPGA coupled with a powerful general purpose processor allows performing large scale computations, and the cache coherent interface enables the implementation of fine grain accelerators.

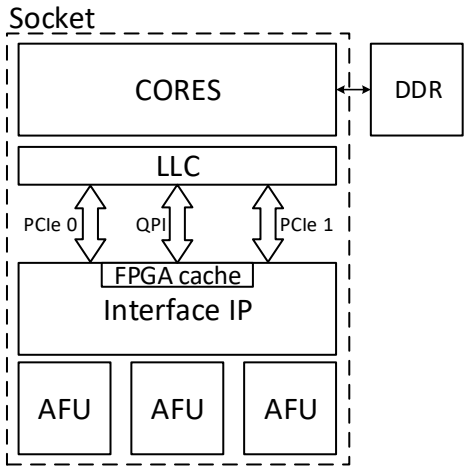


Figure 6.5 QPI and PCIe connections between CPUs and FPGA. Last Level of Cache (LLC), Double Data Rate (DDR), Accelerated Function Unit (AFU).

Intel's HARPv2 connects the host to the accelerator (FPGA) with three links : two PCIe interfaces and one QPI as shown in Figure 6.5. This provides flexibility in data transfer adapted workloads. Intel's Core Cache Interface (CCI-P) [101] allows for leveraging all the links simultaneously or profiting from the use of a specific interface. Normally, the QPI link can achieve a maximum bandwidth of 12.8 GB/s. The platform also offers two PCIe 8x Gen 3 links, each with a theoretical maximum throughput of 8 GB/s giving a combined PCIe bandwidth of 16 GB/s. Combining all three links gives the system a theoretical maximum throughput of 28.8 GB/s.

6.4.2 Characterization Approach of Low Latency Interfaces

This subsection details the method used for MCP characterization, while the related experimental results will be reported in Section V. It should be noted that the results are obtained on a pre-production platform and may not reflect the best performance that a production-ready MCP can offer. Efficiency and optimization analysis of the interfaces should be carried out once more when the production platform is available. This is justified as the pre-production devices might have some unresolved issues to be corrected in the final product.

Latency and bandwidth measurements of the MCP reported later were performed using a loopback application. We first explain the test environment and our workflow, followed by a detailed report of the method used for characterization.

We used the AALSDK 5.0.3 for the host program coded in C, and the Accelerated Function Unit (AFU) in the FPGA is coded in Verilog. Due to the scarce availability of the MCP platform, the current workflow imposes to work remotely on HPC servers [102]. For efficiency purpose, we first simulated the entire environment using the Intel's Accelerator Functional Unit Simulation Environment (ASE) [103]. The ASE is linked to Questasim 10.6a [104] (or Synopsys) for code compilation and simulations, as well as to Quartus 17.0 [105] for the synthesis and place-and-route features. Once a program is validated in the simulation environment, the entire program and the bit file are transferred to the HARP servers for testing. At runtime, the overall FPGA clock is bounded at 400 MHz by the CCI-P maximum frequency, giving a 2.5 ns period.

As the QPI link is cache coherent, the cache initial state needs to be defined. To have reproducible results, this experimentation starts at a state where the cache is not full nor configured to maximize collisions. To optimize the experiments, the current implementation of the loopback enables the system to have a warmed up cache in order to facilitate cache hits [106].

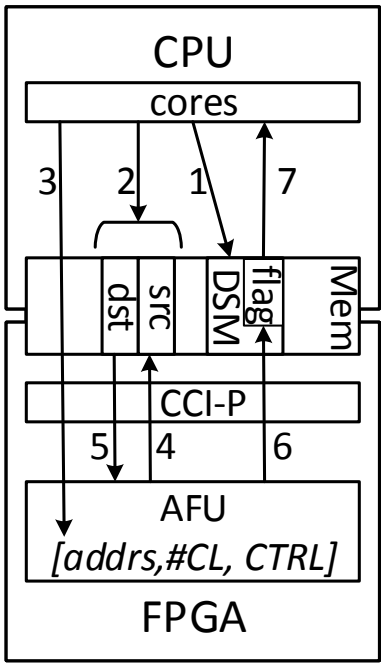


Figure 6.6 The loop back application

Figure 6.6 illustrates an in-depth representation of the proposed loopback test. At high level, a loopback test consists simply in sending a message and waiting until the message is sent back to the transmitter. The CPU starts the test, then the FPGA initiates a write request and starts the time-stamping. As soon as the CPU receives the data, they are written back to another location in the shared (cached) memory. When the FPGA receives an acknowledge that the data were written, time-stamping is stopped. At the end, the FPGA verifies the data transferred.

Note that on the software side, Intel's AALSDK provides an API that can be used to setup a service between the CPU and the FPGA, as well as MMIO bounded functions to use and align virtual and physical addresses. Table 6.2 shows some useful functions used in the considered loopback test.

The hardware implementation of the loopback application is straightforward. The AFU contains three main blocks : a **read** Finite State Machine (FSM), a **write** FSM, and a free running counter based on which important events will be time stamped. The remaining Verilog code assists and enables the AFU and CCI-P communication. Each FSM can receive and send requests to and from the CCI-P.

Typically, for the case of an FPGA write loopback initiated test, the free running counter starts when the processor start flag is received by the AFU and stops when the AFU reads its last cache line. As an example, the operation initiated by the AFU would go as follows :

— Starts a free running counter at an AFU start ;
— Timestamps an event at a write request to the CCI-P ;
— Timestamps an event at a response of the request from the CCI-P ;
— Stops the free running counter at the last read response.

At the end of each test, the timestamps are sent back to the CPU for evaluation.

Figure 6.6 shows the detailed interactions of the processor and the FPGA for platform characterization. Each arrow is associated with a number, which corresponds to the action described by the bullet of the same number listed below. The list corresponds to a typical FPGA initiated loopback test, where the FPGA writes the source buffer in cache, and the CPU copies

Tableau 6.2 Useful AALSDK API Functions

Function Name	Utility
ALIBuffer	Align Virtual to Physical addresses
ALIMMIO	MMIO Access
ALIUMSG	Low Latency message
ALIReset	Reset AFU

that source buffer to the destination buffer. The FPGA then validates the data to confirm the test validity.

1. The processor sets up the Domain Specific Memories (DSMs). The DSMs are a combination of host specific memory spaces as well as cache aligned spaces intended to be used for commands to be passed to the AFU, also called Control Specific Registers (CSRs).
2. The host creates and initializes the source and destination memory spaces. Each space is set to a specific size of 1024 cache lines, which is 64KB. It also corresponds to the size of the FPGA cache on the AFU side.
3. The CPU sets the MMIO CSRs. It creates and asserts a start flag for the test in the CSR. It dictates the AFU to start.
4. The CPU listens until the done flag is set.
5. The AFU writes in the source buffer and then reads the content copied by the CPU in the destination buffer.
6. The done flag is set.
7. The CPU acknowledges the termination of the test, receives the events timestamps, and the AFU verifies that the data in the SRC and DST buffers are the same.

The acknowledgement received in Step 7 concludes a typical execution of the loopback application. The above test is initiated from the FPGA, however the loopback test can also be initiated from the CPU side, which should yield similar results. The low latency interfaces enable us to have a cache management mechanism implemented on an FPGA. The following section provides the details of the implementation of the proposed cache management. The characterization results are presented in Section 6.6.

6.5 Performance Improvements with Queue-Based Cache Management on an MCP

In this section, we propose a novel approach using an FPGA to enable efficient cache management and reduce the waiting time caused by data fetching. In general, cache management is performed by dedicated CPU hardware, such as the Cache Allocation Technology (CAT) [107] for Intel processors, which aims at enhancing the system. However, it is not trivial to implement cache management as well as task scheduling on the CPU side in an efficient manner. Benefiting from the hybrid CPU-FPGA platform shown in Figure 6.7 and its low latency transmission delay through the QPI interface, we propose a queue-based cache management

technique inside the FPGA, which can be used in real-time applications, such as task allocation and scheduling. Indeed, cache management with deterministic performance can be implemented with the help of modules in the FPGA. The immediate benefits of this hybrid architecture are summarized as follows :

— Deterministic execution and performance guarantee of a cache manager provided by the FPGA for real-time applications ;

— Low latency QPI and high throughput PCIe buses for communication between a CPU and the FPGA ;

— Reconfigurable implementation of cache management and upgraded capacity for function acceleration with the FPGA over the CPU alone.

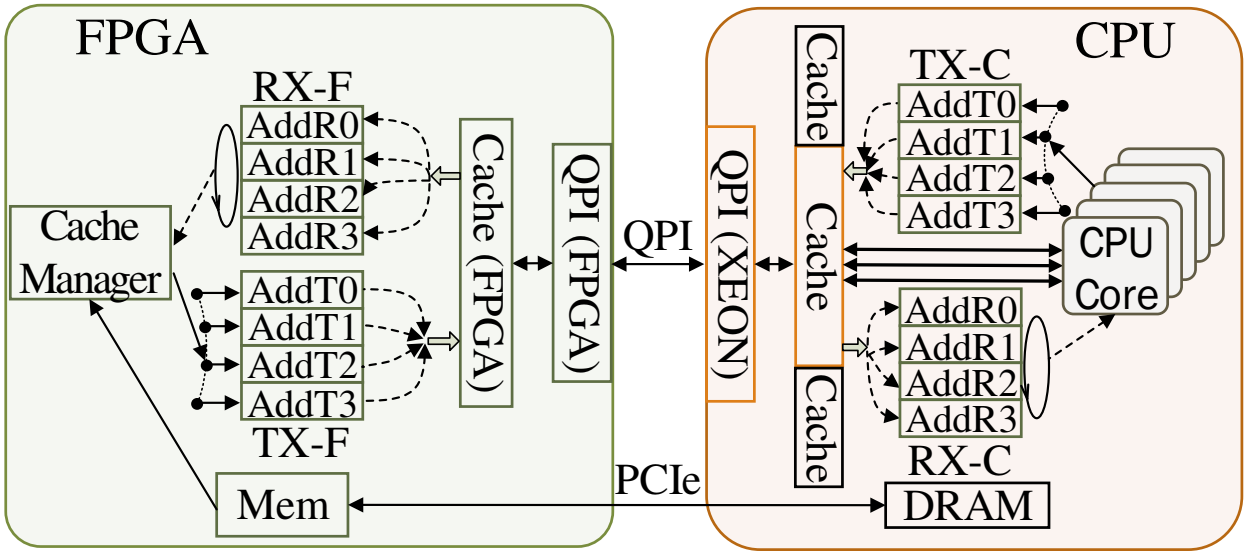


Figure 6.7 Proposed hybrid architecture composed of CPUs and an FPGA.

As the main processor in the MCP is cache coherent with its accelerator, FPGA cache can be regarded as a copy of a designated part of the CPU's Last Level of Cache (LLC). Therefore, any change in either cache will be replicated to the other side via the low latency QPI bus. This mechanism enables the management of the shared cache with an FPGA. It is known that an FPGA offers parallel computation capability. Thus, the FPGA can provide deterministic performance cache management without interference from an operating system (OS). Furthermore, in practice, it is possible to boost execution performance and hide transmission latency by pre-fetching data from main memory to cache before task execution. However, it may be impossible to pre-fetch large amounts of data for a given execution, due to cache current state and overall size. Thus, efficient management schemes are required.

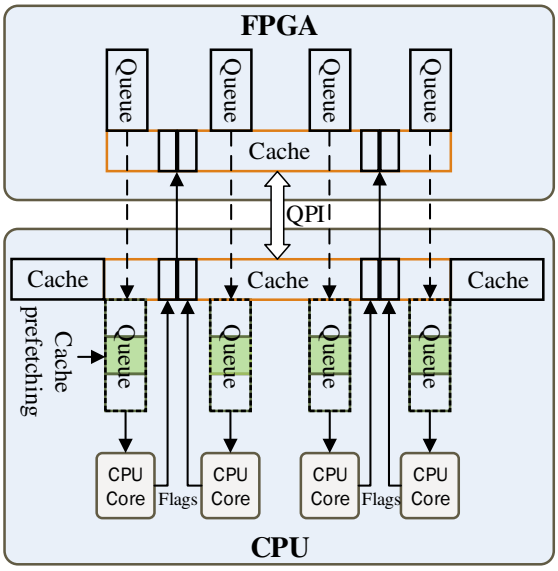


Figure 6.8 Queue-based cache management.

In its basic implementation, the management scheme is represented as content queues written by the FPGA to be replicated or read from the CPU through the cache. This mechanism can be explained with a task scheduler using our cache management scheme. Therefore, for a given Directed Acyclic Graph, a queue-based cache manager is proposed and implemented in an FPGA.

As shown in Figure 6.8, the shared cache is divided in two main memory blocks. The first block is used to receive the control signals for and to the FPGA, such as done, ready, send and acknowledge flags, represented as the sub-blocks arrow in the figure. The other block is used to write the fetched data, in order to have it ready for task execution by a CPU core. In a typical flow, the FPGA containing ready task queues for each CPU execution units fetches the necessary data ahead of task execution if it is ready into the corresponding CPU queues. The fetched data are therefore ready for execution. In order for the FPGA to keep track of tasks execution and to maintain its ready task queues, the CPU sends control information to the FPGA such as task ID, ready and done flags.

The handshake between the processor and the accelerator ensures an efficient resource management. Using the handshake and the FPGA in a well configured OS environment enables a near deterministic hardware accelerated cache management, improving the underlying system's performance. Multiple steps, such as adding kernel patches [108], running on a bare metal OS [109], disabling hyperthreading [110], as well as using core isolation techniques,

can be taken in order to improve the determinism. Even though taking all these steps should produce a much more deterministic cache management, it remains to be confirmed experimentally whether other sources of performance variability would remain.

Note that in order for the previously exposed scheme to be efficient, it is assumed that there are enough tasks ready for execution. Besides, task execution time needs to be longer or comparable to the time required for prefetching data. Thus, normally, benefiting from queue-based cache management and cache prefetching, a continuous task processing can be realized without any waiting delay due to input data loading from main memory to cache.

To validate the expected benefits from the queue-based cache management and cache prefetching scheme introduced above, we present a "case study" in the following section. It serves as a proof of concept on which future work may expand.

6.6 Validation of Low Latency Communication and Case Study of Cache Management

In this section, the low latency interfaces are characterized and utilization recommendations of the hybrid platform are provided. Then, we implement the proposed cache management scheme in an FPGA embedded in the MCP architecture to validate the improvements it can bring to system performance. Note that due to the scarcity or non-availability of socketed and MCP type CPU-FPGA architectures from other vendors, such as Xilinx, this paper focuses on the comparison of delay and bandwidth for all three architectures mentioned in Section 6.3, based on the Intel platforms.

6.6.1 Characterization Results of Low Latency Interfaces

This subsection presents and dives in depth in all the results obtained on the MCP platform. It also provides some recommendations on how the platform may be used to generate low overhead applications. The MCP CPU-FPGA QPI communication has typical cache memory access patterns, where a cache transaction is described as either a "cache hit" or a "cache

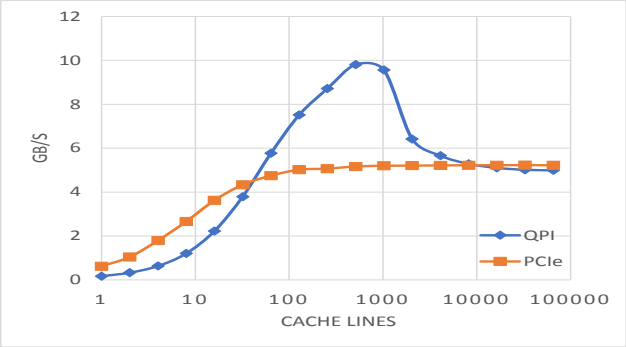
Tableau 6.3 MCP access latency

Access	Latency (FPGA cycles)
Read Hit	25
Write Hit	28
Read Miss	119
Write Miss	129

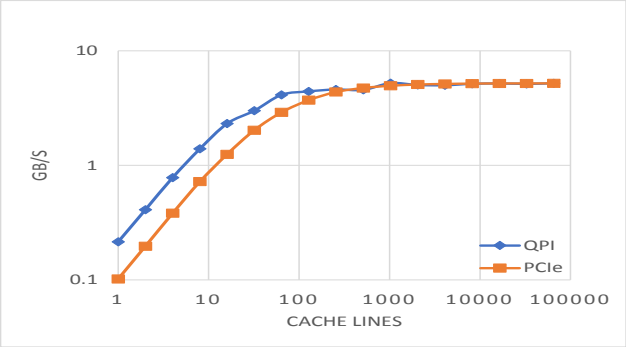
miss". As shown in Table 6.3, the QPI's cache hit latencies are similar to those exposed in [76]. However the average read and write miss latencies are reduced by around 65%. We also observed that unlike [76], the empirical bandwidths are close to their theoretical values for QPI, about 12.6 GB/s as seen in Figure 6.10b, but still lacking for PCIe with about 5.2 GB/s for PCIe, as shown in Figure 6.9c.

To better understand the intricacies of the different links, we first compare their performances in the cold cache environment, then with a warm cache to conclude with a discussion on which link is adequate for what situation. Figure 6.9a shows the write throughput of PCIe alongside QPI's, for transfers from 1 to 65535 cache lines in a cold cache environment. We observe that PCIe has a higher write throughput than QPI for small transfers, i.e., 1 to 32 cache lines. However for transfers over 32 cache lines up to the cache capacity, QPI's throughput is better than that of PCIe, exhibiting a peak at around 10 GB/s, almost twice PCIe's throughput and dropping down towards the level of PCIe after cache capacity is reached. For all transfers over the 64 KB cache capacity, throughput for QPI and PCIe are similar around 8192 cache lines, while QPI lacks behind for larger transfers, due to the cache coherency protocol (evictions and cache misses). For the case of read throughput, Figure 6.9b shows that QPI has a slight advantage over PCIe for read throughput up to 128 cache lines, at which point PCIe and QPI have a similar read throughput of about 5.2 GB/s, equivalent to PCIe's measured limit. Figure 6.9c and Figure 6.9d, show the detailed evaluations of PCIe and QPI for their read and write throughput in the aforementioned cold cache environment. As for the links latency, we observe in Figure 6.9e that QPI has a smaller latency of about 100 cycles for transfers from 1 to 32 caches lines reaching PCIe's latency around 128 cache lines. However the inverse happens for PCIe, where its write latency in this cache environment is smaller than QPI's for the same small transfer interval, as shown by comparing Figure 6.9f and Figure 6.9g. Figure 6.9h compares PCIe and QPI loopback latencies. It shows that they are similar, meaning that the PCIe write affinities and QPI's read affinities cancel out in a cold cache environment. It indicates that for all applications requiring quick loopback performance in a cold cache environment, either links are suitable.

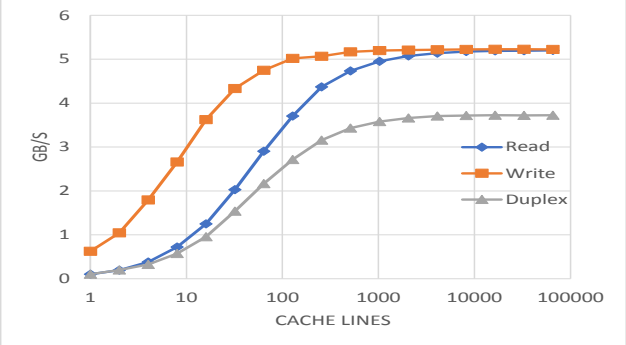
As for results in the warmed cache environment, we observe that QPI is greatly affected by the cache state. It can be observed that its results in a cold cache environment are both worst in read and write latencies over the warmed cache results. Figure 6.10a shows that we can achieve the lowest read and write latencies for QPI, i.e., 25 and 28 cycles respectively. The warmed environment allows for QPI to reach its announced maximum throughput of 12.6 GB/s for reads and writes, as shown in Figure 6.10b. Due to the multiple links available on the MCP, using an aggregate of the two PCIe and the QPI link is possible. Figure 6.10c and Figure 6.10d, show the overall latency of read and write transfers when the links are used



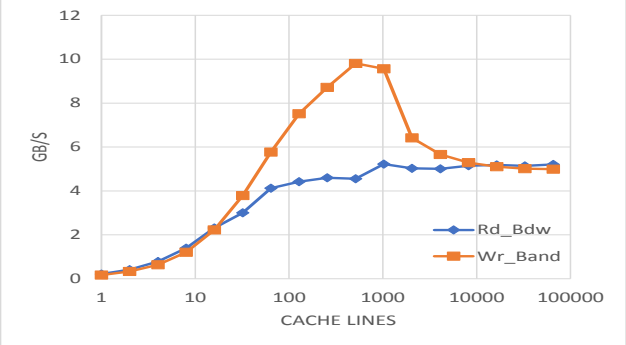
(a) QPI vs PCIe write throughput



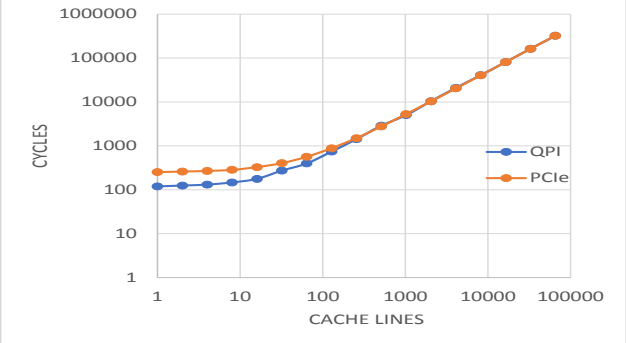
(b) QPI vs PCIe read throughput



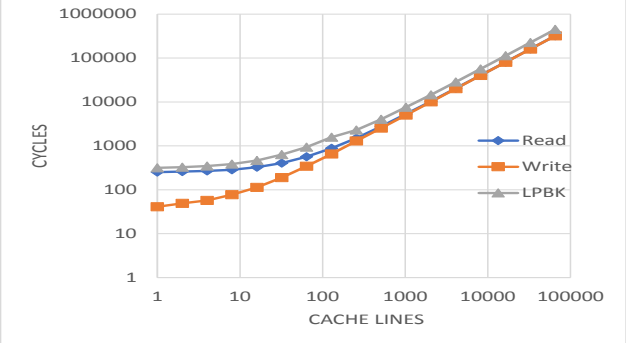
(c) PCIe throughput results



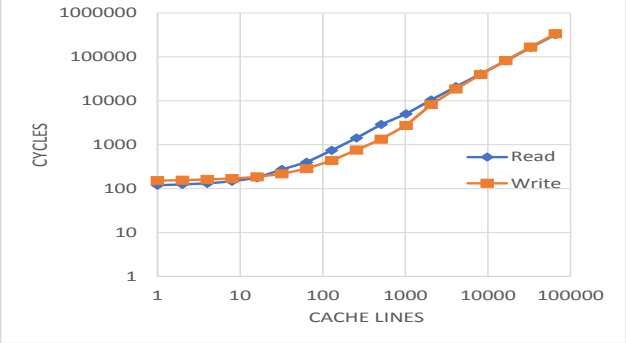
(d) QPI throughput results



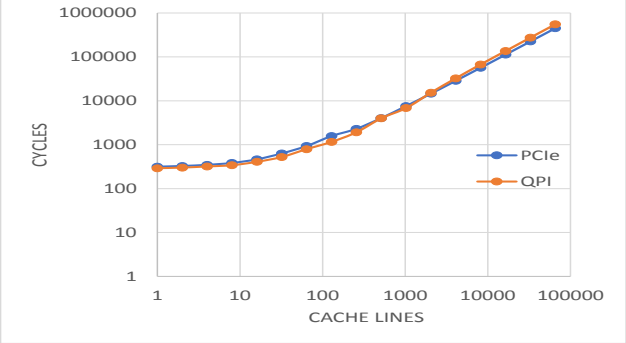
(e) QPI vs PCIe read latency



(f) PCIe read and write latency

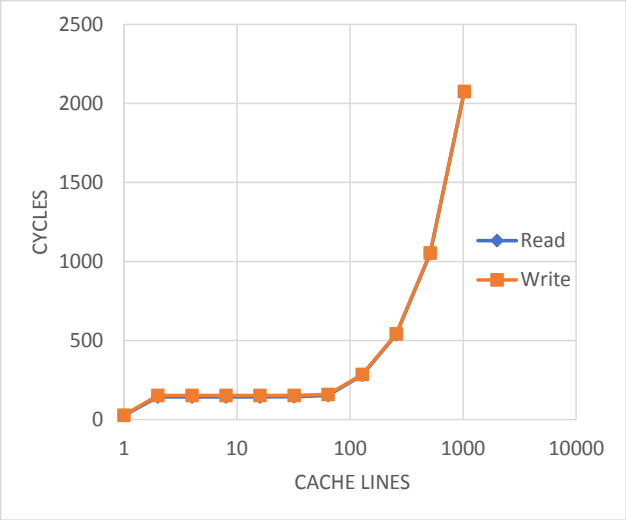


(g) QPI Read and Writes latency

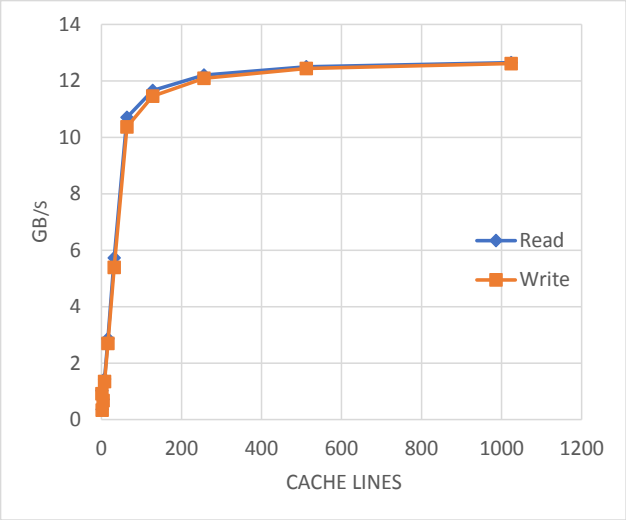


(h) QPI vs PCIe loopback latency

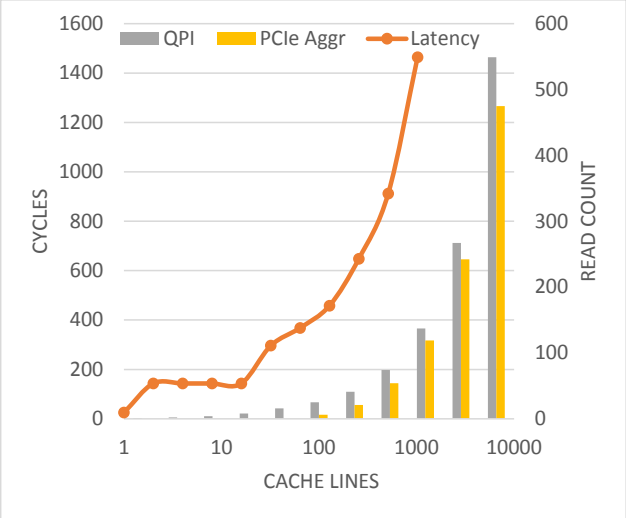
Figure 6.9 Cold cache characterization results



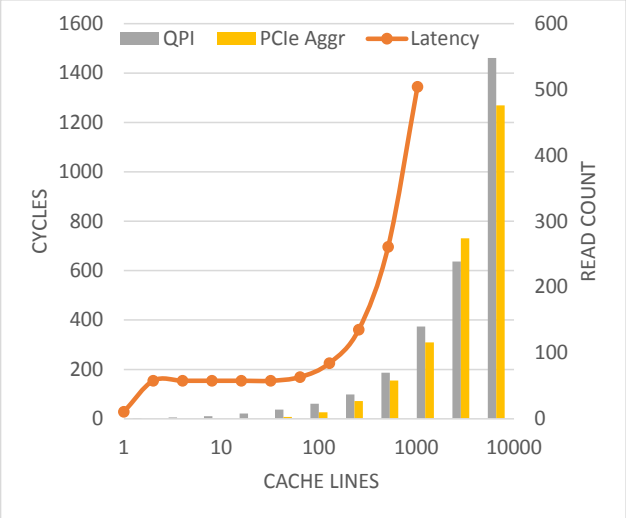
(a) QPI read and write latency



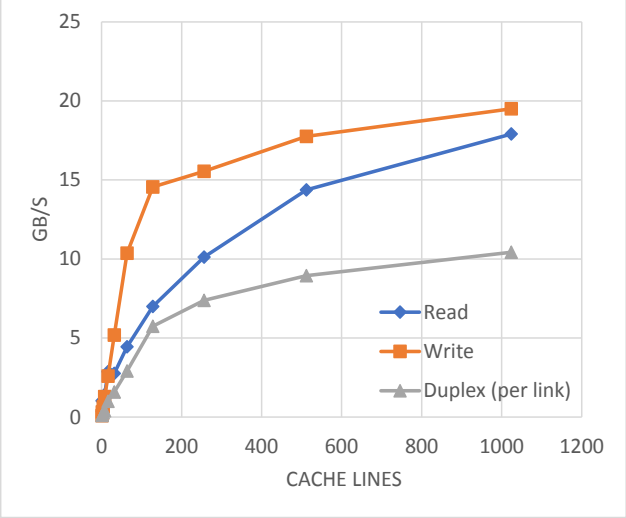
(b) QPI throughput results



(c) Aggregate Link Read latency



(d) Aggregate Link Write Latency



(e) Aggregate Link Throughput

Figure 6.10 Warmed cache characterization results

simultaneously. The histograms on each figure display the number of reads and writes done by each type of links (QPI and 2x PCIe). For instance, it can be observed that, in Figure 6.10b, the aggregate read latency is equivalent to that of QPI in a warmed cache ; that is due to the aggregated protocol using only QPI, as its histogram display 1 count for QPI and 0 for the combination of both PCIe links. For a bigger transfer of 256 cache lines, we observe that the read latency of the aggregate is better than PCIe alone but worse than QPI alone for the same transfer size, which can also be explained by the transfer count of the aggregate favouring QPI over PCIe, with 137 QPI reads and 119 PCIe reads. Lastly, Figure 6.10e shows that the highest throughput can only be achieved using aggregated transfers, with an average of 19 GB/s and a max of 20.634 GB/s for a 1024 cache lines writes.

To summarize, in the cold cache environment, the QPI link does not have a clear advantage over PCIe. PCIe has a small latency over QPI for writes when transfering less than 32 cache lines, which is also evident for write throughput. On the other hand, QPI displays better read latencies and throughput across the board for all transfers under the FPGA cache size. Once the cache size limit is reached, PCIe and QPI have similar performances. Lastly, in the warmed cache environment, QPI read and writes are better than PCIe in all instances where the data fit in cache, where the read latency reaches as low as 25 FPGA cycles, as well as reaching the link's theoretical max bandwidth of 12.6 GB/s. Note that as PCIe is a non-cache coherent link, its performance is not affected by the cache state. The results show that as long as the transfers fit in the available cache, QPI is the preferred interface to use for low latency communication with the processor, while it cannot deliver a high throughput for under 32 cache line writes. In most of the results, PCIe comes second to QPI in performance, except in write performance for small data transfers due to the PCIe write affinity to main memory. Note that as QPI is a cache coherent link, data transfers are cached, allowing for better performing applications based on data locality alone. A good compromise for latency and bandwidth is the use of aggregated transfers on the three available links. Note that in both read and write latencies, the QPI link is favoured over the PCIe links. This ensures the lowest possible latencies for reads and writes, while getting the best overall throughput. Spreading the transfer load on multiple interfaces could have a benefit on memory contention with long running applications. However, more experiments are necessary to verify this hypothesis. We also noted that in most of our warmed-up cache experiments, we would get a cache miss on the second cache-line transfer. More investigations are needed to find the source of the problem, although it does not invalidate the insights and the results provided in this paper.

In light of the results in Figure 6.9 and Figure 6.10, conclusions on how to better use the MCP can be drawn. Most of the insights provided in [76] on offload to an accelerator with an architecture similar to Figure 6.1 hold true, especially those related to data movements.

However, some assumptions and recommendations targeting the PCIe in particular are invalidated by the use of the unified memory space of the MCP. A decision tree to choose interfaces in different circumstances is provided in [76]. The following tips and suggestions should be taken as recommendations for MCP users and designers.

Insight 1 : For the lowest latency transfers, QPI should be prioritized. We would recommend that as long as the amount of data to be transferred is less than the remaining available cache size, the QPI interface should be used over the PCIe. For a cacheline, the typical latency of a QPI transfer is 64 ns (25 cycles) versus 625 ns (250 cycles) of a PCIe transfer. A QPI cache miss penalty of a cacheline is still more latency efficient than a PCIe transfer.

Insight 2 : QPI offers a higher bandwidth than one PCIe x8 Gen3 for payloads up to 64 KB (for an empty cache). In order to have a higher bandwidth irrespective of latency, both high latency links should be used, offering up to 16 GB/s throughput.

Insight 3 : If latency variability is not an issue, using the Intel automated aggregate interface outperforms the PCIe interface alone for most cases. Using both interfaces provides the "best of both worlds" as exposed by Figure 6.10. Whenever the payload is small enough, QPI will be the prioritized interface, while PCIe will be for larger payloads and both for the in-between. Compared to PCIe only, even with the latency added by memcopy, the aggregate transfers are more efficient. The same applies for high throughput needs, and Figure 6.10 shows a throughput up to 20 GB/s using the automatic aggregate interfaces, out of a theoretical 28 GB/s. However using all the links at once, a well crafted manual implementation might allow for a higher bandwidth.

Insight 4 : The MCP accelerator may not be the solution for all use cases of FPGA acceleration. For example, if the data from the functions to be accelerated are to be transferred to another machine, the multi-interface transfer latencies might invalidate the use of the integrated accelerator. That would be the case if the latency to perform an additional back and forth transfer between the CPU and FPGA adds more time to the execution than the processing time saved. In this case a NIC equipped daughter card FPGA might be preferable.

Insight 5 : On the accelerator side, using the FPGA cache might still be ill-advised due to its long access latency (64 ns for reads and 70 ns for writes) compared to a BRAM or register access (typically 1 to 2 cycles). The cache is better used as an interface between the CPU-FPGA than as a storage medium for FPGA's AFUs.

Insight 6 : The MCP accelerator is better used in the unified memory space of the system. Whenever offsite memory accesses are needed, a DMA enabled daughter accelerator card might be preferred.

6.6.2 A Case Study - Merkle Tree Acceleration

Since the inception of Bitcoin by a still unknown creator going under the pseudonym of "Satoshi Nakamoto" [111] in January 2009, hash-based cryptography for decentralized applications gained a widespread popularity. The growing cryptocurrencies realm counts over 1624 different "active" tokens over multiple exchanges and platforms. Most of these tokens rely on a hash-based tree structure for their digital signature schemes. This tree based generalization of a hash list is called a "Merkle tree" [112]. Each leaf node is a hash of a block of data and a non-leaf node is the hash of its children. Typically, a Merkle tree has a branching factor of two.

An integral part of decentralized trust-less transaction systems, such as the first pseudo-anonymous Bitcoin [113], the enterprise aware Ethereum peer-to-peer networks [114] or the privacy token Monero [115], is called a "Block". The block represents the main scalability feature of cryptocurrencies, where it is part of a multi-level data structure within the "Blockchain". The block records all the transaction data for the case of Bitcoin. Therefore the sequence of multiple blocks over time represents the blockchain. Figure 6.11 shows the relationship between the block and the Merkle tree, and also displays a typical hash-based tree structure.

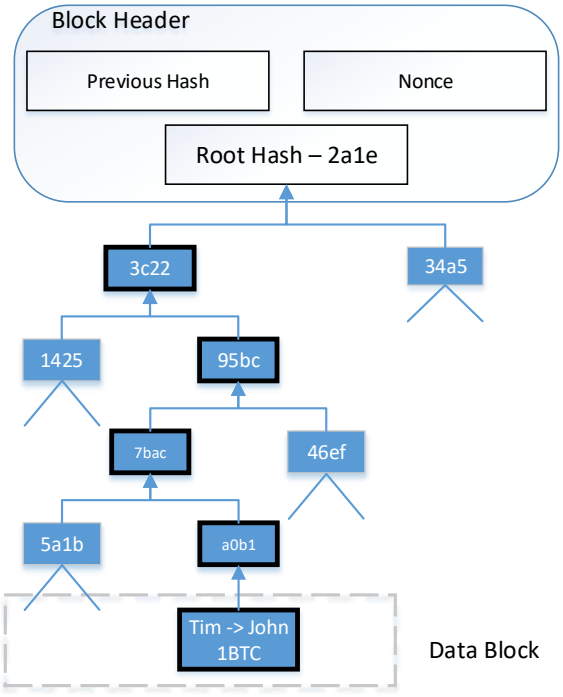


Figure 6.11 Merkle Tree in Blockchain applications.

The case study considered in the present work includes a simple implementation of a blockchain with a SHA-256 hash function. Within the Bitcoin network, the SHA-256 hash function serves for two mains purposes [111] :

— Mining, where a "miner" creates a new block to the block chain with a process called "proof-of-work" to ensure that all the blocks added to the blockchain are legitimate and free of fraudulent information. The proof-of-work mechanism solves a SHA-256 hash function.
— Creation of new addresses, where every new address needs a public key to be hashed. SHA-256 is the hash function used for the creation of new addresses because it allows for greater security as well as shorter addresses.

The advantage of creating Merkle trees within blocks in the case study is that the size of a SHA-256 hash function is 256 bits, i.e., 32 bytes [116], which is coincidentally half a cacheline long. It allows for our implementation of hardware prefetching to combine data, addresses, and any other possibly relevant data in one cacheline transfer between the CPU and the AFU. Another advantage is that the computation of a more complex hash function on the CPU side allows for longer computation, and hence, it hides some of the communication time implied by the accelerator prefetching on the MCP.

This makes use of a simple Merkle tree with a SHA-256 cryptographic hash function reminiscent of cryptocurrencies, such as Bitcoin, representing an adequate application to test our cache-based management sheme for hash computation acceleration using an accelerator prefetching. The following subsection dives into the details of the method and implementation, as well as giving the results of this proof of concept.

6.6.3 Implementation of Merkle Tree and Experiment Results

In order to show that accelerator prefetching could be beneficial, we developed a simple Bitcoin-esc blockhain chain with an oversized Merkle tree. For each block, we created a 30 MB Merkle tree to make sure that during the tests the CPU would not fetch the entire tree when needing to compute the hash of a given node. For each test, a block is created and it would iterate multiple times to compute the hash of random nodes until 64 000 nodes are computed. The use of the random function allows for low to no compiler optimization. Listing 6.1 shows the order in which the application runs.

Listing 6.1 Merkle Tree Implementation Pseudo Code

```
  merkt_t *merkt = merkt_create();

  prevAddr = startAddr;

for (uint32_t i = 0; i < 64000; ++i) {
  nextAddr = rand_node(merkt, seed,addrSpan);
  send_Addr(getNextAddr,m_inputVirt);
  compute_hash(prevAddr);
  prevAddr = nextAddr;
}

merkt_delete(merkt);

print(‘‘End Test’’)
```

Each time we access a specific node of the Merkle tree to compute its hash, we use a pseudo random function that chooses the next branch to compute. The predicted address is relayed to the FPGA in order to prefetch the data associated with the address, while the processors compute the previous node's hash. In order to transfer the address from the CPU to the FPGA, a simple service is implemented. A virtual address is linked to a cache-aligned physical address. Listing 6.2 shows the use of pointers to the virtual address to copy the next node to compute via the communication service implemented with the AALSDK.

Listing 6.2 Address transfer through a service

```
  //Save size for input virtual address
  m_pALIBufferService->
  bufferAllocate(ALLOCATION_SIZE, &m_InputVirt)

  m_InputSize = BUFFER_SIZE;

  m_InputPhys = m_pALIBufferService->
  bufferGetIOVA(m_InputVirt);

  // Set the input address for communication
  m_pALIMMIOService->
  mmioWrite64(SRC_ADDR, CL_ALIGN_ADDR(m_InputPhys));
```

On the FPGA side a simple memcopy type application is implemented. It receives the address and reads the data associated with the address in order to keep the data in cache. The

hardware interface is coded. The AFU "read" FSM is shown in Listing 6.3.

Listing 6.3 Simple application

```
begin
    case(read_fsm)
    2'h0:
    begin // Waits for go
     RdAddr <= 0;
     RdLen <= CL_len;
     RdSop <= 1'b1;
     Num_Read <= 20'h0 + CL_len + 1'b1;

     if(go)
       if(NumLines!=0)
        read_fsm <= 2'h1;
       else
        read_fsm <= 2'h2;
   end

   2'h1:
   begin // Send read requests
    if(RdSent)
    begin
      RdAddr <= RdAddr + CL_len + 1'b1;
      RdLen <= RdLen;
      RdSop <= 1'b1;
      Num_Read <= Num_Read + CL_len + 1'b1;

       if(Num_Read_req == NumLines)
       if(Cont) read_fsm <= 2'h0;
       else read_fsm <= 2'h2;
     end
   end

   default: read_fsm <= read_fsm;
   endcase

end
```

Essentially, once the AFU reads the next address to hash, the data becomes available in the cache for the processor resulting in fewer accesses to the main memory for computation, which should allow the application to run faster.

One thing to note is that the current implementation of the MCP provides a MMIO space accessible by the FPGA, but it has a limited size of only 4MB. This means that the current case study only runs with addresses in that 4MB span. Given that span, the theoretical maximum number of 64B accesses is 64,000. Given the information above, we can calculate the theoretical improvement limit of accelerator prefetching as the maximum latency (data is not in cache) divided by the minimum latency (data is in cache) :

$$Acceleration = \frac{T_{Hash} + T_{MainAccess}}{T_{Hash} + T_{L3Acces}}, \tag{6.2}$$

where "$T_{MainAccess}$" is the typical main memory access time, "$T_{L3Acces}$" the typical L3 cache read access time from the CPU, and "T_{Hash}" the typical time to compute one SHA-256 hash on one CPU core. This equation is true only if the hash computation takes longer than the time it takes for the AFU to receive the next node's address and read. It is important that AFU processing time is masked by the hash computation, otherwise it would add time to the overall execution, thus invalidating the acceleration. For the equation to be effective, it also needs a typical main memory access longer than a L3 cache access. The Xeon E5 provides a typical per core hash rate of 2.06 Mh/s. This implies that it would take about 490 ns to compute one SHA-256 hash, which is 4.9x longer than the 100 ns of worst case main memory access on a CPU. The values in Table 6.3 and Table 6.4 show that a typical send address and read address from the CPU to the AFU would be around 72.5 ns in the best case and in the event of an AFU cache miss and the longest L3 access, 317 ns would be the worst case. The formula is therefore true for our test, and the theoretical acceleration limit for the considered application is 1.157 for the worst scenario, where the respective L3 and main memory accesses would take the most time.

A remarkable outcome is a faster access time implied by the efficient prefetching. As shown in Table 6.4 the AFU-based prefetching reduces a main memory access to a LLC access, giving

Tableau 6.4 Typical Memory Hierarchy Access times

Memory Level	Access Time (ns)
L1 cache	1-5
L2 cache	5-10
L3 cache	10-20
Main Memory	50-100

a 5-fold latency improvement in the worst case scenario. This implies that for a more memory bounded application, the overall application speed up would be much more significant. For an application in which time would be consumed 90% by memory transfers, Amdahl's law expresses :

$$S_{max} = \frac{1}{(1-p) + (\frac{p}{s})}, \tag{6.3}$$

where s is the performance improvement factor and p the part that can be improved. The maximum theoretical speedup of the application would be of a factor of 3.6, where the improvable portion of the application is 90% and the improvement is 5-fold.

After running the Merkle Tree test for 1 million different blocks, we get an average acceleration of 1.118 with a min of 1.10 and a maximum of 1.133, which is close to the theoretical limit of 1.157 computed by (6.3). After some investigation, we may improve the acceleration using a low latency messaging API provided by the AALSDK. Another possible improvement would be to limit the Merkle tree and the random node accesses to a specific 4MB space. It would allow the AFU to "prefetch" all the data to be processed. However such an implementation could induce more evictions and cache misses in the FPGA cache, which could result in a lower performance, thus negating the need for accelerator prefetching.

The case study described above serves as a proof of concept for accelerator-based cache management. The results show that the prefetching can be done using an FPGA embedded in the MCP. Other cache management techniques would be feasible using the MCP, which will be addressed in our future work.

6.7 Conclusion and Future Work

In this paper, we provided an in depth characterization of the new CPU-FPGA MCP. We introduced a new cache management technique, where the accelerator shares the control and is able to prefetch data to make it readily available to the CPU cores. The study of the MCP exposed a new CPU-FPGA platform which enables low latency and high bandwidth resource management, while being able to be used to accelerate latency bounded applications. The MCP's characterization shows that it can achieve access latencies as low as 64 ns and throughput as high as 20.6 GB/s. A fine-grained acceleration became possible due to the coherent FPGA cache with the LLC through the QPI interface, as well as being able to deal with larger datasets using the CPU's main memory's address space through the PCIe. We made a proof of concept using the accelerator of the MCP to prefetch the following hash to be computed of a Merkle tree, which resulted in a typical application speedup of 1.118 and

a 5x data access speedup in section 6.6.2. We also confirm the recommendations provided in [76], while adding newer ones for the use of the new MCP platform.

This research is a first step towards low latency resource allocation in HPC environments. As future work, it is planned to use the on-board FPGA of the MCP to study cache managements techniques on top of using lock-free and wait-free programming to obtain lower latency and more deterministic processing for more real-time applications.

CHAPITRE 7 DISCUSSION GÉNÉRALE

Dans ce chapitre, nous revisitons les objectifs fixés au Chapitre 1.3 et discutons de la manière dont nous les avons atteints. Nous observons également l'impact significatif du travail exposé dans la thèse pour l'industrie en décrivant les contributions, les résultats et les limites spécifiques de notre recherche.

7.1 Synthèse des travaux

Notre stratégie de définition des jalons et d'orientation de nos recherches pour atteindre les objectifs a été discutée plus tôt dans le Chapitre 3. Dans cette section, nous approfondissons la discussion sur la progression de notre recherche en identifiant des exemples spécifiques de nos travaux qui nous ont aidés à atteindre les jalons mis en place.

Équilibrage de la charge de travail et ordonnancement dynamique de tâches. Nous commençons le travail de la thèse avec le développement d'algorithmes au niveau de la deuxième couche pour une distribution équilibrée des charges de travail aux processeurs dans le but de maximiser l'utilisation des processeurs dans une grappe de calcul, tout en fournissant une bande passante garantie pour le transfert de données entre les nœuds de calcul et une mémoire cache réservée pour permettre l'exécution des tâches avec une faible variance. Nous créons une bibliothèque complète basée sur la bibliothèque DPDK d'Intel. Nous proposons une solution utilisant des systèmes de "streaming" avec des techniques de traitement parallèle. Cela nous a permis d'avoir une allocation de tâches dynamique, tout en diminuant la variance et l'exécution du traitement des groupes de tâches et ce à une granularité plus fine. En effet, nous observons que nous avons une amélioration de 11% de la bande passante, 24% sur l'utilisation des coeurs de calculs et d'un facteur de 2.27 sur le temps d'exécution moyen par tâche comparé à la solution implémentée par notre partenaire industriel. Cependant, nous observons que bien que nous ayons grandement amélioré notre grappe de calcul, nous nous rendons compte qu'afin d'avoir des meilleurs résultats au niveau de la latence et de la variance de traitement, l'introduction d'accélérateurs matériels est nécessaire.

Calculs et communications à faible variance avec un plan de contrôle implémenté par matériel. Dans ce premier article, nous commençons où nous avons laissé au chapitre précédent. Nous déchargeons l'algorithme d'allocation de tâches développé dans le chapitre

précédent vers un FPGA servant d'accélérateur du plan de contrôle. L'utilisation du HRA nous permet d'améliorer le traitement de tâches de l'ordonnanceur par un facteur de 760 en comparaison de la version logicielle, tout en diminuant la latence effective dans le cas d'une décharge par PCIe en utilisant le bus de contrôle QPI du CPU. L'architecture hybride permet donc de rencontrer les requis des nouvelles normes de télécommunication, tout en rendant la solution d'utiliser des COTS plus flexible lors de mises à jour futures, en réduisant les coûts. Nous pensons donc que l'utilisation d'architectures hétérogènes CPU-FPGA améliorera grandement le plan de contrôle des grappes de calcul visant la virtualisation de protocoles de télécommunications.

Tâches à faible variance d'exécution sur un processeur multicœur à l'aide d'un plan de contrôle matériel. Continuant avec le premier article publié, nous observons qu'en déchargeant les tâches vers l'accélérateur matériel a l'effet d'améliorer la latence et variance d'exécution des tâches, qui est alors limitée seulement par la variabilité du transfert. Nous comparons dans l'article, les différences de variabilité, de latence et de bande passante de transfert lors de l'utilisation de PCIe et QPI. Nous concluons que QPI a une plus grande chance d'être plus utile pour notre application, et que les performances sont surtout limitées par le remplissage de la cache et la localité des données. Il semble aussi que dans le pire cas, c'est-dire dans le cas d'un "cache miss" la solution exploitant QPI semble supérieure à celle exploitant PCIe quand on considère les contraintes des applications visées.

Caractérisation et validation du système. Le deuxième article publié inclut au Chapitre 6 examine une toute nouvelle architecture combinant un FPGA d'Intel avec un de leur processeur Xeon. Nous observons que la plateforme CPU-FPGA en socle offre des fonctionnalités d'accès de cache à partir de la partition matérielle du CPU, côté FPGA. Ça nous a permis de valider que les spécifications du système nous permettent d'avoir des performances adéquates pour une éventuelle virtualisation de la norme LTE basée sur les résultats exposés dans le premier article de la thèse. Nous développons aussi des règles et une ligne directrice expliquant comment utiliser ce genre de plateforme pour de futurs utilisateurs, en plus de formuler des recommandations sur ce qui pourrait être amélioré lors d'une prochaine itération.

Interconnexion à faible latence et variance pour une grappe informatique en temps réel. Pour finir dans le deuxième article publié, nous commençons l'étude d'un système de communication basse latence et faible variance intégré au sein de notre grappe hétérogène. Nous proposons une nouvelle technique de gestion de cache de CPU grâce à un

accélérateur matériel. Nous observons qu'en utilisant notre nouveau système de gestion, nous pouvons avoir une communication CPU-FPGA à travers la cache, oeuvrant avec une haute bande passante (> 20GBps) tout en ayant une latence sous la microseconde pour des données de la taille d'une ligne de cache. Ces performances sont seulement invalidées dans le cas d'un "cache miss". Nous étudions la proposition avec une étude de cas, où nous implémentons le traitement d'un Merkle Tree, où la communication des informations se fait entre le CPU et le FPGA, avec un traitement sur l'accélérateur matériel. Nous observons une accélération par un facteur de 5 du temps d'accès aux données, juste avec notre système de communication amélioré.

7.2 Impact de la recherche

Le travail contribue dans un domaine à la fine pointe de la technologie et s'attaque à l'accélération de grappe de calcul pour la latence. Compte tenu, des implications monétaires associées avec les technologies au croisement des ces domaines de recherche, il existe peu d'informations dévoilées par les industriels. Cela rend difficile de concrètement évaluer l'impact de la recherche dans son ensemble.

Due à l'évolution constante des normes de communication, la demande pour des solutions permettant de réduire les CAPEX et OPEX ne cessera de grandir. Les requis en latence et bandes passantes sont aussi de plus en plus importants, prenons comme exemple la 5G qui demande 10 fois la vitesse et réduit d'un ordre de grandeur la latence par rapport à la norme LTE. L'aspect d'ouverture et généralité (avec l'utilisation de COTS) de notre système contribue bien à la demande de réduction des CAPEX dans le cas de normes évolutives. L'accélération résultant de systèmes hétérogènes moins demandant en ressources énergétiques comme les FPGA positionne notre travail comme une solution viable pour la réduction d'OPEX des opérateurs. Ajoutons que les résultats obtenus à travers cette thèse se prêtent facilement aux requis des futures normes de télécommunications.

Dans le contexte de l'ère des plans contrôle programmables, notre recherche offre des alternatives en termes d'accélération d'allocation de tâches dans une grappe de calcul informatique en réduisant la latence d'exécution des différents groupes de tâches nécessitant à une granularité plus fine exploitée de façon dynamique. Un exemple d'application qui requière les technologies proposées est le traitement de trames d'enregistrement dans un réseau 5G. Les technologies proposées permettent l'insertion d'un plus grand nombre d'entrées dans les tables du dataplane, et ce avec une latence réduite.

En ce qui a trait aux gains des performances obtenus grâce aux travaux effectués au niveau de la cache de CPU par le biais des FPGA, il s'agit d'un domaine de recherche nécessitant plus d'implication des scientifiques du domaine. Nous pensons avoir été pionnier et que de nombreuses autres solutions possibles ne demandent qu'a être explorées. Nous estimons donc notre contribution comme significative en termes d'utilisation d'accélérateurs pour la gestion et l'ordonnancement de tâche dans une cache de x86.

Cela marque un changement majeur dans l'évolution future de l'accélération d'ordonnacement des tâches dans une grappe de calcul. De par le fait que notre travail avec le partenaire industriel est sur une application réelle et que notre solution a été complètement déployée sur une grappe chez le partenaire suggère le potentiel d'une adoption industrielle rapide de nos recherches. En dehors de cela, notre bibliothèque logicielle pourrait être mise en tant que logiciel libre afin de permettre à d'autres chercheurs de continuer à avancer le domaine sans avoir le besoin de la plateforme hétérogène disponible ou de compétences en programmation parallèle sur une grappe multinoeuds.

Nous pensons aussi que l'utilisation d'architecture hybride telle que le CPU-FPGA (HARP) pourrait avoir de grands avantages au niveau du "dataplane". Pouvant offrir une solution "tout-en-un" pour une haute bande passante, basse latence et plus grande expansibilité que des COTS serveurs, si combinés avec des cartes NIC basse latence. L'expansion du système exposé dans cette recherche devra être explorée dans des travaux futurs, car le potentiel semble grandissant.

7.3 Limitations de la solution proposée

Même si nous nous efforçons d'adopter une approche approfondie du logiciel et du matériel en matière d'accélération pour un système d'allocation de ressources dynamique à granularité fine, certaines limitations de notre recherche existent toujours.

De nombreux aspects du prototypage de FPGA peuvent mettre un frein a la conception d'un système hétérogène, nous pensons en particulier au :

— **Temps de conception**, rendant le prototypage long par rapport au développement logiciel.
— **L'adaptabilité et reconfigurabilité**, les normes de télécommunications qui changent vite et l'adaptation de la solution pourrait être couteuse en temps, argent et ressources matérielles.

Cependant, nous voyons surtout une limitation sur la portabilité de la solution. Tous les FPGA ne sont pas adéquats pour la solution proposée, les bonnes caracatéristiques doivent

être rencontrées pour un design fonctionnel. Les ressources exposées dans le Chapitre 5 rendent la majorité des FPGA disponibles sur le marché trop petit et imposent l'usage des plus gros FPGA de chaque famille des différents fournisseurs, tel qu'Intel et Xilinx. Ainsi afin d'obtenir les résultats exposés dans le Chapitre 6, il faut une version très spécifique et présentement peu disponible dans l'offre de SoC d'Intel FPGA. La famille Agilex des FPGA d'Intel devrait pallier au problème de rareté dans les années suivant la sortie de cette thèse.

Notamment en lien avec les fonctionnalités exposées que l'utilisation de la cache de processeur par l'accélérateur FPGA peut avoir des effets néfastes et polluer les données pour le reste des applications. Plus d'investigations devront être faites sur les différentes techniques de réserve de zones du dernier niveau de cache, ainsi que l'impact de la réduction de zone disponible sur l'ensemble des applications.

Pour finir, tout le travail fait et exposé à travers cette thèse peut avoir un impact limité si le reste de l'architecture réseau n'a pas été mise en place adéquatement. Aucune accélération ne pourra pallier une communication lente et non déterministe entre les noeuds. De plus, la variabilité de réception des messages pourrait avoir des résultats néfastes sur la solution proposée, car utiliser des zones de caches pour des données clairsemées, dépendantes sur des intervalles irréguliers, risque d'encore une fois affecter les résultats d'un système de grappe hétérogène.

CHAPITRE 8 CONCLUSION ET RECOMMANDATIONS

8.1 Conclusion

À une époque où la quantité de données utilisées et transférées ne cesse d'augmenter, la nécessité de calcul plus rapide suit la tendance [70]. La stagnation de la fréquence d'opération des processeurs, en raison des contraintes d'alimentation et des problèmes de courants de fuite, a poussé les fournisseurs de matériel à augmenter le parallélisme dans leurs processeurs pour améliorer les performances de nombreuses applications. Cela a conduit à des architectures hétérogènes exploitant des accélérateurs parallèles, associés à des processeurs compatibles x86, qui sont modestement parallèles. Cependant, la nécessité d'une latence très faible et de performances garanties, tout en conservant un débit élevé, n'a pas été largement étudiée ces dernières années.

Nous estimons que nous sommes des précurseurs dans le domaine en explorant les améliorations sur la variance d'exécution et la latence des grappes de calcul. Nous démontrons que l'intégration d'accélérateurs matériels dans des noeuds hétérogènes a le potentiel d'améliorer les grappes d'unités de traitement afin de rencontrer les requis nécessaires pour la visualisation des protocoles de communication de pointe. Nous le faisons en proposant des techniques d'amélioration au niveau système, logicielles et matérielles, aidant à améliorer la latence et la bande passante pour chaque noeud et ultimement chaque grappe. Puis, nous promouvons des architectures hybrides, permettant la gestion de cache des CPU par accélérateur, créant ainsi un nouveau système de communication à très basse latence avec une faible variance quand on compare aux solutions existantes.

Les recherches que nous avons effectuées recoupent divers domaines de l'informatique : les systèmes d'exploitation et d'exécution, les accélérateurs matériels et la réseautique. Nous espérons que les travaux présentés dans cette thèse contribueront à faire avancer le domaine de l'accélération de l'ordonnancement de tâches dans des grappes de calcul hétérogènes.

8.2 Améliorations futures

Avec pour but de palier aux limitations de la solution proposée et d'avoir une solution complète et robuste, les travaux futurs porteront sur la composante réseau de la solution. C'est-à-dire étudier et ultimement, réduire la latence de communication entre différents noeuds d'une grappe de calcul pour augmenter le déterminisme des messages envoyés. Une fois les spécifications adéquates atteintes, nous finirions par l'état de la consolidation de ressources

au sein de la grappe. Les sous-sections ci-dessous explorent un peu plus en détail les futurs axes de recherche à explorer.

8.2.1 Interconnexion réseau

L'interface de passage de messages (MPI) est la norme de facto pour la communication entre les processus exécutés sur une grappe. MPI est un système qui comprend un riche ensemble d'API pour échanger des données entre les nœuds et de nombreuses implémentations sont disponibles auprès de la communauté offrant des logiciels libres et auprès des fournisseurs de cartes d'interface réseau (NIC). Les implémentations de base utilisent généralement la pile logicielle TCP/IP et elles sont très loin d'être considérées comme invariante dans le temps. Les systèmes commercialement disponibles basés sur MPI fournissent des accélérateurs MPI hautement optimisés qui sont utilisés dans les produits d'interconnexion. La fonctionnalité matérielle la plus courante est un moteur RDMA qui permet le transfert de données entre les nœuds de calcul sans intervention logicielle. Ces systèmes basés sur des MPI optimisés et accélérés par le matériel offrent de meilleures performances, car ils sont étroitement liés à la mise en œuvre matérielle de les cartes réseau. Il faudrait donc d'abord évaluer les performances de ce que la carte réseau non cohérente actuelle et leurs logiciels associés peuvent prendre en charge. Ensuite, nous étudierons comment tirer parti de l'interface cohérente en cache pour proposer et développer une nouvelle architecture déterministe et à faible latence pour une carte réseau. Un modèle logiciel de cette carte réseau sera utilisé pour valider l'architecture et le prototype de matériel informatique utilisant des FPGA sera développé. Enfin, cette accélération lente et déterministe est censée être une ressource contrôlée par l'algorithme d'allocation de ressources développé dans le cadre de l'axe 1.

8.2.2 Consolidation dynamique des ressources

Au cœur de la troisième couche, nous devrions développer des solutions de consolidation de ressources dynamique capables d'ajuster l'équipement informatique pour automatiser les charges de travail en désactivant les serveurs disponibles et en utilisant des fonctions d'économie d'énergie, telles que le *Dynamic Voltage and Frequency Scaling*. Il est nécessaire de développer des algorithmes efficaces et d'effectuer une évaluation de l'optimalité. D'autres techniques d'optimisation et de contrôle seraient également envisagées.

RÉFÉRENCES

[1] J. Wu, S. Rangan et H. Zhang, *Green communications : Theoretical Fundamentals, Algorithms, and Applications*, CRC Press, 2012.

[2] J. Malmodin *et al.*, "Greenhouse gas emissions and operational electricity use in the ICT and entertainment and media sectors," *Journal of Industrial Ecology*, vol. 14, nº. 5, p. 770–790, 2010. [En ligne]. Disponible : https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1530-9290.2010.00278.x

[3] H. Wen, P. K. Tiwary et T. Le-Ngoc, *Wireless Virtualization*, ser. SpringerBriefs in Computer Science. Springer International Publishing, 2013.

[4] Z. Kong *et al.*, "ebase : A baseband unit cluster testbed to improve energy-efficiency for cloud radio access network," dans *2013 IEEE International Conference on Communications (ICC)*, June 2013, p. 4222–4227.

[5] C. Liu *et al.*, "The case for re-configurable backhaul in cloud-ran based small cell networks," dans *2013 Proceedings IEEE INFOCOM*, April 2013, p. 1124–1132.

[6] C. Chen, "C-RAN : The road towards green radio access network," *White paper*, 2011. [En ligne]. Disponible : http://ss-mcsp.riit.tsinghua.edu.cn/cran/C-RAN%20ChinaCOM-2012-Aug-v4.pdf

[7] M. Gémieux, "Analyse de faisabilité de l'implantation d'un protocole de communication sur processeur multicoeurs," Masters, École Polytechnique de Montréal, avr. 2015. [En ligne]. Disponible : https://publications.polymtl.ca/1709/

[8] Intel, "Data Plane Development Kit Overview." [En ligne]. Disponible : https://software.intel.com/en-us/articles/data-plane-development-kit-overview

[9] R. Buyya, A. Beloglazov et J. Abawajy, "Energy-efficient management of data center resources for cloud computing : A vision, architectural elements, and open challenges," *PDPTA 2010 : The 2010 International Conference on Parallel and Distributed Processing Techniques and Applications*, 06 2010.

[10] I. Monga, E. Pouyoul et C. Guok, "Software-defined networking for big-data science - architectural models from campus to the wan," dans *2012 SC Companion : High Performance Computing, Networking Storage and Analysis*, Nov 2012, p. 1629–1635.

[11] M. Lin *et al.*, "Dynamic right-sizing for power-proportional data centers," dans *2011 Proceedings IEEE INFOCOM*, April 2011, p. 1098–1106.

[12] M. Chiang *et al.*, "Layering as optimization decomposition : A mathematical theory of network architectures," *Proceedings of the IEEE*, vol. 95, nº. 1, p. 255–312, janv. 2007.

[13] V. Ganti et G. Ghatikar, “Smart grid as a driver for energy-intensive industries : A data center case study,” dans *Grid-Interop 2012*, Irving, TX, déc. 2012.

[14] Wei Xu *et al.*, “Predictive control for dynamic resource allocation in enterprise data centers,” dans *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006*, April 2006, p. 115–126.

[15] Y. Guo *et al.*, “Automated and agile server parameter tuning by coordinated learning and control,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, n°. 4, p. 876–886, April 2014.

[16] R. Urgaonkar *et al.*, “Dynamic resource allocation and power management in virtualized data centers,” dans *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, April 2010, p. 479–486.

[17] B. Jennings et R. Stadler, “Resource management in clouds : Survey and research challenges,” *Journal of Network and Systems Management*, vol. 23, n°. 3, p. 567–619, 2015.

[18] M. Sharifi, H. Salimi et M. Najafzadeh, “Power-efficient distributed scheduling of virtual machines using workload-aware consolidation techniques,” *The Journal of Supercomputing*, vol. 61, n°. 1, p. 46–66, juill. 2012.

[19] Intel, “Data Plane Development Kit : Programmer’s Guide.” [En ligne]. Disponible : https://www.intel.com/content/www/us/en/embedded/technology/packet-processing/dpdk/dpdk-programmers-guide.html

[20] “FPGAs Glimmer on the HPC Horizon, Glint in Hyperscale Sun,” nov. 2015. [En ligne]. Disponible : http://www.nextplatform.com/2015/11/17/fpgas-glimmer-on-the-hpc-horizon-glint-in-hyperscale-sun/

[21] B. Belean, S. Pogacian et A. Bot, “FPGA based hardware architectures for high performance computing applications,” dans *Tier 2 Federation Grid, Cloud High Performance Computing Science (RO-LCG), 2012 5th Romania*, oct. 2012, p. 11–14.

[22] N. Oliver *et al.*, “A reconfigurable computing system based on a cache-coherent fabric,” dans *2011 International Conference on Reconfigurable Computing and FPGAs*, nov. 2011, p. 80–85.

[23] “TOP500 Supercomputer Sites June 2016,” juin 2016. [En ligne]. Disponible : https://www.top500.org/lists/2016/06/

[24] L. Wang *et al.*, “Scientific cloud computing : Early definition and experience,” dans *10th IEEE International Conference on High Performance Computing and Communications, 2008. HPCC ’08*, sept. 2008, p. 825–830.

[25] K. Keahey, “Cloud computing for science,” dans *Scientific and Statistical Database Management*, M. Winslett, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, vol. 5566, p. 478–478.

[26] W. Seo *et al.*, “Big or little : A study of mobile interactive applications on an asymmetric multi-core platform,” dans *2015 IEEE International Symposium on Workload Characterization (IISWC)*, oct. 2015, p. 1–11.

[27] D. Patterson, “The parallel computing landscape : a Berkeley view,” dans *2007 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, août 2007, p. 231–231.

[28] R. Inta, D. J. Bowman et S. M. Scott, “The ‘Chimera’ : an off-the-shelf CPU/GPGPU/FPGA hybrid computing platform,” *International Journal of Reconfigurable Computing*, vol. 2012, p. e241439, mars 2012.

[29] F. A. Escobar, X. Chang et C. Valderrama, “Suitability analysis of FPGAs for heterogeneous platforms in HPC,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, nº. 2, p. 600–612, févr. 2016.

[30] O. Segal *et al.*, “High level programming framework for FPGAs in the data center,” dans *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, sept. 2014, p. 1–4.

[31] M. Dashtbani, A. Rajabzadeh et M. Asghari, “High level synthesis as a service,” dans *2015 5th International Conference on Computer and Knowledge Engineering (ICCKE)*, oct. 2015, p. 331–336.

[32] T. El-Ghazawi *et al.*, “The Promise of High-Performance Reconfigurable Computing,” *Computer*, vol. 41, nº. 2, p. 69–76, 2008.

[33] J. Hursey, J. M. Squyres et T. Dontje, “Locality-aware parallel process mapping for multi-core HPC systems,” dans *2011 IEEE International Conference on Cluster Computing*, sept. 2011, p. 527–531.

[34] R. Tessier, K. Pocek et A. DeHon, “Reconfigurable computing architectures,” *Proceedings of the IEEE*, vol. 103, nº. 3, p. 332–354, mars 2015.

[35] A. Putnam *et al.*, “A reconfigurable fabric for accelerating large-scale datacenter services,” dans *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, juin 2014, p. 13–24.

[36] P. Chow, “Why put FPGAs in your CPU socket ?” dans *2013 International Conference on Field-Programmable Technology (FPT)*, déc. 2013, p. 3–3.

[37] “High-performance, energy-efficient platforms using in-socket FPGA accelerators,” ser. FPGA ’09, New York, NY, USA.

[38] F. Lin, H. Wang et J. Bian, “HW/SW interface synthesis based on Avalon bus specification for Nios-oriented SoC design,” dans *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.*, déc. 2005, p. 305–306.

[39] P. Waldeck et N. Bergmann, “Evaluating software and hardware implementations of signal-processing tasks in an FPGA,” dans *2004 IEEE International Conference on Field-Programmable Technology, 2004. Proceedings*, déc. 2004, p. 299–302.

[40] V. Rajagopalan *et al.*, “Xilinx Zynq-7000 EPP : An extensible processing platform family,” dans *2011 IEEE Hot Chips 23 Symposium (HCS)*, août 2011, p. 1–24.

[41] C. Steffen et G. Genest, “Nallatech in-socket FPGA Front-Side Bus Accelerator,” *Computing in Science Engineering*, vol. 12, n°. 2, p. 78–83, mars 2010.

[42] R. Mancuso *et al.*, “A hardware architecture to deploy complex multiprocessor scheduling algorithms,” dans *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, août 2014, p. 1–10.

[43] “FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions.” [En ligne]. Disponible : http://www.freertos.org/index.html

[44] P. Laplante, *Real-Time Systems Design and Analysis*. Wiley, 2004.

[45] A. Munshi, “The OpenCL specification,” dans *2009 IEEE Hot Chips 21 Symposium (HCS)*, août 2009, p. 1–314.

[46] C. Augonnet *et al.*, “Starpu : a unified platform for task scheduling on heterogeneous multicore architectures,” *Concurrency and Computation : Practice and Experience*, vol. 23, n°. 2, p. 187–198, 2011.

[47] R. Mancuso *et al.*, “Real-time cache management framework for multi-core architectures,” dans *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, avr. 2013, p. 45–54.

[48] D. Lo *et al.*, “Heracles : Improving resource efficiency at scale,” dans *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, juin 2015, p. 450–462.

[49] A. J. Conejo *et al.*, *Decomposition techniques in mathematical programming : engineering and science applications*. Springer Science & Business Media, 2006.

[50] V. Boyer, D. El Baz et M. Elkihel, “Dense dynamic programming on multi gpu,” dans *2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*. IEEE, 2011, p. 545–551.

[51] J. L. Jerez, G. A. Constantinides et E. C. Kerrigan, “Fpga implementation of an interior point solver for linear model predictive control,” dans *2010 International Conference on Field-Programmable Technology*, Dec 2010, p. 316–319.

[52] “Scheduling tasks over multicore machines enhanced with acelerators : A runtime system’s perspective,” Ph.D. dissertation.

[53] I. Mavroidis *et al.*, “ECOSCALE : Reconfigurable computing and runtime system for future exascale systems,” dans *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, mars 2016, p. 696–701.

[54] B. Radunovic et J. Y. L. Boudec, “A Unified Framework for Max-Min and Min-Max Fairness With Applications,” *IEEE/ACM Transactions on Networking*, vol. 15, nº. 5, p. 1073–1083, oct. 2007.

[55] D. Nace et M. Pioro, “Max-min fairness and its applications to routing and load-balancing in communication networks : a tutorial,” *IEEE Communications Surveys Tutorials*, vol. 10, nº. 4, p. 5–17, 2008.

[56] S. Keshav, *An Engineering Approach to Computer Networking : ATM Networks, the Internet, and the Telephone Network*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1997.

[57] D. Bertsekas et R. Gallager, *Data Networks (2Nd Ed.)*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1992.

[58] B. McCormick *et al.*, “Real time alpha-fairness based traffic engineering,” dans *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’14. New York, NY, USA : ACM, 2014, p. 199–200.

[59] J. Celaya et U. Arronategui, “Fair scheduling of bag-of-tasks applications on large-scale platforms,” *Future Gener. Comput. Syst.*, vol. 49, nº. C, p. 28–44, août 2015.

[60] “Project Catapult.” [En ligne]. Disponible : https://www.microsoft.com/en-us/research/project/project-catapult/

[61] A. Putnam *et al.*, “A reconfigurable fabric for accelerating large-scale datacenter services,” *SIGARCH Comput. Archit. News*, vol. 42, nº. 3, p. 13–24, juin 2014.

[62] “An Introduction to the Intel®QuickPath Interconnect.” [En ligne]. Disponible : https://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html

[63] Intel, “Intel® Threading Building Blocks Developer Guide.” [En ligne]. Disponible : https://software.intel.com/en-us/tbb-user-guide

[64] MIT, “StreamIt.” [En ligne]. Disponible : http://groups.csail.mit.edu/cag/streamit/

[65] B. Mutnury *et al.*, "QuickPath Interconnect (QPI) design and analysis in high speed servers," dans *19th Topical Meeting on Electrical Performance of Electronic Packaging and Systems*, oct. 2010, p. 265–268.

[66] Intel, "Task-Based Programming." [En ligne]. Disponible : https://software.intel.com/en-us/node/506100

[67] "Optimization techniques for task allocation and scheduling in distributed multi-agent operations," Ph.D. dissertation.

[68] M. I. Gordon, W. Thies et S. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs," *ACM SIGPLAN Notices*, vol. 41, nº. 11, p. 151–162, 2006.

[69] F. da Silva et H. Senger, "Scalability limits of bag-of-tasks applications running on hierarchical platforms," *Journal of Parallel and Distributed Computing*, vol. 71, p. 788–801, 06 2011.

[70] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, nº. 1, p. 82–85, 1998.

[71] D. C. Marinescu, "Parallel and distributed computing : Memories of time past and a glimpse at the future," dans *2014 IEEE 13th International Symposium on Parallel and Distributed Computing*, juin 2014, p. 14–15.

[72] V. W. Lee, E. Grochowski et R. Geva, "Performance benefits of heterogeneous computing in HPC workloads," dans *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, mai 2012, p. 16–26.

[73] M. Véstias et H. Neto, "Trends of CPU, GPU and FPGA for high-performance computing," dans *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, sept. 2014, p. 1–6.

[74] "Intel begins shipping Xeon chips with FPGA accelerators," avr. 2016. [En ligne]. Disponible : http://www.eweek.com/servers/intel-begins-shipping-xeon-chips-with-fpga-accelerators.html

[75] S. Talwar *et al.*, "Enabling technologies and architectures for 5g wireless," dans *2014 IEEE MTT-S International Microwave Symposium (IMS2014)*. IEEE, 2014, p. 1–4.

[76] Y. k. Choi *et al.*, "A quantitative analysis on microarchitectures of modern CPU-FPGA platforms," dans *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, juin 2016, p. 1–6.

[77] I. Corporation, "An Introduction to the Intel® QuickPath Interconnect," 2009. [En ligne]. Disponible : http://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html

[78] A. Benoit *et al.*, “Scheduling concurrent bag-of-tasks applications on heterogeneous platforms,” *IEEE Transactions on Computers*, vol. 59, nº. 2, p. 202–217, 2009.

[79] R. Chen, S. Siriyal et V. Prasanna, “Energy and memory efficient mapping of bitonic sorting on fpga,” dans *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015, p. 240–249.

[80] “Amazon EC2 F1 Instances.” [En ligne]. Disponible : https://aws.amazon.com/ec2/instance-types/f1/

[81] Y. Umuroglu *et al.*, “Finn : A framework for fast, scalable binarized neural network inference,” dans *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, p. 65–74.

[82] M. Courbariaux, Y. Bengio et J.-P. David, “Binaryconnect : Training deep neural networks with binary weights during propagations,” dans *Advances in neural information processing systems*, 2015, p. 3123–3131.

[83] J. W. Lockwood *et al.*, “A low-latency library in FPGA hardware for high-frequency trading (HFT),” dans *2012 IEEE 20th Annual Symposium on High-Performance Interconnects*, Aug 2012, p. 9–16.

[84] “Intel Begins Shipping Xeon Chips With FPGA Accelerators,” avr. 2016. [En ligne]. Disponible : http://www.eweek.com/servers/intel-begins-shipping-xeon-chips-with-fpga-accelerators.html

[85] PCI-SIG, “PCI Express® Base Specification Revision 3.0,” nov. 2010. [En ligne]. Disponible : http://composter.com.ua/documents/PCI_Express_Base_Specification_Revision_3.0.pdf

[86] A. Caulfield *et al.*, “A Cloud-Scale Acceleration Architecture,” *Microsoft Research*, oct. 2016. [En ligne]. Disponible : https://www.microsoft.com/en-us/research/publication/configurable-cloud-acceleration/

[87] M. Gémieux *et al.*, “A cache-coherent heterogeneous architecture for low latency real time applications,” dans *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, May 2017, p. 176–184.

[88] D. Koeplinger *et al.*, “Automatic generation of efficient accelerators for reconfigurable hardware,” dans *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, p. 115–127.

[89] P. Colangelo *et al.*, “Application of convolutional neural networks on intel xeon processor with integrated fpga,” dans *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2017, p. 1–7.

[90] S. Krishnan *et al.*, "Accelerator templates and runtime support for variable precision cnn," dans *CISC Workshop*, 2017.

[91] Intel, "Intel QuickAssist Technology Accelerator Abstraction Layer (AAL)," 2007. [En ligne]. Disponible : https://blog-assets.oss-cn-shanghai.aliyuncs.com/18951/6103fadf4dd3a0dfbd0d637308a94b8e99e799d2.pdf

[92] ——, "Open Programmable Acceleration Engine (OPAE) C API Programming Guide," févr. 2017. [En ligne]. Disponible : https://www.altera.com/en_US/pdfs/literature/ug/opae-programming-guide.pdf

[93] "SDAccel Development Environment." [En ligne]. Disponible : https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[94] Intel, "Intel FPGA SDK for OpenCL - Overview." [En ligne]. Disponible : https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html

[95] J. Stuecheli *et al.*, "Capi : A coherent accelerator processor interface," *IBM Journal of Research and Development*, vol. 59, n°. 1, p. 7 :1–7 :7, Jan 2015.

[96] "FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions." [En ligne]. Disponible : http://www.freertos.org/index.html

[97] B. K. Kim et K. G. Shin, "Scalable hardware earliest-deadline-first scheduler for atm switching networks," dans *Proceedings Real-Time Systems Symposium*, Dec 1997, p. 210–218.

[98] P. Kuacharoen, M. Shalan et V. J. Mooney, "A configurable hardware scheduler for real-time systems." dans *Engineering of Reconfigurable Systems and Algorithms*. Citeseer, 2003, p. 95–101.

[99] N. Gupta *et al.*, "A hardware scheduler for real time multiprocessor system on chip," dans *2010 23rd International Conference on VLSI Design*, Jan 2010, p. 264–269.

[100] D. Gregorek, C. Osewold et A. Garcia-Ortiz, "A scalable hardware implementation of a best-effort scheduler for multicore processors," dans *2013 Euromicro Conference on Digital System Design*, Sept 2013, p. 721–727.

[101] Intel, "Intel®FPGA IP Core Cache Interface (CCI-P)," sept. 2017. [En ligne]. Disponible : https://01.org/sites/default/files/downloads/opae/intel-fpga-ip-cci-p-inter-spec-external-0.5.pdf

[102] "Universitat Paderborn - University." [En ligne]. Disponible : http://www.uni-paderborn.de/en/university/

[103] Intel, “Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE)- User Guide,” Rapport technique. [En ligne]. Disponible : https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug-ase.pdf

[104] “Questa Advanced Simulator - Mentor Graphics.” [En ligne]. Disponible : https://www.mentor.com/products/fv/questa/

[105] “Intel®Quartus®Prime Software - What’s New in Quartus Prime.” [En ligne]. Disponible : https://www.altera.com/products/design-software/fpga-design/quartus-prime/what-s-new.html

[106] J. W. Haskins et K. Skadron, “Memory reference reuse latency : Accelerated warmup for sampled microarchitecture simulation,” dans *2003 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS 2003.*, March 2003, p. 195–203.

[107] “Introduction to Cache Allocation Technology in the Intel®Xeon®Processor E5 v4 Family | Intel®Software.” [En ligne]. Disponible : https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[108] F. Cerqueira et B. Brandenburg, “A comparison of scheduling latency in linux, preempt-rt, and litmus rt,” dans *9th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications*. SYSGO AG, 2013, p. 19–29.

[109] M. Aichouch, J. C. Prévotet et F. Nouvel, “Evaluation of the overheads and latencies of a virtualized rtos,” dans *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2013, p. 81–84.

[110] X. Lu *et al.*, “Performance characterization and acceleration of big data workloads on openpower system,” dans *2017 IEEE International Conference on Big Data (Big Data)*, Dec 2017, p. 213–222.

[111] S. Nakamoto, “Bitcoin : A peer-to-peer electronic cash system,” 2009. [En ligne]. Disponible : https://bitcoin.org/bitcoin.pdf

[112] G. Becker, “Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis,” p. 28. [En ligne]. Disponible : https://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker_1.pdf

[113] F. Tschorsch et B. Scheuermann, “Bitcoin and beyond : A technical survey on decentralized digital currencies,” *IEEE Communications Surveys Tutorials*, vol. 18, n°. 3, p. 2084–2123, third quarter 2016.

[114] “wiki : The Ethereum Wiki -,” mai 2018, original-date : 2014-02-14T23 :05 :17Z. [En ligne]. Disponible : https://github.com/ethereum/wiki

[115] S. Noether, S. Noether et A. Mackenzie, “A note on chain reactions in traceability in CryptoNote 2.0,” p. 8, September 2014. [En ligne]. Disponible : https://lab.getmonero.org/pubs/MRL-0001.pdf

[116] R. P. McEvoy *et al.*, “Optimisation of the sha-2 family of hash functions on fpgas,” dans *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, March 2006, p. 317–322.