Génération automatique de plate-forme matérielles distribuées pour des applications de traitement du signal

Mathieu Leonel Mba

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ ET DE L'UNIVERSITÉ DE YAOUNDÉ I

Spécialité **INFORMATIQUE**

École Doctorale Informatique, Télécommunications et Électronique / ***Laboratoire d'Informatique de Paris 6*** (Paris)

Centre de Recherche et de Formation Doctorale en Sciences, Technologies et Géosciences / ***Laboratoire d'Informatique et Applications*** (Yaoundé)

GÉNÉRATION AUTOMATIQUE DE PLATEFORMES MATÉRIELLES DISTRIBUÉES POUR DES APPLICATIONS DE TRAITEMENT DU SIGNAL

Pour obtenir le grade de
DOCTEUR de SORBONNE UNIVERSITÉ et **DOCTEUR / Ph.D de l'UNIVERSITÉ DE YAOUNDÉ I**

Par

MBA Mathieu Leonel
MSc Informatique

Directeurs : Bertrand GRANADO, Professeur, Sorbonne Université
Paulin MELATAGIA YONTA, Chargé de Cours, Université de Yaoundé I

Co-Directeur : Julien DENOULET, Maître de Conférence, Sorbonne Université

Présentée et soutenue publiquement le 26 Septembre 2023

Devant le jury composé de :

M. Loïc LAGADEC	Professeur, ENSTA Bretagne	Rapporteur
M. Elie FUTE TAGNE	Associate Professor, Université de Dschang	Rapporteur
M. René NDOUNDAM	Associate Professor, Université de Yaoundé I	Examinateur
Mme. Isabelle PUAUT	Professeur, Université de Rennes	Présidente du Jury
M. Christian GAMOM	Associate Professor, ENSET, Université de Douala	Examinateur
M. Maurice TCHUENTE	Professeur Émérite, Université de Yaoundé I	Invité
M. Bertrand GRANADO	Professeur, Sorbonne Université	Directeur de Thèse
M. Paulin MELATAGIA	Maître de Conférences/CC, Université de Yaoundé I	Directeur de Thèse
M. Julien DENOULET	Maître de Conférences, Sorbonne Université	Co-Directeur de Thèse

Année académique 2022/2023

A ma très chère mère Juliette dit "mama na moan",
Autant de phrases aussi expressives soient-elles ne sauraient montrer le degré d'amour et d'affection que j'éprouve pour toi. Toi qui as toujours été mon principal modèle, ma principale source d'inspiration. Tu n'as cessé de me soutenir et de m'encourager durant toutes les années de mes études, tu as toujours été présente à mes cotés pour me galvaniser quand il fallait. En ce jour mémorable, pour moi ainsi que pour toi, reçoit ce travail en signe de ma vive reconnaissance et mon profond estime. Puisse le tout puissant te donner santé, bonheur et longue vie afin que je puisse te combler à mon tour.

Ma chère grand-mère Ernestine
Que ce modeste travail, soit l'expression des vœux et prières que tu n'as cessé de formuler à mon endroit. Que Dieu te préserve santé et longue vie.

Abstract

Local languages or mother tongues of individuals play an essential role in their fulfillment in their various socio-economic activities. African languages and specifically Cameroonian languages are exposed to disappearance in favor of foreign languages adopted as official languages after independence. This is why it is essential to digitalize and integrate them into the majority of dematerialized services for their sustainability. Speech recognition, widely used as a human-machine interface, can be not only a tool for integrating local languages into applications but also a tool for collecting and digitizing corpora. Embedded systems are the preferred environment for deploying applications that use this human-machine interface. This implies that it is necessary to take measures (through the reduction of the reaction time) to satisfy the real-time constraint very often met in this type of application. Two approaches exist for the reduction of the application's response time, namely parallelization and the use of efficient hardware architectures. In this thesis, we exploit a hybrid approach to reduce the response time of an application. We do this by parallelizing this application and implementing it on a reconfigurable architecture. An architecture whose implementation languages are known to be low-level. Moreover, given the multitude of problems posed by the implementation of parallel systems on reconfigurable architecture, there is a problem with design productivity for the engineer. In this thesis, to implement a real-time speech recognition system on an embedded system, we propose an approach for the productive implementation of parallel applications on reconfigurable architecture. Our approach exploits MATIP, a platform-based design tool, as an FPGA Overlay based on high-level synthesis. We exploit this approach to implement a parallel model of a feature extraction algorithm for the recognition of tonal languages (characteristic of the majority of Cameroonian languages). The experimentation of this implementation on isolated words of the Kóló language, in comparison to other implementations (software version and hardware IP), shows that our approach is not only productive in implementation time but also the obtained parallel application is efficient in processing time. This is the reason why we implemented XMATIP an extension of MATIP to make this approach compatible with hardware-software co-design and co-synthesis.

Keywords : MP-RSoC, Platform generation, MPI-2 RMA parallel applications, Signal processing applications, Automatic speech recognition

Résumé

Les langues locales ou langues maternelles propres aux individus jouent un rôle important pour leur épanouissement dans leurs différentes activités socio-économiques. Les langues africaines, et spécifiquement les langues camerounaises sont exposées à la disparition au profit des langues étrangères adoptées comme langues officielles au lendemain des indépendances. C'est la raison pour laquelle il est primordial de les numériser et les intégrer dans la majorité des services dématérialisés pour leur pérennisation. La reconnaissance vocale, largement utilisée comme interface d'interaction homme machine, peut être non seulement un outil d'intégration des langues locales dans les applications, mais aussi un outil de collecte et de numérisation des corpus. Les systèmes embarqués sont l'environnement par excellence de déploiement des applications qui exploitent cette interface d'interaction homme machine. Cela implique qu'il est nécessaire de prendre des mesures (à travers la réduction du temps de réponse) pour satisfaire la contrainte de temps réel très souvent rencontrée dans ce type d'application. Deux approches existent pour la réduction du temps de réponse des applications à savoir la parallélisation et l'usage des architectures matérielles efficaces. Dans cette thèse, nous exploitons une approche hybride pour réduire le temps de réponse d'une application. Nous le faisons par la parallélisation de cette application et sa mise en œuvre sur architecture reconfigurable. Une architecture dont les langages de mise en œuvre sont connus pour être de bas niveau. De plus, au vu de la multitude des problématiques posées par la mise en œuvre des systèmes parallèles sur architecture reconfigurable, il se pose un problème de productivité de l'ingénieur. Dans cette thèse, en vue de mettre en œuvre un système de reconnaissance vocale temps réel sur système embarqué, nous proposons, une approche de mise en œuvre productive d'applications parallèles sur architecture reconfigurable. Notre approche exploite MATIP un outil de conception orienté plateforme, comme FPGA Overlay basé sur la synthèse de haut niveau. Nous exploitons cette approche pour mettre en œuvre un modèle parallèle d'un algorithme d'extraction des caractéristiques pour la reconnaissance des langues à tons (caractéristique de la majorité des langues camerounaises). L'expérimentation de cette solution sur des mots isolés de la langue Kóló, en comparaison à d'autres propositions (version logicielle et IP matérielles), montre que, notre approche est non seulement productive en temps de mise en œuvre, mais aussi l'application parallèle obtenue est efficace en temps de traitement. C'est la raison pour laquelle nous avons mis en œuvre XMATIP une extension de MATIP pour rendre cette approche compatible à la co-conception et co-synthèse matérielle logicielle.

Mots clés : MP-RSoC, Génération de plateforme, Applications parallèles MPI-2 RMA, Applications de traitement du signal, Reconnaissance automatique de la parole.

Remerciements

D'entrée de jeu, je tiens à remercier le Professeur Maurice TCHUENTE qui a initié la collaboration qui a donné naissance à cette thèse (depuis mon Master dans le cadre du projet "Kit-Master" en 2016). Je remercie toute l'équipe qui a dirigé cette thèse : le Professeur Bertrand GRANADO, le Docteur Paulin MELATAGIA YONTA les Directeurs ; le Docteur Julien DENOULET le Co-Directeur, et le Professeur Roland Christian GAMOM, qui a été une personne ressource très importante pour cette thèse. A tous je leur suis très reconnaissant pour leur patience, leurs conseils, leur encadrement et leur motivation qui nous a permis de conduire jusqu'au bout ce projet de recherche.

Je remercie le laboratoire UMMISCO IRD-SU, le Laboratoire Informatique de Paris 6 (LIP6) qui ont soutenu financièrement toutes mes années de thèse. Un remerciement particulier aux membres de l'équipe UMMISCO, notamment le directeur Jean-Daniel ZUCKER, le coordonnateur du programme doctoral international (PDI) Christophe CAMBIER, dont les œuvres ont contribué qui m'ont accompagné dans la construction de ma culture scientifique. Je n'oublie pas le personnel administratif de l'IRD dont la réactivité a toujours permis à ce que mes séjours à Paris et mon financement se déroulent très bien. Je pense à M. Pedro VERGE-DEPRE, Mme. Felicia SEN NGWA, Mme. Colette ESSONO, Mme. Élisabeth PEREIRA, Mme. Kathy BAUMONT et Mme. Rolande ALTEMAIRE.

Je remercie mes camarades du PDI durant mes séjours en France pour leur participation à rendre l'ambiance détendue et fraternelle, de quoi permettre de travailler sereinement. Je pense à Papa Massar NIANE, Radegonde RUSAGARA, Julio CARDENAS, BA Ibrahima SIDIKI, Cassien Diabe NDIAYE, Jean Michel SARR, Lamiae SAIDI, Yvan GUIFO, Ahmad FALL, Badara SANE, Jorge Arturo FLORES VALIENTE, Mamadou LAMINE THIAM, Cyrine CHENAOUI, etc.

Je remercie les membres de l'équipe SYEL pour leur accueil, leur accompagnement et leurs conseils. Je pense à Khalil HACHICHA, Andrea PINNA, FERUGLIO Sylvain, Hichem SAHBI, Vallette FAROUK, BOUYER Manuel, RHOUNI Amine, etc. Je pense également à mes camarades de thèse et voisin de bureau, Thomas GARBAY, Sylvain TAKOUGANG, JANIAK Vincent, KHACEF Kahina, Li SONGLIN qui ont emprunté ce chemin rude mais enrichissant et avec qui j'ai partagé des moments inoubliables, échecs, succès, surtout nos discussions pendant les pauses déjeuner.

Je tiens aussi à remercier le chef du département d'informatique de l'Université de Yaoundé I pour sa disponibilité et son accompagnement administratif tout au long de cette thèse. Je remercie également mes enseignants de ce département qui m'ont tenu depuis mes premières années en filière informatique et m'ont capacité pour satisfaire les prérequis de cette thèse. Je pense notamment aux professeurs Marcel FOUDA, Roger ATSA, René NDOUNDAM, Norbert TSOPZE, Michael NKWENTI, aux docteurs Priso ESSAWE NDEDI, Eric Désiré KAMENI, Gilbert TINDO, Serge MOTO, Hippolyte TAPAMO, Etienne KOUOKAM, Jules WAKU, Donatien CHEDOM, Eric NGOKO, Patrick KAMGUEU, Valerie MONTHE, Serge EBELE, Adamou HAMZA, Rodrigue DOMGA et Fidel JIOMEKONG, etc.

Je remercie mes amis et camarades pour leurs conseils, encouragements, soutien, partage d'expérience, la communion fraternelle à travers des moments de distraction. Les rencontres avec eux permettent d'avoir plus de force pour aller de l'avant. Je pense en premier à la team des members : Littissia Foning, Adonice Ngansop, Claude KANYOU, Darius SAHA, Joseph NEMI et à leur coté, Donald VOUNDI, Raoul TCHASSEU, Boris FOKO. Mes amis depuis le

lycée : Raphael Arthur NDENDE, Romuald NDIGUI, Martiale ABEGA, Clemence YONKEU, Arlette BOYSALEL. A tous mes camarades de promos : les promotions "Innovation" et "Visionnaires" à l'Ecole Normale Supérieure de Yaoundé. A tous mes amis du JCR : Guy MESSI, Yves NGUEMBI, Frank JOB, Cedrick BIATAT, Dieudonné MOUYOUME, Wilfried MBOUMENE, etc. Je remercie aussi particulièrement à Arsene ZIEM, pour son accueil et soutien à Douala. Pour leur accueil et soutien à chacun de mes séjours en France, je remercie M. Blaise NDJINKEU, Ernest AGUOUZE.

Je remercie aussi toute ma famille qui m'a été d'un soutien incommensurable. Il s'agit en premier de mes parents, Grand mère Ernestine. Ensuite mes frères et sœurs, notamment : Martial, Daisy, Marie-Reine, Freddy, Rodrigue, Bibiche, Nicaise, Nadège, Sonia, Gyslaine, Sorel. Je n'oublierai pas mes oncles et tantes : Mama Marie, Mama Rose, Angeline, Ernestine, Dorothe, Gustave, etc.

Je remercie tous ceux qui m'ont été d'un quelconque soutient durant mes années de thèse et dont les noms ne figurent pas dans les paragraphes précédents, je vous prie de recevoir mes sincères excuses et j'espère pouvoir vous adresser mes remerciements de vive voix par un autre canal.

Je me permets de conclure cette section en remerciant la source par qui tout ceci a été possible, merci à Dieu mon Créateur.

Table des matières

Table des figures

Liste des tableaux

Chapitre

1 Introduction générale

Sommaire

1.1 Introduction

Nelson MANDELA a dit : *"Si vous parlez à un homme dans une langue qu'il comprend, vous parlez à sa tête. Si vous lui parlez dans sa langue, vous parlez à son cœur."*. C'est dire combien l'usage de la langue propre à une personne pour ses différentes activités socio-économiques occupe une place importante pour la qualité et la pertinence de celles-ci. L'environnement socio-économique actuel est fortement numérisé, avec la majeure partie des services socio-économiques dématérialisée, et l'environnement de vie de plus en plus connecté avec des objets dits *"intelligents"* qui meublent le quotidien des individus. Dans cet écosystème d'objets intelligents, la reconnaissance vocale occupe une place de choix, étant l'un des modes d'interaction homme machine. La plupart des applications actuelles exploitent essentiellement des langues internationales telles que l'anglais, le français, l'espagnol, etc. Pour plus d'intérêt et de plus value, il est important que ces applications intègrent les différentes langues des utilisateurs afin que celles-ci parlent à leur *"cœur"*. Ce besoin est vital pour les langues africaines et spécifiquement pour les langues camerounaises dont l'usage a beaucoup régressé au lendemain de la colonisation, au profit des langues étrangères adoptées dans les différents pays africains comme langues officielles. Le cas de la Côte d'Ivoire illustre bien cette situation où Open G, un pionnier de la tech, a créé le premier smartphone à commandes entièrement vocales, avec un accès dans seize langues locales (*Dioula, Sénoufo, Bété...*)[1] pour les utilisateurs qui ne savent ni lire ni écrire. Les langues camerounaises auxquelles nous accordons de l'intérêt dans ce travail sont très différentes des langues internationales manipulées dans les systèmes de reconnaissance vocale dans la mesure où pour la plupart, ce sont des langues à tons (langues pour lesquelles l'intonation donnée aux syllabes a de l'impact sur la signification du mot).

Les applications de reconnaissance vocale sont souvent déployées sur des systèmes et elles doivent réagir sous la contrainte de temps réel, en d'autres termes leur temps de réponse est borné pour que le résultat soit utile et pertinent pour l'utilisateur. La satisfaction de cette contrainte de temps de réponse réduit, passe souvent par différentes stratégies à savoir : le choix d'une architecture pour une mise en œuvre efficace en termes de vitesse de traitement, la paral-

1. https ://information.tv5monde.com/video/cote-d-ivoire-le-smartphone-qui-maitrise-16-langues-locales

lélisation des modules de l'application, etc.

Dans le cadre de ce travail, nous adressons une approche hybride qui consiste en la mise en œuvre d'une version parallèle d'une application sur une architecture efficace en temps de traitement : les architectures reconfigurables. La mise en œuvre d'applications parallèles sur ce type d'architecture présente de nombreuses problématiques notamment : l'utilisation de langages HDL (Hardware Description Language) de bas niveaux, la mise en œuvre de l'infrastructure de communication entre tâches parallèles, la synchronisation, etc. La prise en compte de toutes ces problématiques pour mettre en œuvre une application engendre un défi de productivité pour les ingénieurs dans un domaine qui est très compétitif en termes de temps de mise sur le marché des produits (time to market). Ainsi, dans le but de développer un système de reconnaissance vocale des langues à tons temps réel sur système embarqué, nous proposons une approche de mise en œuvre productive d'applications parallèles sur architectures reconfigurables.

1.2 Positionnement du travail

1.2.1 Objectifs de la thèse

L'objectif de cette thèse est de proposer une approche de mise en œuvre productive d'applications parallèles sur système embarqué avec les architectures reconfigurables. L'idée est de libérer le développeur au maximum de toutes les problématiques liées à la mise en œuvre des systèmes parallèles sur architecture reconfigurable ; et lui permettre de se concentrer uniquement sur l'application à paralléliser. Cet objectif peut être décliné en sous objectifs à savoir :

- Offrir au développeur un outil qui abstrait toutes les problématiques liées à la mise en œuvre d'un système parallèle (interconnexion, communication, synchronisation, etc.) ;
- Proposer une approche qui permet au développeur de mettre en œuvre les tâches parallèles en langages de haut niveau tel que le C et C++ ;
- Proposer à l'ingénieur une approche qui aboutit à un système performant en temps de traitement ;
- Concevoir et mettre en œuvre une infrastructure de communication qui permet la communication du système parallèle obtenu avec d'autres composants ou systèmes et qui rend notre approche compatible à la co-conception et la co-synthèse matérielle logicielle sur architecture reconfigurable.

Tous ces objectifs sont explorés dans le cadre du développement d'une application de reconnaissance des langues à tons sur système embarqué. L'intérêt étant de pouvoir satisfaire la contrainte de temps réel par la parallélisation.

1.2.2 Contributions

L'atteinte des objectifs de cette thèse nous a permis d'aboutir à trois principales contributions.

La première contribution est la proposition d'une méthodologie de conception productive d'applications parallèles sur architecture reconfigurable ; basée sur la combinaison des approches de synthèse de haut niveau et de FPGA Overlay. Pour être plus précis, nous avons proposé une approche de FPGA Overlay basée sur la synthèse de haut niveau pour la mise en œuvre d'applications parallèles à mémoire distribuée communicant grâce au paradigme MPI-2 RMA. L'approche FPGA Overlay permet d'abstraire les différentes problématiques liées à la mise en œuvre des systèmes parallèles. L'approche synthèse de haut niveau offre au développeur la possibilité de mettre en œuvre les tâches de traitement parallèles à partir des langages de haut niveau utilisés en environnement logiciel tels que le C, le C++, etc.

La seconde contribution est la conception et la mise en œuvre d'un modèle de communication MPI-2 RMA entre plusieurs clusters hétérogènes. En effet, afin de permettre au système parallèle obtenu à l'issue de l'exécution de notre approche, de communiquer avec d'autres systèmes ; et afin de rendre notre approche compatible à la co-conception et co-synthèse matérielle logicielle, nous avons conçu et mis en œuvre un modèle de communication entre clusters hétérogènes qui s'inspire d'un modèle existant et qui est adapté au paradigme MPI-2 RMA.

La troisième contribution est la mise en œuvre de l'accélérateur matériel d'un algorithme très précis d'extraction des fréquences fondamentales (F_0) utilisées comme principales caractéristiques pour la reconnaissance des langues à tons. En effet, dans le but de mettre en œuvre un système de reconnaissance des langues camerounaises sur système embarqué, nous nous sommes focalisés dans un premier temps sur la composante d'extraction des caractéristiques. La recherche de la satisfaction de la contrainte de temps réel par la parallélisation nous a permis d'aboutir à un accélérateur matériel implémenté sur FPGA qui peut être réutilisé pour toutes les applications qui nécessitent une fonction d'extraction des F_0 sur le signal vocal.

1.3 Structure du document

La suite de ce manuscrit est organisée en 4 chapitres et une conclusion générale. Nous précisons que la dernière section de chaque chapitre est consacrée au résumé de celui-ci.

Chapitre 2 : Intégration d'une application de reconnaissance des langues à tons dans un système embarqué

Le chapitre 2 caractérise les langues camerounaises. Il introduit les concepts clés sur les systèmes de reconnaissance vocale et les systèmes embarqués. Le contexte de cette thèse est justifié par les intérêts de la mise en œuvre des systèmes de reconnaissance vocale temps réel des langues camerounaises sur système embarqué. Pour la satisfaction de la contrainte de temps réel, la parallélisation sur une architecture performante en temps de traitement est choisie comme stratégie. La délimitation d'une mise en œuvre de cette application est faite et elle est réduite à la composante d'extraction des caractéristiques.

Chapitre 3 : Conception d'un système embarqué parallèle sur architecture reconfigurable

Le chapitre 3 rappelle les concepts clés sur les systèmes parallèles. Après la présentation d'un rapport d'implémentation sur environnement logiciel, les notions sur les architectures reconfigurables sont introduites. La compréhension de ces concepts permet de comprendre la problématique posée sur la conception d'applications parallèles sur ce type d'architecture. Un état de l'art sur cette problématique est fait, et le chapitre est bouclé par la description d'une idée d'approche pour résoudre le problème posé.

Chapitre 4 : Proposition d'une approche de facilitation de la co-conception d'applications parallèles MPI sur MP-RSoC

Le chapitre 4 développe la méthodologie proposée sur mise en œuvre productive d'applications parallèles MPI-2 RMA sur architectures reconfigurables. Le flot de conception obtenu est déroulé pour la mise en œuvre de l'extraction des caractéristiques pour la reconnaissance des langues à tons. Ce chapitre est clôturé par une discussion sur la méthodologie proposée.

Chapitre 5 : Conception de XMATIP : une plateforme pour la co-conception des applications parallèles MPI-2 RMA

Le chapitre 5 présente la conception et la mise en œuvre sur architecture reconfigurable d'un modèle de communication MPI-2 RMA entre clusters hétérogènes sur l'architecture FPGA Overlay utilisée. Une discussion est faite sur ce modèle et ses différentes implications sur la méthodologie présentée plus haut.

Chapitre 6 : Conclusion

Le chapitre 6 vient conclure ces travaux et présente quelques perspectives ouvertes suite à la conception de notre approche.

Chapitre

2 Intégration d'une application de reconnaissance des langues à tons dans un système embarqué

Sommaire

2.1 Introduction

D'après Victor Cherbuliez, *"Parler sa langue maternelle, c'est avoir sa patrie sur les lèvres"* [1]. C'est dire combien la langue est importante pour un peuple. Selon Crystal[2], 82% des langues du monde ont moins de 100 000 locuteurs, et 56% moins de 10 000 locuteurs. Malheureusement une majorité de langues risquent de disparaître au profit d'autres langues dominantes[3]. C'est le cas des langues camerounaises qui d'après Bitjaa *" sont en danger "* [4], au profit du français et l'anglais, les deux langues les plus dominantes en Afrique subsaharienne. Pour des institutions comme l'UNESCO, le développement de ressources et d'outils numériques pour de telles langues est une étape nécessaire pour tenter de préserver la diversité linguistique menacée[5]. Parmi ces différents outils, la reconnaissance automatique de la parole joue un rôle très important, car elle est devenue un outil important de l'interface homme machine[6]. En outre, cet outil est de plus en plus utilisé dans divers domaines tels que la médecine de réadaptation, la traduction, le transport intelligent, les jouets intelligents, les robots intelligents et les appareils domestiques intelligents[7]. Du fait des contraintes ergonomiques et de restriction d'espace, ces applications sont généralement déployées sur des systèmes embarqués. Dans ce chapitre, après avoir décrit les langues camerounaises à travers leurs situations géographiques et leurs caractéristiques, nous présentons les intérêts et motivations de concevoir des systèmes de reconnaissances des langues camerounaises sur des systèmes embarqués.

2.2 Caractéristiques des langues camerounaises

2.2.1 Présentation du Cameroun

Le Cameroun, de capitale Yaoundé, est un État d'Afrique centrale, situé entre le Nigeria au nord-ouest, le Tchad au nord-est, la République centrafricaine à l'est, la république du Congo au sud-est, le Gabon au sud, la Guinée équatoriale au sud-ouest et le golfe de Guinée au sud-ouest (voir figure 2.1). Il s'étire vers le nord jusqu'au lac Tchad, formant un triangle de 475 442 km^2 de superficie reliant l'Afrique équatoriale à l'Afrique occidentale. Il compte dix régions administratives qui sont les suivantes (avec leur chef-lieu) : au nord, l'Extrême-Nord (Maroua), le Nord (Garoua), l'Adamaoua (Ngaoundéré) ; à l'ouest, le Nord-Ouest (Bamenda), le Sud-Ouest (Buea), l'Ouest (Bafoussam) et le Littoral (Douala) ; au sud, le Centre (Yaoundé), l'Est (Bertoua) et le Sud (Ebolowa).

2.2.2 Les aires linguistiques au Cameroun

Depuis le début des années 1960, un consensus assez général s'est établi autour de la classification des langues africaines proposée par le linguiste américain Joseph Greenberg. Celui-ci regroupe l'ensemble des langues africaines dans quatre grands ensembles (ou « phylums »)[8] qui sont : (1) le *phylum Niger-Kordofanien/Niger-Congo*, qui regroupe la grande majorité des langues de l'Afrique occidentale avec la grande famille Bantou qui couvre pratiquement toute l'Afrique au sud de l'équateur ; (2) le *phylum nilo-saharien*, qui regroupe des langues parlées essentiellement en Afrique centrale, orientale et nord-orientale ; (3) le *phylum afro-asiatique*, auquel appartiennent les langues sémitiques (dont l'arabe), l'ancien égyptien et le berbère, et un grand nombre de langues du Nigeria et du Cameroun (famille tchadique) ainsi que d'Éthiopie et des régions avoisinantes (familles couchitique et omotique) ; (4) le *phylum khoisan*, en régression, regroupant des langues essentiellement parlées en Afrique australe (Afrique du Sud, Namibie et

FIGURE 2.1: *Carte du Cameroun ; **Source** : CIA World Factbook ; **Auteur** : United States Central Intelligence Agency (Décembre 2002)*

Botswana) mais dont la zone d'extension allait autrefois beaucoup plus au nord.

Le Cameroun est linguistiquement diversifié, avec des langues appartenant à trois grandes familles : l'*afro-asiatique*, le *nilo-saharien* et le *Niger-Congo*. Au sein du Niger-Congo, trois groupes sont représentés : *Adamawa-Ubanguien (en anglais Adamawa-Ubangian)*, *Ouest-Atlantique (en anglais West-Atlantic)* et *Bénoué-congo (en anglais Benue-Congo)* qui comprend les groupes *Grassfields* et *Bantous*. Des recherches sociolinguistiques sur la détermination des unités-langues au Cameroun ont permis de recenser 284 langues locales parfaitement distinctes[4], utilisées ou ayant été utilisées il y a quelques années, par des communautés de locuteurs natifs localisables sur le territoire camerounais, pour une population de 24 348 251 personnes (estimation 2019 [1]). La figure 2.2[9] montre la localisation des principaux groupes.

2.2.3 Langues camerounaises : langues à tons

Parmis les langues utilisées au Cameroun, certaines sont des langues à tons. En fonction des auteurs, la notion de *ton* a plusieurs définitions qui se rencontrent ou se complètent. D'après Germain Ndi[10], en linguistique, on appelle ton : *"la hauteur relative et distinctive de la voix au cours de l'articulation d'une syllabe"*. Prosper Abega[11] définit le ton comme : *"la hauteur relative avec laquelle une syllabe est prononcée et qui contribue à la saisie du sens du mot. Ainsi des homographes ne seront discernés que par cette hauteur relative..."*. Ces deux définitions mettent en exergue le fait qu'en fonction du ton utilisé dans une syllabe, à l'intérieur d'un mot, ce dernier aura un sens différent.

Une langue à tons peut être définie comme *"une langue ayant une hauteur lexicalement significative, contrastive, mais relative, sur chaque syllabe."*[12]. En d'autres termes c'est une langue dont le sens des mots varie en fonction des tons que portent les syllabes. Ainsi les langues de l'Afrique subsaharienne, spécifiquement les langues camerounaises sont classées pour la plupart comme langues à tons[13].

Cette caractéristique des langues camerounaises constitue un enjeu important pour un ensemble de tâches de traitement automatique de langue notamment la reconnaissance vocale, et la compréhension de langue ; avec un grand risque de confusion de classe lexicale ou de sens entre

1. Estimation faite par le *Bureau Central des Recensement et des Etudes de Population (BUCREP)*

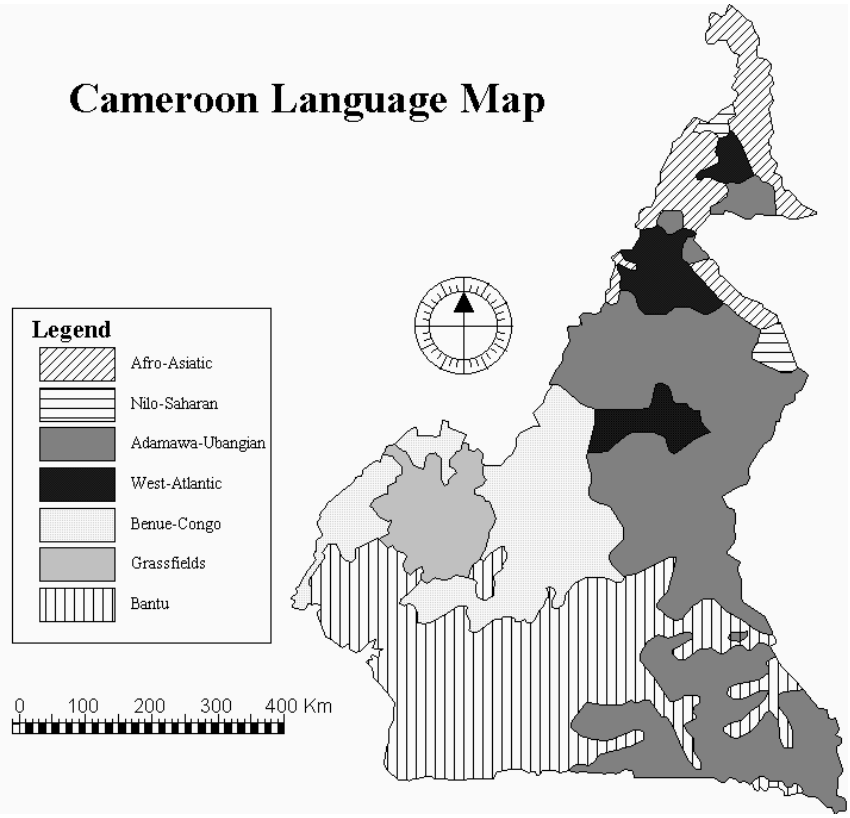

Figure 2.2: *Carte des langues du Cameroun montrant les principales familles [9]*

les différentes paires minimales. Cet enjeu est d'autant plus important dans une société numérique où les langues doivent être informatisées afin d'être intégrées dans les outils numériques du quotidien.

2.2.4 Langues camerounaises : langues peu dotées

V. Berment a proposé dans sa thèse[14], un protocole pour évaluer de manière quantitative le degré d'informatisation d'une langue. Cette évaluation se fait à travers un ensemble de *groupes de services* ou *ressources* qui sont évalués et notés, par exemple la capacité de faire de la reconnaissance automatique de caractères d'une langue, et qui permettent de calculer un indice appelé *indice-σ*.

À partir de l'*indice-σ*, V. Berment a défini 3 niveaux d'informatisation pour classer les langues en 3 groupes :

— les *langues-π* ont une moyenne entre 0 et 9,99 (*langues peu dotées*) ;
— les *langues-μ* ont une moyenne entre 10 et 13,99 (*langues moyennement dotées*) ;
— les *langues-τ* ont une moyenne entre 14 et 20 (*langues très bien dotées*).

Une *langue peu dotée* est ainsi définie comme une langue dont l'indice-σ, n'atteint pas 10/20.

Dans le contexte de la reconnaissance vocale, LE Viet Bac dans sa thèse[15], redéfinit une langue peu dotée comme *"une langue qui ne possède pas encore ou pas beaucoup (en quantité et en qualité) de ressources linguistiques pour la construction d'un système de reconnaissance automatique de la parole, particulièrement dans un contexte d'apprentissage statistique où les données doivent être disponibles en grande quantité."*.

Plusieurs langues africaines et spécifiquement les langues camerounaises sont peu dotées, à tel point que Bitjaa[4] conclut que :*"Toutes les langues camerounaises sont en danger, cependant*

certaines le sont plus que d'autres ". Il est donc important de construire des outils pour accroître leur niveau d'informatisation ; la conception des systèmes de reconnaissance vocale étant l'un des services importants d'après Berment et al.[14].

2.3 Les systèmes de reconnaissance vocale

2.3.1 Définition

La reconnaissance automatique de la parole (en anglais *Automatic speech Recognition*, ASR) permet d'analyser la voix humaine captée au moyen d'un microphone pour la transcrire sous la forme d'un texte exploitable par une machine.

En fonction du contexte, cette tâche peut s'avérer très complexe, car elle doit prendre en compte des contraintes non seulement liées à l'environnement, mais aussi celles intrinsèques à la langue manipulée.

2.3.2 Classification des systèmes de reconnaissance de la parole

Il existe deux principales typologies de systèmes de reconnaissances de la parole : la typologie basée sur le type d'élocution et la typologie basée sur le modèle du locuteur[16].

La typologie basée sur le type d'élocution

Dans cette catégorie, on distingue quatre (04) principaux types de systèmes de reconnaissance : les systèmes à mots isolés, les systèmes à mots connectés, les systèmes à élocution continue et les systèmes à élocution spontanée[16] :

- **Les systèmes à mots isolés** : Ce sont des systèmes qui sont faits pour reconnaître une seule élocution (un seul mot ou expression) à la fois. Ces systèmes sont appropriés pour des situations où le locuteur nécessite un seul mot ou expression pour donner une réponse ou déclencher une commande, mais cela présente des problèmes lorsque beaucoup de mots ou expressions sont prononcés à la fois. Il est très aisé et facile d'implémenter ce type de système, parce que les bornes ou dimensions des mots sont connues et les mots isolés ont souvent tendance à être bien prononcés, ce qui constitue un grand avantage pour ce type de système. Toutefois, tout changement de ces bornes à un moment donné peut affecter le système [17].
- **Les systèmes à mots connectés** : Ils sont similaires aux systèmes à mots isolés, à la seule différence qu'il permet à deux élocutions séparées d'être articulées en même temps avec une pause minimale entre elles. L'élocution étant ici la prononciation d'un mot ou groupe de mots renvoyant à une seule entité et une seule signification pour l'ordinateur.
- **Les systèmes à élocution continue :** Ils permettent au locuteur de s'exprimer naturellement, pendant ce temps le système identifie le contenu de son allocution. Dans ce cas les mots sont prononcés ensemble sans pause ni tout autre moyen de séparation entre les mots. Les systèmes continus sont généralement difficiles à implémenter.
- **Les systèmes à élocution spontanée :** Ces systèmes reconnaissent l'expression en langage naturel. L'élocution spontanée est la parole naturelle qui vient soudainement de la bouche d'un individu. Un système de reconnaissance à élocution spontanée est capable de prendre en compte toutes les caractéristiques du langage naturel à l'instar d'un ensemble de mots articulés simultanément.

La typologie basée sur le modèle du locuteur

Chaque locuteur a une voix singulière liée à sa physionomie et sa personnalité. Les systèmes de reconnaissance de la parole du point de vue du locuteur sont classés suivant les catégories

ci-après[16] :

- **Les systèmes à modèle de locuteur dépendant** : Ce sont des systèmes qui sont construits pour un type spécifique de locuteur. Ils sont généralement performants pour une catégorie particulière de locuteurs et moins efficace pour d'autres types de locuteurs. Ces systèmes sont très souvent peu coûteux, faciles à mettre en œuvre et très performants. Mais ne sont pas flexibles comme les systèmes à modèle de locuteur indépendant.
- **Les systèmes à modèle de locuteur indépendant** : Ce type de système peut reconnaître toute variété de locuteur sans nécessiter une préparation spécialisée de sa part. Les systèmes à modèle de locuteur indépendant sont conçus pour opérer sur n'importe quel locuteur. Ils sont utilisés dans des systèmes de réponses vocales interactifs (IVRS : Interactive Voice Response System) qui peuvent prendre en entrée un grand nombre d'utilisateurs. L'inconvénient ici est que la taille du vocabulaire est souvent limitée. La mise en œuvre de ce type de système est la plus difficile de toutes.
- **Les systèmes à modèle de locuteur adaptatif** : Ce sont des systèmes qui utilisent les données des systèmes à modèle de locuteur dépendant et les adaptent pour un locuteur approprié afin de reconnaître la parole et réduire le taux d'erreur par adaptation. Dans ces systèmes, l'opération d'adaptation se fait en prenant en compte les caractéristiques liées au locuteur.

De tout ce qui précède, nous comprenons que l'importance accordée à une typologie de système de reconnaissance vocale dépend de l'objet d'étude qui nous intéresse dans la parole (l'élocution ou le locuteur), ainsi que les contraintes liées à ce dernier. Une fois l'objet choisi, il est possible de passer à la phase de mise en œuvre du dit système. La section suivante présente le principe de fonctionnement d'un système de reconnaissance automatique de la parole.

2.3.3 Approches de conception des systèmes de reconnaissance de la parole

Dans le principe de base d'un système de reconnaissance automatique de la parole, la personne qui parle émet des variations de pression dans son larynx. Les sons produits sont numérisés par le microphone et transmis par un support ou un réseau. Les sons numérisés sont transformés en unités acoustiques (ou vecteurs acoustiques) via un modèle acoustique. Par la suite, le moteur de reconnaissance analyse cette séquence de vecteurs acoustiques en la comparant à celles qu'il a en mémoire (son modèle de langage) et propose la séquence candidate la plus probable. Il est donc nécessaire que la séquence de vecteurs acoustiques soit proche d'une des séquences mémorisées par le moteur de reconnaissance. Le cœur d'un système reconnaissance vocale peut être vu comme un modèle mathématique qui peut générer un texte correspondant aux morceaux de parole reconnus[18]. Au fil du temps, de nombreuses approches ont été proposées pour la conception des systèmes de reconnaissance vocale. Plusieurs travaux ont fait un état de l'art de ces différentes méthodes[19, 20]. La plupart des approches actuelles s'appuient sur de l'apprentissage statistique, et il est nécessaire d'effectuer des prétraitements pour extraire les propriétés acoustiques du signal.

L'apprentissage machine ou apprentissage statistique est une technique d'intelligence artificielle qui implique d'apprendre à un ordinateur à reconnaître des modèles, contrairement à l'approche traditionnelle, qui consiste à programmer un ordinateur avec des règles spécifiques. L'enseignement se fait par le biais d'un processus d'apprentissage qui consiste à fournir de grandes quantités de données audio à l'algorithme et à lui permettre d'apprendre des données et de détecter des modèles qui peuvent ensuite être utilisés pour réaliser certaines tâches [21]. Les techniques d'apprentissage artificielle les plus couramment utilisées pour la reconnaissance vocale sont le modèle de Markov caché en anglais hidden Markov model (HMM), les Réseaux neuronaux artificiels (en anglais Artificial neural networks/ANN) et les machines à vecteurs de support (en anglais Support-Vector Machine, SVM).

Les modèles de Markov cachés

Les HMM supposent que le signal vocal donné peut être caractérisé comme un processus aléatoire paramétrique, et donc que ses paramètres peuvent être déterminés d'une manière bien définie et précise. Cet algorithme est une extension de la chaîne de Markov, qui peut produire des symboles de sortie indépendamment de l'état dans lequel ils se trouvent[22]. En conséquence, la sortie du HMM est une fonction probabiliste de l'état, et pour la séquence d'entrée, la séquence d'état n'est pas observable, d'où l'utilisation du mot caché dans le nom de l'algorithme. Les HMM ont été l'une des techniques de classification les plus performantes en termes de reconnaissance vocale. Pour cette raison, c'est également l'une des techniques les plus utilisées[23]. Elle est très flexible et peut facilement s'adapter à la structure requise.

Les Réseaux Neuronaux Artificiels

Les Réseaux Neuronaux Artificiels (Artificial neural networks, ANN) sont d'excellents classifieurs pour les problèmes de reconnaissance des formes. Ils sont utilisés pour leur capacité à apprendre et à s'organiser en fonction de l'ensemble de données fourni lors de la phase de d'entraînement. Ils fonctionnent exceptionnellement bien avec des données inconnues. L'inconvénient de l'utilisation d'un ANN est qu'ils ont tendance à surentraîner et à être confrontés au problème des minima locaux. Ils ignorent également la variabilité temporelle présente dans le signal vocal ; ce problème peut être résolu en utilisant des modèles hybrides HMM-ANN. Le modèle hybride est utilisé pour obtenir les avantages des deux modèles[24].

Il existe plusieurs variantes des ANN, Mishaim Malik et al.[25] ont mis en exergue les plus utilisées pour la reconnaissance vocale. Le premier modèle appelé perceptron multicouche (Multilayer perceptrons, MLP) est un réseau neuronal simple de type feed-forward contenant au moins trois couches : entrée, sortie et une cachée. L'algorithme appliqué pendant la phase de d'entraînement est basé sur l'approche de rétro propagation. La sortie générée est basée sur le neurone de sortie ayant l'activation la plus élevée. L'un des principaux inconvénients de ce modèle est qu'il ne peut prendre que des entrées de longueur fixe, ce qui le rend incapable de gérer la dynamique du signal vocal d'entrée. Un autre problème est que cet algorithme ne peut traiter efficacement que de petits vocabulaires, ce qui en fait un bon reconnaisseur de phonèmes mais pas un reconnaisseur de mots efficace[26]. Des travaux montrent qu'il s'agit d'un modèle de classification efficace pour la reconnaissance vocale[27].

Les cartes auto-organisatrices (Self-organizing maps, SOM) ont été introduites en 1982 par Teuvo Kalevi Kohonen[28]. L'idée principale des SOM est que les signaux d'entrée sont placés de manière à produire une carte de contour d'un espace d'entrée de dimension supérieure à un espace de caractéristiques de dimension inférieure. Ainsi, le signal d'entrée est d'abord placé de manière aléatoire dans l'espace des caractéristiques d'entrée, qui est ensuite organisé en différents clusters. Chacun des clusters formés représente une caractéristique unique des signaux d'entrée. Une carte auto-organisatrice est un type de réseau de neurones artificiels constitué d'un ensemble de neurones arrangés suivant une grille à faible dimension. A chaque neurone est associé un vecteur prototype appartenant à l'espace des observations. La classification des données peut se faire en supposant que les observations affectées au même neurone appartiennent à la même classe. Dans ce cas, le nombre de neurones doit correspondre au nombre de classes et deux neurones voisins sur la carte représenteront des classes voisines dans l'espace des observations. Il est possible aussi de projeter les données sur une grande carte, puis de classifier les neurones de la carte pour achever la classification finale des données. Ce modèle est également utilisé pour la reconnaissance vocale[29].

Un autre modèle, le réseau à fonctions de base radiale (Radial basis functions, RBF) est un type de réseau neuronal artificiel supervisé qui utilise l'apprentissage automatique supervisé pour fonctionner comme un classifieur non linéaire. Il utilise des fonctions de base radiales comme fonctions d'activation. Comme d'autres types de réseaux de neurones, les réseaux de fonctions

de base radiales ont des couches d'entrée, des couches cachées et des couches de sortie. Multiples travaux ont utilisé ce modèle pour la reconnaissance vocale[30, 31].

L'autre variante de réseau utilisé est le réseau de neurones récurrents (Recurrent neural network, RNN)[32]. C'est un réseau de neurones artificiels présentant des connexions récurrentes. Un réseau de neurones récurrents est constitué d'unités (neurones) interconnectées interagissant non-linéairement et pour lequel il existe au moins un cycle dans la structure. Des travaux exploitent le modèle RNN pour la reconnaissance vocale[33].

Une autre variante d'architecture utilisée est le réseau de neurones convolutif (Convolutional Neural Network, CNN) qui est une architecture de réseau de neurones profond qui permet de détecter la présence de formes en appliquant un calcul local, appelé convolution. Ce calcul de convolution s'applique à différentes échelles et permet d'identifier progressivement le contenu d'une image ou d'un signal par association et recoupement. En raison de sa bonne capacité de génération de caractéristiques et de discrimination ce modèle est utilisé dans de nombreux travaux pour la reconnaissance automatique de la parole[34].

Les réseaux neuronaux flous (Fuzzy neural networks, FNN) sont une technique hybride qui intègre les concepts d'un système flou dans les réseaux neuronaux. En raison de l'utilisation de systèmes flous, une fonction d'appartenance est utilisée pour s'assurer que chaque élément est cartographié à un degré d'appartenance approprié. Cette fonction d'appartenance s'avère très utile pour cartographier les signaux vocaux, car ils n'ont pas de limites claires[35]. Un autre avantage de l'utilisation du FNN est qu'un ANN nécessite une grande quantité de données pour être entraîné efficacement. Mais les FNN donnent de meilleurs résultats, même avec de petits ensembles de données, car ils convergent pendant la phase d'apprentissage[36]. Des travaux prouvent l'efficacité des réseaux neuronaux flous[37].

Les machines à vecteurs de support

L'objectif de l'algorithme de la machine à vecteurs de support ou séparateur à vaste marge (en anglais support-vector machine, SVM) est de trouver un hyperplan dans un espace à N dimensions (N - le nombre de caractéristiques) qui classe distinctement les points de données. Pour séparer les deux classes de points de données, de nombreux hyperplans peuvent être choisis. L'objectif est de trouver un plan qui présente la marge maximale, c'est-à-dire la distance maximale entre les points de données des deux classes. La maximisation de la distance de la marge fournit un certain renforcement, de sorte que les futurs points de données peuvent être classés avec plus de confiance. Le SVM a été adopté pour effectuer la tâche de reconnaissance de la parole. Le SVM peut être implémenté indépendamment ou comme un modèle hybride avec les HMM[38]. Plusieurs travaux mettent en exergue l'efficacité de ce modèle pour la reconnaissance vocale[39].

2.3.4 Architecture

En général, tous les systèmes de reconnaissance automatique de la parole ont la même architecture, que le vocabulaire soit limité, moyen, large ou très large. Cette architecture peut être modifiée ou complétée en fonction de la reconnaissance à effectuer. Un système de reconnaissance automatique de la parole est généralement composée de cinq éléments typiques : Extraction de caractéristiques ; Modèle acoustique ; Modèle de langue ; Modèle de prononciation ; et Décodeur. Dans l'architecture illustrée à la figure 2.3, le signal vocal est reçu, puis les caractéristiques sont extraites. Les paramètres obtenus sont transmis au décodeur, qui utilise les modèles de langue, acoustique et de prononciation pour l'apprentissage.

Dans la première étape, après un prétraitement du signal, celui-ci est découpé selon un fenêtrage temporel qui souvent prend des fenêtres d'entre 20 et 30 ms de longueur[40] ; les fenêtres ainsi obtenues sont appelées trames. Il existe plusieurs techniques utilisées pour traiter ces trames et en extraire les caractéristiques, on peut distinguer : la méthode LPC (Linear Predictive Coding)[41], RASTA (RelAtive SpecTral)[42], LDA (Linear Discriminant Analysis)[43], les MFCC

(Mel-frequency cepstrum coefficient)[44], la méthode PLP (Perceptual Linear Prediction)[45], etc.

Dans l'étape de reconnaissance des modèles acoustiques, ce sont des modèles mathématiques et ou informatiques permettant d'établir une correspondance entre le signal reçu en entrée et un mot du vocabulaire pris en compte. Le modèle acoustique a pour rôle d'établir une correspondance entre l'information acoustique reçue en entrée et les sous unités de mots (phonèmes, syllabes, tons, etc.) prévus par le modèle. Les approches présentées dans la section 2.3.3 interviennent dans ce composant.

Les modèles du dictionnaire de prononciation décrivent la façon dont un mot est prononcé et représenté. Pour un petit vocabulaire, les modèles basés sur les mots sont utilisés pour définir des mots entiers comme des unités sonores individuelles. Le dictionnaire de prononciation fait partie du modèle de prononciation. La traduction du signal vocal en texte est réalisée en classant le signal vocal en petites unités sonores. Le modèle de prononciation détermine ensuite comment ces petites unités peuvent être combinées pour former des mots valides.

Dans les modèles de langues, des règles sont introduites pour le respect des contraintes linguistiques présentes dans la langue. Le modèle de la langue est utilisé pour prédire la séquence de mots la plus probable pour un texte donné.

Un décodeur est vu comme un algorithme qui combine des connaissances acoustiques et linguistiques pour transcrire automatiquement l'enregistrement d'entrée[46]. Le but du décodage est de déduire la séquence d'états qui a généré les observations données. A partir de cette séquence d'états, il est facile de trouver la séquence des sous unités (phonèmes, tons, mots, etc.) la plus probable qui correspond aux paramètres observés.

Un système de reconnaissance automatique de la parole nécessite une phase d'entraînement et une phase de test. Dans la phase d'entraînement, les modèles acoustiques et modèles de la langue sont générés. Ils seront ensuite utilisés dans la phase de test pour les valider, puis plus généralement une fois validés pour reconnaître la parole.

La classification ou inférence est le processus par lequel, une donnée de test inconnue va être associée à un son d'une des classes de référence du modèle. À cet effet, une base de données acoustique est nécessaire pour entraîner le système. Ratnadeep dans [17], présente une revue sur les bases de données utilisées pour la reconnaissance vocale. La mise en œuvre de ce système nécessite une bonne connaissance des techniques d'extraction des caractéristiques et des algorithmes utilisés pour la classification.

2.3.5 Le facteur temps réel

Le facteur temps réel (*Real-Time Factor, RTF*) est la mesure la plus couramment utilisée pour calculer la vitesse d'un modèle de reconnaissance vocale proposé. Le RTF peut être calculé à l'aide de la formule suivante :

$$RTF = \frac{P}{I}$$

Où P est le temps mis par le système pour traiter l'entrée et I est la durée de l'audio d'entrée. Si le RTF est égal à 1, cela signifie que l'entrée audio a été traitée en *"temps réel"*. RTF est une valeur fortement dépendante du matériel et ne se limite pas au calcul de la vitesse d'un modèle de reconnaissance vocale.

2.3.6 État des lieux sur la reconnaissance des langues à tons

Dans l'état de l'art il existe plusieurs travaux sur la reconnaissance vocale des langues à tons. Ces travaux sont répartis sur des langues de plusieurs régions du monde : Afrique, Amérique, Asie, Asie du Sud, etc. Jaspreet Kaur et al. présentent dans [47], un état de l'art des travaux sur

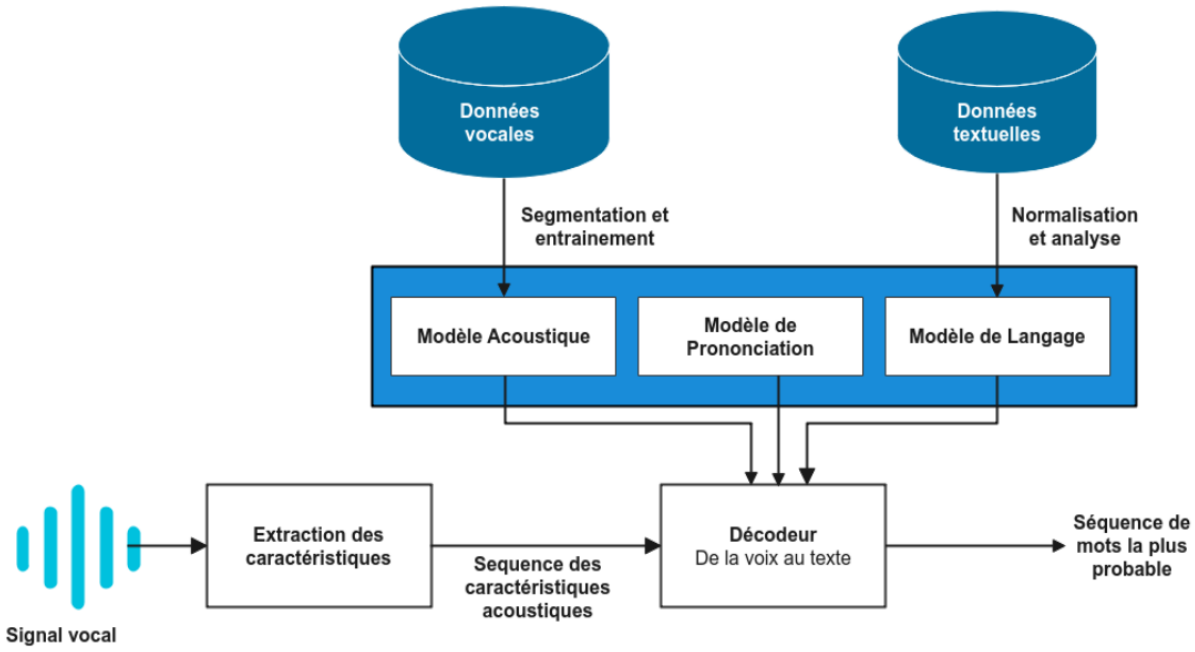

Figure 2.3: *Architecture générale d'un système de reconnaissance vocale.*

la reconnaissance vocale des langues à tons.

De façon spécifiques, pour les langues africaines et camerounaises, quelques travaux ont été réalisés.

Dans[48], Sikasote et al. présentent un corpus appelé *BembaSpeech* pour la reconnaissance vocale de la langue Bemba. Ils utilisent ce corpus pour construire un système de reconnaissance Bemba de bout en bout à partir du système libre DeepSpeeh performant sur le jeu de données de 17,5 heures de discours BembaSpeech.

Doumbouya et al.[49] ont conçu un assistant virtuel simple, mais fonctionnel, capable de gérer des contacts pour des locuteurs analphabètes des langues Maninka, Pular et Susu. Il ont collecté l'ensemble de données nécessaire au développement du module de reconnaissance vocale. Afin de relever le défi des ressources limitées, ils ont exploré l' apprentissage par représentation non supervisé (*Unsupervised Representation Learning*) de la parole dans deux contextes : premièrement, lorsque les représentations sont apprises à partir de langues à fortes ressources et transférées à des langues à faibles ressources non liées. Deuxièmement, lorsque les représentations sont apprises à partir de données de faible qualité, des archives radio abondantes dans de nombreuses langues à faibles ressources, dans les langues cibles à faibles ressources. Enfin, ils ont développé une méthode d'analyse qualitative efficace des représentations apprises.

Dans [50], Gelas et al. présentent les stratégies choisies pour développer un corpus de texte, un dictionnaire de prononciation et un corpus de parole pour le Swahili, une langue bantoue peu dotée parlée dans une large zone de l'Afrique de l'Est. Ils explorent des méthodologies telles que le crowdsourcing ou le processus de transcription coopératif. En outre, ils tirent parti de certaines caractéristiques linguistiques de la langue, telles que la richesse de la morphologie ou le vocabulaire partagé avec l'anglais, pour améliorer les performances de leur système de reconnaissance automatique de la parole en swahili dans la tâche de transcription de la parole diffusée.

Les travaux de [51] proposent des systèmes de reconnaissance vocale pour deux langues à faibles ressources : le Fon et le Igbo. Ils montrent que l'utilisation de réseaux neuronaux profonds de bout en bout (*End-to-End Deep Neural Networks / E2E DNN*) avec la classification

temporelle connexionniste (*Connectionist Temporal Classification / CTC*)[32], permet d'obtenir des résultats prometteurs sans utiliser de modèles de langage, qui nécessitent généralement d'énormes quantités de données pour l'apprentissage.

Odelobi[52] s'est intéressé à l'approche Réseau de Neurones Artificiels pour la reconnaissance des tons du Yoruba. Le modèle obtenu a atteint un taux de reconnaissance de 76% et a obtenu un taux de reconnaissance de 71% pour les tons élevés. Yusof et al. [53] ont passé en revue les travaux effectués sur la reconnaissance automatique de la parole en Yoruba et ont également étudié l'écart de performance entre la reconnaissance automatique de la parole humaine et la reconnaissance automatique de la parole en yoruba. Adetunmbi et al. [54] ont présenté un système de reconnaissance basé sur les syllabes pour le Yoruba.

Luka et al.[55] ont présenté un système de reconnaissance vocale Hausa basé sur un Réseau de Neurones Artificiels. Le réseau neuronal a été entraîné en conjonction d'une extraction de caractéristiques utilisant les MFCC. La taille de la base de données était de 320 mots, et les mots ont été prononcés par deux locuteurs masculins et deux locuteurs féminins. Schlippe et al. [56] ont discuté sur un système de reconnaissance vocale Hausa pour un large vocabulaire avec l'aide de Rapid Language Adaptation Toolkit (RLAT).

Yemmene et al.[57] mettent en lumière les motivations, les défis et les perspectives inhérents à la construction d'un système de reconnaissance automatique de la parole basé sur l'apprentissage profond pour la langue camerounaise *Ngiemboon*[2], minoritaire et sous-financée. Ce travail présente les questions essentielles pour mener des recherches sur le traitement de la parole dans cette langue.

Hamlaoui et al. dans [58] ont présenté le corpus *BULBasaa*, un corpus bilingue Bàsàá-français composé de parole contrôlée (élucidée) et non contrôlée (naturelle). Environ 50 heures de discours Bàsàá ont ainsi été collectées, puis soigneusement reprononcées et traduites oralement en français dans un environnement contrôlé par quelques locuteurs bilingues. Pour un sous-ensemble d'environ 10 heures du corpus, chaque énoncé a été en outre transcrit phonétiquement afin d'établir un standard pour la production des outils de traitement automatique du langage naturel.

Dans [59], Mboning et al. présentent le projet *NTeALan (New Technologies for African Languages)*[3]. Dans ce projet, il a été mis en place plusieurs applications : chatbots, dictionnaire collaboratif, carte linguistique, API pour les ressources linguistiques africaines. Certaines de ces ressources peuvent être utilisées pour l'entraînement des modèles de langage dans les systèmes de reconnaissance vocale des langues camerounaises et africaines. A ce jour, le projet prend en compte plus de 10 langues subsahariennes (avec le yemba, le bulu, le gomala, le duala, le bassa, le fefe, l'eton ... pour ce qui est des langues camerounaises) diversement équipées et traitées par les systèmes, y compris plusieurs outils de traitement automatique du langage naturel[60].

Un système de reconnaissance vocale de par ses différentes variantes et son architecture s'avère être un système complexe dans sa mise en œuvre. La conception des systèmes de reconnaissance des langues à tons n'échappe pas à cette réalité. C'est d'ailleurs ce qui explique le fait qu'une grande communauté de chercheurs se penche sur la question pour offrir aux ingénieurs les bases théoriques et les outils technologiques pour mettre en œuvre les systèmes de reconnaissance vocale pour les langues à tons, et spécifiquement certaines langues camerounaises en l'occurrence. Au vu de la nature des applications dans lesquelles ces systèmes sont utilisés, il est important d'analyser la principale catégorie de plateforme technologique sur laquelle sont déployées ces applications à savoir : les systèmes embarqués. La prochaine section est dédiée à ce concept.

2. Une langue bantoue du Grassfield parlée dans la région de l'Ouest du Cameroun (Afrique) par environ 400 000 personnes

3. Créé en 2017, NTeALan est une association qui œuvre pour la mise en place d'outils technologiques intelligents pour la promotion, le développement et l'enseignement des langues nationales africaines

2.4 Notions sur les systèmes embarqués

2.4.1 Définitions

Il existe plusieurs définitions de la notion de système embarqué, ici nous n'allons retenir que quelques unes.

D'après Qing Li et al. [61] : *"Les systèmes embarqués sont des systèmes informatiques avec une intégration matérielle et logicielle étroitement couplée, qui sont conçus pour exécuter une fonction dédiée."*

Et selon Peter Marwedel[62] : *"Les systèmes embarqués sont des systèmes de traitement de l'information qui sont intégrés dans un produit plus vaste et qui ne sont généralement pas directement visibles par l'utilisateur. "*

Un système embarqué est aussi un *système réactif* c'est-à-dire qu'il est en interaction continue avec son environnement et qui s'exécute à un rythme déterminé par cet environnement [63]. En outre, les systèmes embarqués ont une *fonctionnalité dédiée*. Cela signifie que le système a été conçu pour un objectif spécifique et des tâches prédéfinies.

Pour compléter son propos, Marwedel stipule que les systèmes embarqués sont des systèmes de traitement de l'information répondant à la plupart des caractéristiques que nous présentons dans le prochain paragraphe.

2.4.2 Les caractéristiques/attributs d'un système embarqué

Les caractéristiques communes des systèmes embarqués sont les suivantes :

L'efficacité

Les systèmes embarqués doivent être efficaces. Les indicateurs suivants peuvent être utilisés pour évaluer l'efficacité des systèmes embarqués :

- **L'énergie :** De nombreux systèmes embarqués sont des systèmes mobiles qui obtiennent leur énergie grâce à des batteries. Par conséquent, l'énergie électrique disponible doit être utilisée très efficacement.
- **Taille du code** : Tout le code à exécuter sur un système embarqué doit être stocké avec le système, et dans le cas des systèmes sur puce (SoC), ce stockage doit être réalisé sur une seule puce, la taille de la mémoire est bornée, notamment celle aux instructions, obligeant en fonction des systèmes une gestion efficace de cette mémoire.
- **Efficacité de l'exécution** : Une quantité minimale de ressources doit être utilisée pour mettre en œuvre la fonctionnalité requise.
- **Le poids** : Les systèmes embarqués sont souvent portables ou intégrés dans des objets nécessitant une optimisation du poids, ils se doivent d'être légers.
- **Le coût** : Pour les systèmes embarqués de grand volume, en particulier dans l'électronique grand public, la compétitivité sur le marché est un enjeu crucial, et une utilisation efficace des composants matériels et du budget de développement logiciel est nécessaire.
- **Le temps réel** : De nombreux systèmes embarqués doivent répondre à des contraintes de temps réel. Une contrainte temps réel est dite *dure* si son non-respect peut entraîner une catastrophe [64]. Toutes les autres contraintes temporelles sont dites *molles*. Dans le contexte des systèmes en temps réel, les arguments relatifs à la performance ou au délai moyen ne peuvent être acceptés. Une réponse garantie du système doit être expliquée sans arguments statistiques [64].

La sûreté de fonctionnement

La sûreté de fonctionnement (en anglais *dependability*) d'un système est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre. La

sûreté de fonctionnement est importante dans des systèmes, comme les voitures, les trains, les avions, etc. La sûreté englobe les aspects suivants d'un système :

- **Fiabilité** : La fiabilité est la probabilité qu'un système ne tombe pas en panne.
- **La maintenabilité** : La maintenabilité est la probabilité qu'un système défaillant puisse être réparé dans un certain laps de temps.
- **Disponibilité** : La disponibilité est la probabilité que le système soit disponible. La fiabilité et la maintenabilité doivent toutes deux être élevées pour obtenir une disponibilité élevée.
- **Sécurité** : Ce terme décrit la propriété selon laquelle un système défaillant ne causera aucun dommage.
- **La confidentialité** : Ce terme décrit la propriété selon laquelle les données confidentielles restent confidentielles et la communication authentique est garantie.

2.4.3 Architecture de mise en œuvre d'un système embarqué

Les systèmes embarqués sont structurés comme une collection de composants programmables entourés de circuits intégrés spécialisés (en anglais *Application Specific Integrated Circuits, ASIC*) et d'autres composants standards (*Application Specific Standard Parts, ASSP*) qui interagissent en permanence avec un environnement par le biais de capteurs et d'actionneurs. La collection peut être physiquement un ensemble de puces sur une plaque (une carte mère) on parle alors de système embarqué sur carte mère (en anglais *board based embedded systems*), ou un ensemble de modules sur un circuit intégré sous forme de système dans un boîtier (en anglais *System in a Package* ou *system-in-package, SiP*) constitué d'un certain nombre de circuits intégrés confinés dans un ou plusieurs boîtiers de support de puce qui peuvent être empilés en utilisant un boîtier sur un autre [4][65] ; ou encore sous forme de système embarqué sur puce (en anglais *system on a chip*) connu comme un système complet embarqué sur un seul circuit intégré (*« puce »*), pouvant comprendre de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interface, ou tout autre composant nécessaire à la réalisation de la fonction attendue.

Le logiciel est utilisé pour les fonctionnalités et la flexibilité, tandis que le matériel dédié est utilisé pour des performances accrues et une consommation d'énergie réduite. Les principaux composants programmables sont les microprocesseurs et les processeurs de traitement des signaux numériques (en anglais *Digital Signal Processor, DSP*), qui implémentent la partition logicielle du système. On peut considérer les composants *reconfigurables* tels que les FPGA (*Field-Programmable Gate Array*), surtout s'ils peuvent être reconfigurés à l'exécution, comme des composants programmables à cet égard. Ils présentent des caractéristiques de surface, de coût, de performance et de puissance qui sont intermédiaires entre le matériel dédié et les processeurs. Tous les composants sont connectés par des bus et des réseaux standards ou dédiés, et les données sont stockées dans un ensemble de mémoires.

2.5 Intérêt des systèmes embarqués pour la reconnaissance vocale des langues camerounaises

2.5.1 Intérêts/Motivations pour la conception des systèmes de reconnaissance vocale des langues camerounaises

Yemmene et al. dans [57], ont présenté les motivations ou intérêts de la recherche et du développement de la reconnaissance automatique de la parole dans les langues minoritaires de

4. Les boîtiers contenant des circuits intégrés peuvent être empilés verticalement sur un substrat. Elles sont reliées intérieurement par des fils fins qui sont collés au boîtier. Alternativement, avec une technologie de puce retournée, des bosses de soudure sont utilisées pour joindre les puces empilées ensemble.

l'Afrique subsaharienne, en prenant comme exemple une langue camerounaise, la langue *ngiemboon*. Ces intérêts sont de plusieurs ordres : des intérêts sociolinguistiques, des intérêts liés au retard d'alphabétisation, des intérêts de développement économique et communautaire, des intérêts juridiques et des intérêts scientifiques et technologiques.

Intérêt sociolinguistiques

Les langues locales camerounaises sont parlées soit dans le village de leurs locuteurs natifs, soit dans leurs foyers, et sont souvent utilisées pour le patrimoine et l'identification culturelle[66]. Dans ce paysage linguistique diversifié, de nombreuses industries et domaines n'ont actuellement accès à la reconnaissance automatique de la parole que dans les langues à forte ressource que sont le français et l'anglais, où actuellement, les systèmes de reconnaissance automatique de la parole trouvent une grande variété d'applications dans les domaines de l'assistance médicale, robotique industrielle, médecine légale et application de la loi, défense et aviation, industrie des télécommunications, domotique et contrôle d'accès sécurisé, informatique et électronique grand public[67]. Aussi vitales que ces applications puissent être, elles restent un luxe pour les locuteurs de langues locales camerounaises. Ainsi construire des applications de reconnaissance automatique de la parole pour les langues camerounaises serait une opportunité très intéressante pour le développement économique, social et communautaire de leurs locuteurs.

Pallier au déficit d'alphabétisation

D'après l'Institut de statistique de l'UNESCO [5], le taux d'alphabétisation, total des adultes (% des personnes âgées de 15 ans et plus) au Cameroun, est de 77% (les derniers chiffres sont de 2018 [6]). Ce même institut stipule qu'environ 3 316 943 millions d'individus (% des personnes âgées de 15 ans et plus) ou plus vivant au Cameroun sont analphabètes (les derniers chiffres sont de 2018). Nous n'avons pas de raison de croire que cela a beaucoup changé au cours des dernières années. Dans un contexte d'analphabétisme tel que celui-ci, l'utilisation de la communication orale est prépondérante et pratique. L'interaction homme machine par la voix a un grand potentiel pour le développement économique et communautaire.

Intérêts de développement économique et communautaire

Au Cameroun, ces dernières années, l'industrie de la téléphonie mobile a connu un essor important, dans cette région du monde où l'usage du téléphone portable est devenu très répandu. Par exemple, le nombre total abonnements mobiles en 2019 est évalué à 21 400 736[68] tous les opérateurs confondus. Cela constitue des opportunités accrues d'interaction Homme-Machine. Comme les téléphones mobiles ne nécessitent qu'une alphabétisation de base, ils sont accessibles à une grande partie de la population, quel que soit son niveau d'alphabétisation. Outre la communication vocale, ils permettent le transfert de données, qui peuvent être utilisées dans le cadre d'applications vocales à des fins de santé, d'éducation, de commerce et/ou de gouvernance. Les téléphones mobiles peuvent être utilisés comme un mécanisme permettant d'assurer une plus grande participation des différents segments de la population aux efforts de développement communautaire. Les innovations de ce type augmenteront la probabilité de connecter les locuteurs des langues camerounaises aux informations vitales dont ils ont besoin pour améliorer leur qualité de vie et contribuer au développement de leurs communautés.

5. Organisation des Nations unies pour l'éducation, la science et la culture
6. http ://uis.unesco.org/fr/country/cm

Intérêts juridiques

Une autre motivation pour développer un système de reconnaissance automatique de la parole en langues camerounaises est de permettre à cette communauté linguistique d'exercer l'un de ses droits fondamentaux exprimé dans l'article 40 de la Déclaration Universelle des Droits Linguistiques, *"Dans le domaine des technologies de l'information, toutes les communautés linguistiques ont le droit de disposer d'un équipement adapté à leur système linguistique et d'outils et produits dans leur langue, afin de tirer pleinement parti des possibilités offertes par ces technologies pour l'expression personnelle, l'éducation, la communication, l'édition, la traduction et le traitement de l'information et la diffusion de la culture en général "* . En outre, dans l'article 47, *"Tous les membres d'une communauté linguistique ont le droit de disposer, dans leur propre langue, de tous les moyens nécessaires à l'exercice de leurs activités professionnelles, tels que les documents et ouvrages de référence, les instructions, les formulaires, ainsi que les équipements, outils et produits informatiques"* . Le droit linguistique identifié s'aligne sur la priorité des droits de l'homme du 21e siècle. En plus de donner à cette communauté linguistique l'opportunité d'exercer l'un de ses droits fondamentaux, le développement d'un système de reconnaissance automatique de la parole pour les langues camerounaises peut être également une entreprise fascinante.

Intérêts Scientifiques

Le développement d'un système de reconnaissance vocale dans les langues camerounaises jouera un rôle important dans la revitalisation et la sauvegarde de la langue. Il fournira également un cadre pour la documentation numérique de ces langues. Étant donné les complexités et les particularités linguistiques de ces langues (langues peu dotées, langues à tons), une recherche sur la reconnaissance de la parole en ces langues pourrait aboutir à des découvertes susceptibles d'enrichir les connaissances scientifiques existantes et croissantes dans ce domaine passionnant et stimulant qu'est la reconnaissance automatique de la parole dans les langues peu dotées.

2.5.2 Intérêts de la mise en œuvre de la reconnaissance vocale sur un système embarqué.

La reconnaissance vocale est très souvent utilisée comme interface de communication homme-machine pour de nombreuses applications telles que la domotique [69] où on exploite des puces de type microcontrôleur[70] et même des FPGA[71] ; les systèmes de sécurité basés sur la reconnaissance vocale[72] ; etc.

En fonction de l'application et du type de système de reconnaissance (voir section 2.3.2), il est probable que le traitement impose des contraintes au système de traitement. À cet effet, Melnikoff dans [73] estime que : *"il peut être avantageux de transférer le traitement de la parole vers une forme de coprocesseur ou une autre implémentation matérielle"*. Pour la plupart des applications de reconnaissance vocale, il est largement suffisant de produire des résultats en temps réel, et il existe déjà des solutions logicielles à cet effet. Cependant, il existe plusieurs scénarios qui nécessitent des vitesses de reconnaissance plus rapides, et qui pourraient donc bénéficier d'une accélération matérielle. Par exemple, dans les applications de centres d'appels téléphoniques, le système de reconnaissance vocale doit traiter un grand nombre de requêtes vocales en parallèle.

En outres, il est possible d'économiser du temps et de l'argent également pour des applications similaires non en temps réel, telles que la transcription hors ligne pour la dictée, où un seul système serait capable de traiter plusieurs flux de parole à grande vitesse.

Une plus grande puissance de calcul peut également être utilisée pour passer des systèmes à locuteur dépendant à des systèmes à locuteur indépendant, ou pour rendre le système plus robuste et moins sensible au bruit de fond.

Dans le cadre de leur travaux, les ingénieurs et chercheurs ont besoin des données sur les langues camerounaises. Ces données peuvent être utilisées pour des applications telles que la traduction automatique ou la transcription de l'audio en texte. Cela peut être réalisé à l'aide d'une application s'exécutant sur un système embarqué pour effectuer la collecte sur site avec prétraitement direct pour éviter des mauvaises données qui nuiraient à la qualité des algorithmes d'apprentissage.

Les langues camerounaises sont en général peu dotées, d'où la nécessité de construire des corpus pour la conception des outils de traitements automatiques de la langue. La transcription textuelle via la reconnaissance vocale s'avère être une approche assez efficace pour la construction de ces corpus. Cependant, à travers une analyse faite par Bitjaa Kody[4], sur la dynamique des langues camerounaises en contact avec le français, on découvre que 60% d'enfants issus des ménages citadins possèdent une compétence de compréhension et d'élocution des langues de leurs parents et 40% ont le français comme seule et unique langue de communication. Contrairement aux enfants de la zone rurale qui pratiquent tous couramment la langue de leurs parents. Il est logique qu'il soit plus facile de trouver des locuteurs en zone rurale qu'en zone urbaine. Or les zones rurales sont en général caractérisées par une faible pénétration de la connexion internet et un faible taux d'électrification soit seulement 14% pour le cas des zones rurales au Cameroun[74]. Il est par conséquent plus intéressant de construire des systèmes de reconnaissance vocale autonomes.

2.6 Architectures électroniques pour la reconnaissance vocale sur système embarqué

Nous avons vu qu'il est pertinent de concevoir un système embarqué autonome permettant de faire de la reconnaissance vocale. Il est évident que celui-ci doit être temps réel, c'est-à-dire satisfaire une contrainte permettant son utilisation. C'est une contrainte molle dans ce cas qui doit être définie en fonction du débit de la parole humaine.

Les différentes architectures existantes pour concevoir un système embarqué sont : les processeurs graphiques (*Graphics Processing Unit, GPU*)[75], des circuits intégrés spécialisés (*ASIC*)[76], les processeurs de traitement numérique de signal(*DSP*)[77] et les réseaux logiques programmables (*FPGA*)[78]. L'implémentation matérielle spécialisée réduit les mouvements de données, la consommation d'énergie, la consommation de surface, etc. ; en utilisant plusieurs techniques de flux de données et de réutilisation des données.

Une autre approche consiste à exploiter la simultanéité induite du module de traitement en vue d'exécuter les différentes parties sous forme de taches concurrentes déployées sur plusieurs unités de traitement ; on parle alors de parallélisation[79]. L'exécution simultanée des différentes tâches permet d'obtenir un gain en temps d'exécution comparé à une exécution séquentielle.

Dans le cadre de cette thèse nous adressons une approche hybride pour optimiser le temps de réponse des tâches de traitement. Précisément, il est question de paralléliser une application et la mettre en œuvre sur architecture reconfigurable à l'aide d'accélérateurs matériels. L'idée d'une telle approche est de gagner en performance autant sur l'efficacité de l'algorithme que sur la plateforme matérielle d'implémentation.

Les architectures reconfigurables, notamment les FPGA, constituent une plate-forme viable pour l'accélération d'applications. Cependant, ces plateformes restent moins attrayantes pour les développeurs car le flot de conception habituel nécessite une bonne maîtrise des langages de description du matériel (par exemple VHDL et Verilog), qui sont de bas niveaux. Ainsi, implémenter une application parallèle sur ce type de plateforme induit de grandes problématiques pour le développeur. La première renvoie à la difficulté de conception de l'application et du modèle parallèle associé. La seconde problématique est liée à l'implémentation proprement dite du modèle d'application parallèle obtenue. Cette implémentation induit de nombreux sous problèmes

notamment la mise en œuvre de l'infrastructure d'interconnexion, le choix et le déploiement des éléments processeurs, le choix et la mise en œuvre du modèle de communication, la mise en œuvre du modèle de programmation parallèle et l'implémentation et déploiement des tâches parallèles. Le développeur est submergé par beaucoup d'autres problématiques en dehors de l'application parallèles à concevoir et à déployer. Dans ce travail, nous adressons la problématique de productivité de conception d'applications parallèles sur architecture reconfigurable. Spécifiquement l'objectif est de trouver des réponses aux questions suivantes : Comment permettre au développeur de se concentrer essentiellement sur l'application à déployer via un modèle parallèle sur architecture reconfigurable sans se soucier des autres problématiques liées à la plateforme ? Dans quelle mesure notre approche de solution peut exploiter les langages de hauts niveaux connus par les développeurs d'application parallèles sur environnement logiciel ? Et comment cette approche peut intégrer la co-conception matérielle logicielle ?

Au vu de la complexité de l'architecture d'une application de reconnaissance vocale tel que présenté dans la section 2.3.4, nous considérons un sous-ensemble des composants de cette architecture. La prochaine section spécifie la délimitation faite sur cette application pour la suite de ce travail.

2.7 Reconnaissance d'une langue camerounaise sur embarqué

2.7.1 Spécification fonctionnelle de l'application

Comme nous l'avons dit dans la section 2.2.4, les langues camerounaises sont peu dotées et on aimerait construire des outils de transcription automatique de la parole au texte pour la construction rapide et efficace des corpus. Un système de reconnaissance vocale constitue l'outil et la technologie adéquate pour répondre efficacement à cette problématique. Par ailleurs, la plupart de ces langues sont des langues à tons, il est donc nécessaire de construire des systèmes capables de faire une bonne discrimination entre les paires minimales pour ne pas biaiser les résultats escomptés.

Pour un but expérimental, nous nous sommes proposés de construire un système de reconnaissance des mots isolés dont le vocabulaire est constitué de paires minimales de la langue *Kóló*. La langue *Kóló* de la famille des langues Bantou[80] est parlée dans la partie du centre du Cameroun par plus de 2 500 000 personnes c'est la langue nationale la plus parlée à Yaoundé la capitale et les régions voisines.

2.7.2 Délimitation de l'application : extraction des caractéristiques pour la reconnaissance des langues à tons

D'après les objectifs de l'application présentée ci-avant, le système associé se doit d'être temps réel. Les systèmes embarqués constituent une plateforme intéressante pour la mise en œuvre de cette application. Dans la section 2.3.4 nous avons vu l'architecture d'un système de reconnaissance vocale. Dans le cadre de cette thèse nous allons nous limiter à la partie extraction des caractéristiques de cette architecture. Dans la reconnaissance de la parole, l'extraction de caractéristiques vise à donner une représentation utile du signal vocal en capturant les informations importantes de celui-ci. Il existe différentes façons de décrire le signal de parole en termes de caractéristiques, parmi toutes ces méthodes, le pitch (fréquence fondamentale F_0) est la caractéristique qui, seule ou en combinaison avec d'autres, rend la reconnaissance des langues tonales plus précise [81]. Dans le cadre de cette thèse, nous nous proposons de mettre en œuvre une implémentation parallèle à l'aide du langage MPI-2 RMA de l'un des algorithmes d'estimation de pitch parmi les plus précis, l'idée étant d'accélérer cette étape importante pour une meilleure performance globale des applications de reconnaissance vocale des langues tonales en termes de *latence* et de *débit* comme suggéré dans [82]. En d'autres termes, il est question de la mise en

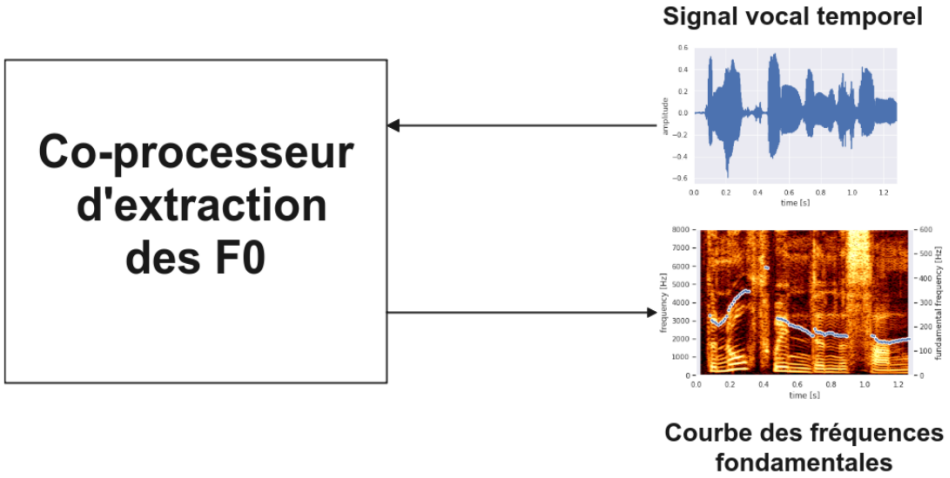

Figure 2.4: *Co-processeur d'extraction des fréquences fondamentales pour la reconnaissance des langues à tons*

œuvre sur système embarqué d'un module d'extraction des caractéristiques, une sorte de coprocesseur qui prend en entrée des échantillons de signal vocal et renvoie en sortie la courbe des fréquences fondamentales utiles pour la discrimination des langues à tons telles que la langue Kóló. La figure 2.4 illustre de façon schématique la description de ce module.

2.7.3 Algorithmes d'extraction de Pitch/F_0

La fréquence fondamentale est définie comme la fréquence à laquelle les cordes vocales vibrent lors d'un son voisé[83]. Une solution algorithmique pour la détermination du Pitch, appelée Pitch Determination Algorithm (PDA), peut être subdivisée en trois étapes de traitement : (i) le préprocesseur, (ii) l'extracteur de base, et (iii) le postprocesseur[84]. Le préprocesseur a pour tâche de réduire les données afin de faciliter le fonctionnement de l'extracteur de base, qui effectue la véritable tâche de mesure : il convertit le signal d'entrée en une série d'estimations du pitch. Le post-processeur fonctionne de manière plus orientée vers l'application : correction des erreurs, lissage du contour du pitch et affichage graphique sont quelques-unes de ses tâches possibles. La catégorisation des PDA peut être faite en fonction du domaine du signal d'entrée de l'extracteur de base. Le premier groupe est caractérisé par un signal temporel comme signal d'entrée, ces assistants numériques travaillent dans le domaine temporel. C'est le cas de ceux basés sur la fonction d'auto-corrélation (ACF)[85], de l'algorithme robuste de suivi de pitch (RAPT :Robust Algorithm for Pitch Pracking)[86], de l'approche YIN[87], etc. Le deuxième groupe opère dans le *domaine fréquentiel* en exploitant la structure harmonique du spectre comme l'algorithme SWIPE (Sawtooth Waveform Inspired Pitch Estimator)[88], et le SHS [89]. Le dernier groupe opère dans les deux domaines comme Aurora[90] qui repère les candidats F0 dans le domaine fréquentiel et affine ensuite la détection dans le domaine temporel, ou NDF[91] qui exploite l'auto-corrélation sous-bande et le calcul de la fréquence instantanée.

2.7.4 Choix de l'algorithme d'extraction de Pitch/F_0 pour la reconnaissance des langues camerounaises

Toutes les méthodes ci-dessus sont intéressantes en tant que PDA mais il est important à un moment donné de pouvoir faire un choix lors de leur utilisation dans une application. C'est pour cette raison que certains auteurs ont travaillé sur des évaluations de méthodes d'estimation de F_0 comme dans [92]. Le travail qui a attiré notre attention est celui de Sofia Strömbergsson[93] qui,

par le biais d'une enquête bibliométrique, examine quelles méthodes ont été les plus fréquemment utilisées dans la communauté scientifique de la parole au cours des années 2010-2016, puis les méthodes les plus utilisées ont été évaluées par rapport à une référence de vérité-terrain, avec un accent particulier sur leur précision dans l'estimation de F0 chez les locuteurs masculins et féminins, respectivement. Les résultats montrent que Praat ACF[85] est de loin la méthode dominante, suivie de STRAIGHT[91], RAPT et YIN. Dans l'évaluation incluant Praat ACF, RAPT et YIN avec leurs paramètres par défaut et adaptés au sexe, Praat ACF s'est également avéré être le plus précis[93]. Tous ces éléments justifient le choix de Praat ACF comme méthode retenue dans le cadre de cette thèse.

2.8 Résumé du chapitre

Le Cameroun est un pays culturellement riche en partie par sa grande diversité linguistique avec environs 284 langues locales recensées, réparties dans trois grandes familles de langues sur les quatre identifiées en Afrique. Ces langues ont majoritairement en commun deux caractéristiques : la première étant qu'elles sont des langues à tons, c'est-à-dire des langues dont le sens des mots varie en fonction des tons que portent les syllabes ; la deuxième caractéristique correspond au fait qu'elles sont toutes des langues peu dotées, c'est-à-dire qu'il n'existe pas assez de ressources pour la pérennisation de celles-ci. Dans le sens de la reconnaissance vocale, ce sont des langues qui ne possèdent pas encore ou pas beaucoup (en quantité et en qualité) de ressources linguistiques pour la construction d'un système de reconnaissance automatique de la parole. Ces deux caractéristiques constituent donc des enjeux majeurs pour la conception des systèmes de reconnaissance vocale sur système embarqué. Une catégorie d'applications avec une architecture assez stable et connue dans l'état de l'art. La reconnaissance vocale des langues a tons a déjà été exploré par un grand nombre d'auteurs, et Jaspreet Kaur et al.[47] en présentent une liste non exhaustive y compris les travaux sur les langues africaines et camerounaises.

La conception d'un système de reconnaissance vocale sur système embarqué pour les langues camerounaises adressée dans le cadre de cette thèse est motivée par un ensemble d'éléments qui sont liés : à des intérêts sociolinguistiques, au déficit d'alphabétisation, à des intérêts de développement économique et communautaire, à des intérêts juridiques et même scientifiques. Le choix des systèmes embarqués sur architecture reconfigurable comme espace de solution quant à lui est motivé par un ensemble de contraintes non fonctionnelles liées, au caractère embarqué des applications dans lesquelles la reconnaissance vocale est utilisée comme interface utilisateur, à la nécessité d'accélération de ce type d'applications liée aux contraintes de temps réel, aux contraintes de connectivité à d'autre système et de consommation en énergie. En outre, à ce choix de plateforme, nous associons la mise en œuvre de l'application sous un modèle parallèle pour plus de performance. Toutefois, ces différents choix impliquent des problématiques autant du point de vue applicatif car la conception d'un modèle parallèle n'est pas évidente ; que du point de vue plateforme d'implémentation car le flot de conception des applications sur architecture reconfigurable est assez complexe. Nous adressons donc cette problématique à deux volets dans la suite de ce travail.

En ce qui concerne le système de reconnaissance vocale, nous nous limitons à l'implémentation de l'extraction des fréquences fondamentales (F_0), l'une des caractéristiques les plus discriminantes pour la reconnaissance des langues à tons. Nous implémenterons donc l'algorithme d'auto-corrélation de Praat pour illustrer la validité et l'efficacité de la méthode que nous proposons pour améliorer la productivité de la conception des applications parallèles sur des systèmes embarqués à architecture reconfigurable. Le prochain chapitre présente l'état de l'art des approches existantes dans le domaine.

Chapitre

3 Conception d'un système embarqué parallèle sur architecture reconfigurable

Sommaire

3.1 Introduction

La reconnaissance vocale constitue une technologie importante pour la numérisation des langues africaines, spécifiquement celles camerounaises (généralement peu dotées). Quel que soit le cas, les systèmes embarqués sont la plateforme technologique par excellence pour le déploiement de ce type d'application. Pour plus d'efficacité, un système de reconnaissance vocale se doit d'être temps réel. A cet effet, en dehors de l'usage des plateformes d'exécution qui accélèrent le processus de traitement (DSP, GPU, etc.), la parallélisation constitue un moyen fiable pour l'atteinte des objectifs d'efficacité et de temps réel. Il est donc important d'explorer comment mettre œuvre la parallélisation de notre application sur un système embarqué. Il est également nécessaire de regarder dans quelle mesure il est possible de décharger le designer au maximum des problématiques liées aux systèmes parallèles et à celles intrinsèques à l'architecture utilisée. Dans ce chapitre, nous déroulons cette exploration dans le cadre d'un environnement logiciel (système parallèle constitué de CPUs), puis dans le cadre d'un système embarqué en parcourant l'état de l'art sur les approches de mise en œuvre productive des applications parallèles sur architecture parallèle sur MP-RSoC. La suite du chapitre est structurée comme suit : dans la première section nous rappelons les concepts clés sur les systèmes parallèles, la section suivante met en exergue le rapport et l'évaluation de la mise en œuvre d'un modèle parallèle de l'extraction des caractéristiques pour la reconnaissance des langues à tons, nous continuons avec les systèmes reconfigurables, leurs atouts et l'état de l'art sur les approches de mise en œuvre productive des applications parallèle sur MP-RSoC. Après synthèse nous clôturons avec une identification d'approche de solution pour une mise en œuvre productive d'applications parallèles MPI-2 RMA sur architecture reconfigurable.

3.2 Notions sur les systèmes parallèles

Le calcul parallèle est une méthode permettant d'accélérer le temps de traitement en parallélisant certaines de tâches à réaliser. Dans la suite de cette section, nous rappelons les concepts clés liés aux systèmes parallèles.

3.2.1 Les architectures parallèles : classification de flynn

La taxonomie de Flynn[94] est le schéma de classification le plus connu pour les systèmes parallèles. Dans ce schéma, la catégorie dépend du parallélisme présent dans le flux d'instructions et le flux de données. Un processus peut être considéré comme l'exécution d'une séquence d'instructions (le flux d'instructions) qui manipule une séquence d'opérandes (le flux de données). La classification de Flynn aboutit donc à quatre catégories (SISD, SIMD, MISD, MIMD).

SISD (Single Instruction stream Single Data stream)

La catégorie SISD fait référence au traitement d'un seul flux d'instructions et un seul flux de données.

SIMD (Single Instruction stream Multiple Data stream)

La catégorie SIMD caractérise les systèmes parallèles ayant un seul flux d'instructions mais plusieurs flux de données.

MISD (Multiple Instruction stream Single Data stream)

La catégorie MISD correspond aux systèmes parallèles ayant plusieurs flux d'instructions, mais un seul flux de données. Cela correspond souvent à un pipeline composé de plusieurs unités fonctionnelles indépendantes fonctionnant sur un seul flux de données et transmettant les résultats d'une unité fonctionnelle à la suivante.

MIMD (Multiple Instruction stream Multiple Data stream)

La catégorie MIMD concerne les systèmes parallèles ayant plusieurs flux d'instructions et plusieurs flux de données.

3.2.2 Les architectures parallèles : classification basée sur l'organisation de la mémoire

Il existe un critère de classification lié à l'organisation interne de la mémoire de l'architecture. Deux types d'architectures se distinguent fondamentalement.

Les architectures à mémoire partagée

Dans les architectures à mémoire partagée (voir Figure 3.1), toutes les unités de traitement partagent les mêmes ressources mémoire ; par conséquent, toutes les modifications apportées à un emplacement mémoire donné sont visibles par toutes les unités de traitement du système. Du point de vue de la conception de l'architecture, les machines à mémoire partagée sont peu évolutives entre la mémoire et le CPU. L'ajout d'une unité de traitement supplémentaire peut augmenter par un facteur k le trafic sur le chemin partagé entre la mémoire et les unités de traitement, et pour les systèmes à cohérence de cache, augmenter par un facteur k le trafic associé à la gestion du cache et de la mémoire. Les architectures à mémoire partagée nécessitent des mécanismes de synchronisation tels que les sémaphores, les barrières et les verrous, car il n'existe pas de communication explicite. Dans une architecture à mémoire partagée, différents processus peuvent facilement échanger des informations par le biais de variables partagées, mais cela nécessite une gestion minutieuse de la synchronisation et de la protection de la mémoire.

Les architectures à mémoire distribuée

Dans les architectures à mémoire distribuée (voir Figure 3.2), chaque unité de traitement possède sa propre mémoire privée ; par conséquent, elle ne peut pas lire directement dans la mémoire d'une autre unité de traitement. Les transferts de données sont réalisés à l'aide de protocoles de passage de messages. Les machines à mémoire distribuée sont plus évolutives puisque seul le support de communication est partagé entre les unités de traitement.

Les architectures à mémoire distribuée nécessitent des mécanismes pour supporter les communications explicites entre les processus. Habituellement, on utilise une bibliothèque de primitives qui permettent d'écrire dans des canaux de communication. MPI (Message Passing Interface) [95]

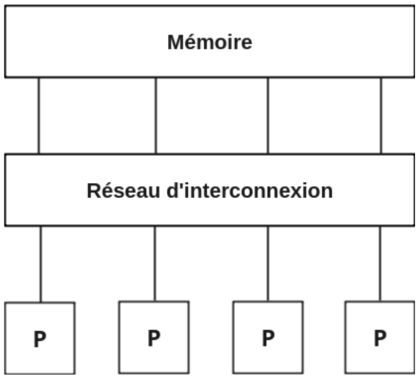

Figure 3.1: *Architecture parallèle à mémoire partagée*

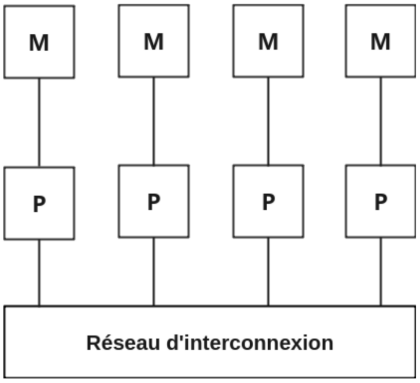

Figure 3.2: *Architecture à mémoire distribuée*

est la norme la plus populaire. Dans une architecture à mémoire distribuée, une infrastructure des communications est nécessaire pour connecter les unités de traitement et leurs mémoires et permettre l'échange d'informations.

3.2.3 Les modèles de programmation parallèles

Les différents modèles de programmation que nous mettons en avant ici s'articulent autour de l'architecture matérielle ; plus précisément sur sa façon de gérer la mémoire entre les différents éléments processeurs.

Modèle de programmation sur architecture en mémoire partagée

Les modèles de programmation par mémoire partagée se basent sur un ensemble de nœuds d'exécution qui évoluent de façon concurrente dans un même espace d'adressage. Les communications (lectures/écritures) sont réalisées implicitement, par l'intermédiaire de la mémoire de la machine. De cette manière la tâche du programmeur est simplifiée en ce sens qu'il doit uniquement s'occuper du partage des tâches et potentiellement de la synchronisation en utilisant des verrous ou des sémaphores lors des accès à la mémoire partagée.

Le standard des threads POSIX (ou pthreads) [96] est très fréquemment rencontré : ces derniers permettent de créer divers processus légers dont l'exécution sera gérée par le système selon une politique d'ordonnancement définie en fonction du nombre de nœuds de calcul.

Il existe plusieurs langages de programmations et bibliothèques qui implémentent le standard des pthreads, notamment : OpenMP [97], Cilk[98], Intel Threading Building Blocks (TBB) [99], etc.

Modèle de programmation sur architecture à mémoire distribuée

Il s'agit des modèles de programmation par *passage de message* qui sont destinés principalement aux architectures à mémoire distribuée, où le seul moyen d'échanger des informations entre unités de traitement est l'envoi de messages sur le réseau d'interconnexion.

Le modèle le plus connu et le plus utilisé dans cette catégorie est MPI (Message Passing Interface)[100]. C'est la spécification d'un protocole de communication destiné à la base au calcul scientifique sur machines parallèles à mémoire distribuée grâce à des communications de type point-à-point ou des communications collectives, ayant pour buts d'être performantes, portables et capables de passer à l'échelle sur des machines de grande taille. MPI a connu un grand succès depuis sa création dans les années 90 et s'impose aujourd'hui comme l'outil le plus populaire pour les communications par passage de message.

Le standard de communication MPI-2 RMA

La spécification originale de MPI (MPI-1) est un modèle de communication par passage de messages basé sur des opérations d'envoi et de réception. Dans ce modèle, la communication implique à la fois l'émetteur et le récepteur, et la synchronisation est réalisée implicitement par les opérations de communication. Ce modèle est également appelé communication bilatérale en anglais "*two-sided communication* ". En tant qu'extension de MPI-1, la norme MPI-2 [100] introduit le modèle de communication unilatérale en anglais "*One-sided communication* ". Dans ce modèle, un processus spécifie tous les paramètres de communication, et la synchronisation est effectuée explicitement pour assurer l'achèvement de la communication. Les opérations de communication unilatérale dans MPI-2 comprennent *MPI_ Put*, *MPI_ Get* et *MPI_ Accumulate*.

Contrairement aux modèles traditionnels de communication bilatérale et collective, la communication unilatérale dissocie le mouvement des données de la synchronisation, éliminant ainsi les coûts de synchronisation inutiles et permettant une plus grande concurrence. En outre, les coûts de mise en correspondance des messages et de mise en mémoire tampon qui sont nécessaires pour les communications bilatérales sont éliminés, ce qui entraîne une réduction significative des coûts de communication.

3.2.4 Méthodologies de conception d'applications parallèles

La clé du traitement parallèle est la simultanéité exploitable. La simultanéité existe dans un problème de traitement de l'information lorsque le problème peut être décomposé en sous-problèmes qui peuvent être exécutés en toute sécurité en même temps. Pour être utile, cependant, il doit être possible de structurer le programme pour exposer et ensuite exploiter la simultanéité et permettre aux sous-problèmes de s'exécuter réellement en même temps ; en d'autres termes, la simultanéité doit être exploitable. Décomposer et structurer ces tâches de manière à être efficace compte tenu du problème est un défi pour le développeur. C'est la raison pour laquelle des auteurs ont proposé des méthodologies pour l'atteinte de cet objectif. Il existe plusieurs méthodes dans la littérature, notamment la méthode PCAM, la parallélisation incrémentale, la méthode de parallélisation automatique, la méthode Skeleton-Based[101] présentée dans [102]. Celle qui revient beaucoup dans l'état de l'art est la méthode PCAM de Foster[103], que nous allons décrire dans la suite de cette section.

La méthodologie de conception décrite par Foster a pour but de favoriser une approche exploratoire de la conception, dans laquelle les questions indépendantes de la machine, telles que la concurrence, sont prises en compte très tôt et les aspects de la conception spécifiques à la machine sont reportés à une étape ultérieure du processus de conception. Cette méthodologie structure le processus de conception en quatre étapes distinctes : le *partitionnement*, la *communication*, l'*agglomération* et le *mappage* d'où l'acronyme PCAM. Dans les deux premières étapes,

cette méthode se concentre sur la *concurrence*[1] et la *scalabilité*[2] et cherche à découvrir des algorithmes possédant ces qualités. Dans les troisième et quatrième étapes, l'attention se déplace vers la *localité*[3] et d'autres questions liées aux performances.

Le partitionnement

L'étape de partitionnement d'une conception a pour but d'exposer les possibilités d'exécution parallèle. Il s'agit ici de décomposer l'application en un certain nombre de taches qui pourront s'exécuter en parallèle. Dans les étapes ultérieures de la conception, l'évaluation des exigences de communication, de l'architecture cible ou des questions d'ingénierie logicielle peuvent nous amener à renoncer aux possibilités d'exécution parallèle identifiées à ce stade. Une bonne partition divise en morceaux à la fois le calcul associé à un problème et les données sur lesquelles ce calcul opère.

La communication

Les tâches générées par un partitionnement sont destinées à s'exécuter simultanément mais ne peuvent s'exécuter totalement indépendamment. Le calcul à effectuer dans une tâche nécessite généralement des données associées à une autre tâche. Les données doivent alors être transférées entre les tâches afin de permettre la poursuite du calcul. Ce flux d'informations est spécifié dans la phase de communication d'une conception. La communication associée à un algorithme peut être spécifiée en deux phases. Premièrement, on définit une structure de canal qui relie, directement ou indirectement, les tâches qui ont besoin de données (consommateurs) aux tâches qui possèdent ces données (producteurs). Ensuite, on spécifie les messages qui doivent être envoyés et reçus sur ces canaux.

Penser en termes de tâches et de canaux aide à réfléchir quantitativement aux problèmes de localité et aux coûts de communication afin d'optimiser les performances en distribuant les opérations de communication de manière à permettre une exécution simultanée.

L'agglomération

L'algorithme qui résulte des deux premières étapes du processus de conception reste abstrait dans le sens où il n'est pas spécialisé pour une exécution efficace sur une architecture électronique particulière. Dans l'étape d'agglomération, on passe de l'abstrait au concret. On réexamine les décisions prises dans les phases de partitionnement et de communication en vue d'obtenir un algorithme qui s'exécutera efficacement sur une certaine architecture électronique. En particulier, on se demande s'il est utile de combiner, ou d'agglomérer, les tâches identifiées par la phase de partitionnement, de manière à obtenir un plus petit nombre de tâches, chacune de plus grande taille. On détermine également s'il est utile de répliquer les données et/ou les calculs.

Le nombre de tâches résultant de la phase d'agglomération, bien que réduit, peut toujours être supérieur au nombre de ressources disponibles dans l'architecture électronique, par exemple le nombre de processeurs. Dans ce cas, la conception reste quelque peu abstraite, puisque les questions relatives à l'affectation des tâches ne sont pas résolues.

Le mappage

Dans la quatrième et dernière étape du processus de conception d'un algorithme parallèle, il est spécifié sur quelle ressource de l'architecture électronique les taches de l'application doivent

1. La concurence fait référence à la capacité d'effectuer de nombreuses actions simultanément.
2. La scalabilité indique la résistance à l'augmentation du nombre de processeurs.
3. La localité renvoie à un rapport élevé entre les accès à la mémoire locale et les accès à la mémoire distante (communication).

s'exécuter. En général, le mappage reste un problème difficile qui doit être abordé explicitement lors de la conception d'algorithmes parallèles.

Il existe plusieurs outils réalisant une analyse de l'application permettant de faire le mappage ; parmi les plus connues, on distingue : AIMS (Automated Instrumentation and Monitoring System)[104], nupshot distribué avec MPE[105], Pablo[106], SvPablo[107], Paradyn[108], VAMPIR[109], etc.

Pour l'optimisation du temps de réponse de notre application d'extraction des caractéristiques pour la reconnaissance vocale des langues à tons sur système embarqué, nous avons adopté une structuration à mémoire distribuée pour ses divers avantages, et accompagné du paradigme MPI-2 RMA. Dans la prochaine section nous proposons une première version de la mise en œuvre du modèle parallèle sur environnement logiciel avec CPU.

3.3 Une première mise œuvre du modèle parallèle d'un algorithme d'extraction des caractéristiques pour la reconnaissance des langues à tons

Dans la section 2.7.2 nous avons fait une délimitation de notre application à l'extraction des caractéristiques pour la reconnaissance des langues à tons. A travers une exploration de l'état de l'art nous avons pû identifier l'un des algorithmes d'extraction des fréquences fondamentales les plus précis et les plus utilisés de l'état de l'art, à savoir la fonction d'auto-corrélation de Praat. Dans la suite de cette section nous présentons cet algorithme et faisons un rapport de la mise en œuvre sur CPU d'un modèle parallèle MPI-2 RMA de celui-ci.

3.3.1 Fonction d'autocorrélation de Praat

Praat ACF présenté dans[85] est selon la classification de Hess & al.[110], un *PDA d'analyse à court terme*. Dans tout PDA d'analyse à court terme, le signal vocal est divisé en une série de trames ; une trame individuelle est obtenue en prenant un nombre limité d'échantillons consécutifs du signal $s(n)$ depuis un point de départ, $n = q$, jusqu'au point d'arrivée, $n = q+K$. La longueur de la trame K (ou $K+1$) est choisie suffisamment courte pour que le(s) paramètre(s) à mesurer puisse(nt) être supposé(s) approximativement constant(s) dans la trame. D'autre part, K doit être suffisamment grand pour garantir que le paramètre reste mesurable. Pour la plupart des PDA d'analyse à court terme, une trame nécessite donc au moins deux ou trois périodes complètes.
L'idée de base de l'estimation de la fréquence fondamentale par la fonction d'autocorrélation est la suivante : pour un signal temporel $x(t)$ qui est stationnaire (c'est-à-dire que ses statistiques sont constantes), l'autocorrélation $r_x(\tau)$ en fonction du retard τ est définie comme dans l'équation (3.1). Cette fonction a un maximum global pour $\tau = 0$. S'il existe également des maxima globaux en dehors de 0, le signal est dit périodique et il existe un retard de T_0, appelé *période*, pour que tous ces maxima soient placés aux retards de nT_0, pour tout entier de n, avec $r_x(nT_0) = r_x(0)$. La *fréquence fondamentale* F_0 de ce signal périodique est définie comme $F_0 = 1/T_0$.

$$r_x(\tau) = \int_{-\infty}^{+\infty} x(t)x(t+\tau)dt \tag{3.1}$$

Plus précisément, les étapes de la détermination de la fréquence fondamentale d'un son par Praat ACF sont les suivantes :

1. La première étape de l'analyse est l'application d'une fonction de fenêtrage $w(t)$ sur la trame d'analyse $s(t)$. Pour cela, nous prenons du signal $x(t)$ une trame de durée T (le *longueur de fenêtre*), centrée autour de t_{mid}. On soustrait de cette trame sa moyenne μx et on multiplie le résultat par une fonction de fenêtrage $w(t)$ (ici les fenêtres de Hanning

définies dans l'équation (3.2)), de manière à obtenir le *signal fenêtré* $a(t)$ comme dans l'équation (3.3).

$$w(t) = \frac{1}{2} - \frac{1}{2}\cos\frac{2\pi t}{T} \tag{3.2}$$

$$a(t) = (x(t_{mid} - \frac{1}{2}T + t) - \mu x)w(t) \tag{3.3}$$

2. L'autocorrélation normalisée $r_a(\tau)$ de $a(t)$ est calculée comme dans l'équation (3.4). Les autocorrélations du signal fenêtré peuvent également être calculées numériquement par transformation de Fourier rapide. Ceci est possible car l'autocorrélation peut être obtenue en calculant d'abord la *transformation de Fourier* du signal fenêtré (équation (3.5)), ce qui donne dans le domaine de fréquence , puis en calculant la *transformation de Fourier inverse* de la *densité de puissance* $\mid \tilde{a}(\omega) \mid^2$, ce qui nous amène au domaine de retard comme dans l'équation (3.6).

$$r_a(\tau) = r_a(-\tau) = \frac{\int_0^{T-\tau} a(t)a(t+\tau)dt}{\int_0^T a^2(t)dt} \tag{3.4}$$

$$\tilde{a}(\omega) = \int a(t)e^{-i\omega t}dt \tag{3.5}$$

$$r_a(\tau) = \int \mid \tilde{a}(\omega) \mid^2 e^{i\omega\tau}\frac{d\omega}{2\pi} \tag{3.6}$$

3. L'autocorrélation normalisée $r_w(\tau)$ de la fonction fenêtre $w(t)$ est calculée d'une manière exactement analogue aux équations (3.4) ou (3.6) . L'autocorrélation normalisée d'une fenêtre de Hanning est définie par l'équation (3.7).

$$r_w(\tau) = (1 - \frac{\mid \tau \mid}{T})(\frac{2}{3} + \frac{1}{3}\cos\frac{2\pi\tau}{T}) + \frac{1}{2\pi}\sin\frac{2\pi \mid \tau \mid}{T} \tag{3.7}$$

4. L'autocorrélation $r_x(\tau)$ du segment de signal original est obtenue en divisant l'autocorrélation $r_a(\tau)$ du signal fenêtré par l'autocorrélation $r_w(\tau)$ de la fenêtre comme dans l'équation (3.8).

$$r_x(\tau) = \frac{r_a(\tau)}{r_w(\tau)} \tag{3.8}$$

5. Dans la dernière étape, nous choisissons les positions τ_i et les amplitudes $r(\tau_i)$ des n premiers extrema locaux dans l'autocorrélation corrigée $r(\tau_i)$. Maintenant, les $1/\tau_i$ sont les fréquences des hauteurs candidates et les $r(\tau_i)$, toujours un nombre compris entre 0 et 1, sont les *poids* des candidats. Plus $r(\tau_i)$ est proche de 1, plus le candidat est fort.

Dans notre mise en œuvre, nous avons utilisé la *transformation de Fourier inverse* de la *densité de puissance* pour calculer $r_a(\tau)$ et $r_w(\tau)$ selon l'équation (3.6). La figure 3.3 illustre de manière schématique les étapes de l'approche de calcul de la fonction d'autocorrélation adoptée et mise en œuvre dans ce travail.

3.3.2 Paramètres de l'algorithme

En tant que PDA d'analyse à court terme, l'algorithme Praat ACF a besoin d'un ensemble de paramètres pour calculer les meilleurs F_0 candidats à partir d'une trame d'échantillon[111]. Ces paramètres sont :

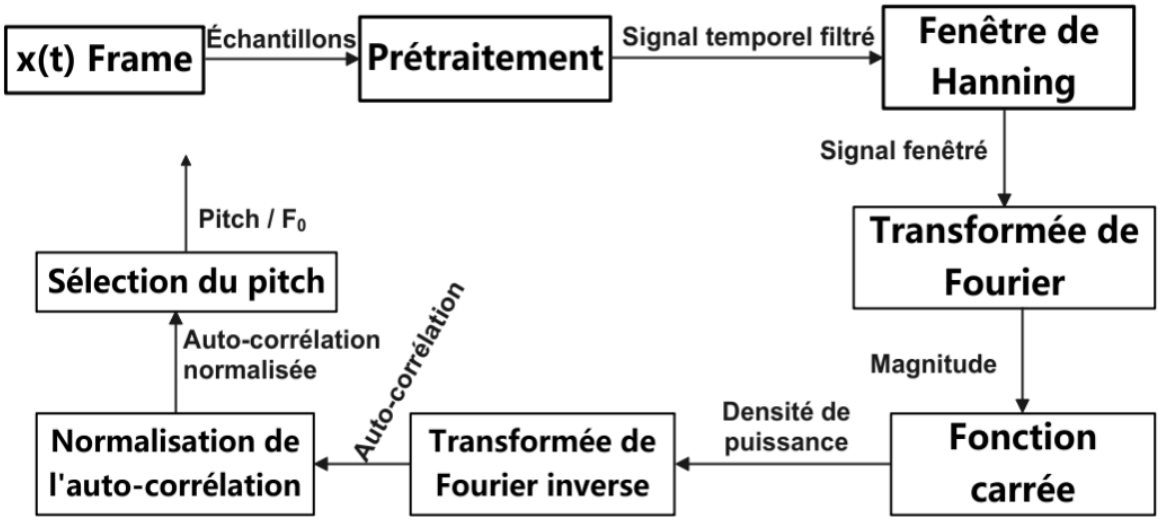

Figure 3.3: *Les étapes du calcul de la fonction d'autocorrélation de Praat basé sur l'approche de la densité spectrale de puissance (DSP).*

1. *Longueur de fenêtre* : Comme indiqué précédemment, le signal est découpé en petits segments qui seront analysés individuellement. La longueur de la fenêtre est la durée d'un tel segment ;
2. *Le pas de temps* (valeur standard pour Praat : 0,0 s) : Ce paramètre détermine la quantité de chevauchement entre les segments successifs.
3. Le *Pitch minimal* (valeur standard : 75 Hz) : La fréquence candidate la plus basse à prendre en compte.
4. Le *Nombre maximal de candidats* (valeur standard : 15) : Il détermine le nombre de maxima locaux dans l'autocorrélation qui doivent être retenus.
5. *Très précis* : Il détermine la fonction de fenêtre. Si *off*, une fenêtre de Hanning est utilisée avec la même durée que la fenêtre d'analyse. Si *on*, une fenêtre gaussienne (voir l'équation 3.9) est sélectionnée avec une durée deux fois supérieure à la longueur effective de la fenêtre.
6. *Seuil de silence*(valeur standard : 0,03). Les trames sonores dans lesquelles les plus grandes amplitudes ne dépassent pas cette valeur, par rapport au pic maximal global, sont considérées comme silencieuses et donc aphones.
7. *Pitch maximum* (valeur standard : 600 Hz). Les candidats qui dépassent cette fréquence sont ignorés. Pour les voix masculines, on peut abaisser le plafond à, environ 300 Hz.

$$w(n) = exp(-\frac{1}{2}(\frac{n - N/2}{\sigma N/2})^2), 0 \leq n \leq N, \sigma \leq 0.5 \tag{3.9}$$

Avec σ l'écart-type de la fonction gaussienne et N/2 la période d'échantillonnage.

3.3.3 Conception du modèle parallèle

Par l'exploitation de la méthodologie de Foster[103], notamment en suivant une approche de *décomposition fonctionnelle* telle que présentée ci-dessus, et en tenant compte de la structure intrinsèque de la fonction d'autocorrélation de Praat, nous avons abouti à un algorithme parallèle basé sur le modèle pipeline. Le choix du modèle pipeline s'explique ici par l'arrivée graduelle des données. En effet, les échantillons du signal vocal sont envoyés en flux par un microphone. Il est donc plus efficace de traiter une fenêtre à la fois, d'où le choix du modèle pipeline. L'algorithme

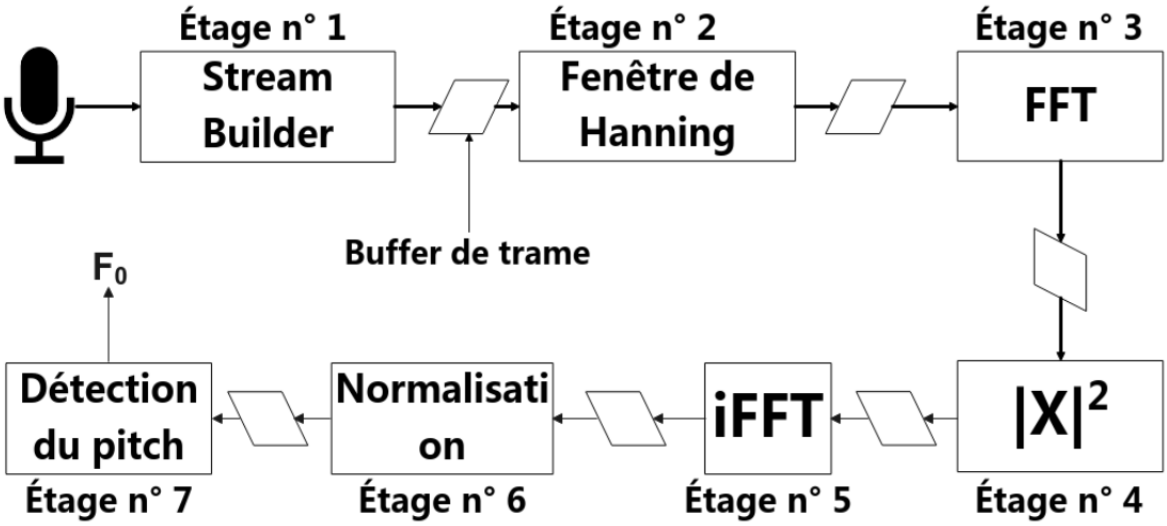

Figure 3.4: *Modèle pipeline de la fonction d'auto-corrélation de Praat*

parallèle ainsi obtenu est un pipeline à 7 étages (voir *Figure 3.4*). Le premier étage que nous avons nommé "*stream builder*" consiste à extraire le signal de la parole du microphone qui envoie les échantillons en temps réel. Le "*stream builder*" est ensuite chargé de diviser le signal de parole en un ensemble de trames qu'il envoie l'une après l'autre au reste du pipeline. Le deuxième étage effectue le fenêtrage de Hanning et envoie le résultat à l'étage 3 qui est responsable de la transformée de Fourier ; vient ensuite l'étage 4 qui effectue le carré point par point d'un vecteur. Les étages 5, 6 et 7 réalisent respectivement la transformée de Fourier inverse, la normalisation de l'autocorrélation et la détection des fréquences fondamentales comme décrit dans la section 3.3.1.

3.3.4 Conception du scénario de communication MPI

Pour tirer parti des performances offertes par le mécanisme d'accès distant à la mémoire, nous avons conçu un scénario d'implémentation qui exploite le mécanisme de communication RMA de MPI-2 tout en construisant le modèle de parallélisation pipeline. La *Figure 3.5* illustre ce scénario de communication. Chaque tâche traite une trame de données provenant de la tâche précédente (à l'exception de la première tâche qui opère sur les données provenant de la source comme un micro) par le biais de la primitive *MPI_PUT* [112]. Ensuite, elle traite l'information selon la fonction de calcul qui lui est allouée et écrit le résultat dans la mémoire locale de la tâche suivante. De proche en proche, les différentes tâches forment ainsi un pipeline.

3.3.5 Implémentation C++ MPI de Praat ACF

L'implémentation de la fonction d'autocorrélation de Praat a été réalisée en C++ en deux versions. À des fins expérimentales, nous avons d'abord réalisé une version séquentielle, puis nous avons développé la version parallèle MPI-2 RMA. L'application a été déboguée et compilée en utilisant l'implémentation Open MPI[113]. L'algorithme 1 montre la structure générale du code des différentes tâches du programme parallèle. Chaque tâche i expose sa fenêtre de communication ; lorsque les données sont disponibles, elle effectue le traitement qui lui est alloué puis transmet les résultats à la tâche suivante (tâche i + 1). A l'exception des tâches 1 et 7 qui ne comportent respectivement pas l'étape 1 correspondant à la réception des données de la tâche i - 1 et l'étape 3 qui correspond à l'envoi des données à la suivante (i + 1). L'environnement de test matériel était un ordinateur de 2,2 GHz, 12 noeuds *Intel(R) Core(TM) i7-8750H CPU* avec 8 Go de RAM, et l'environnement de test logiciel est basé sur Ubuntu 20.04 comme système

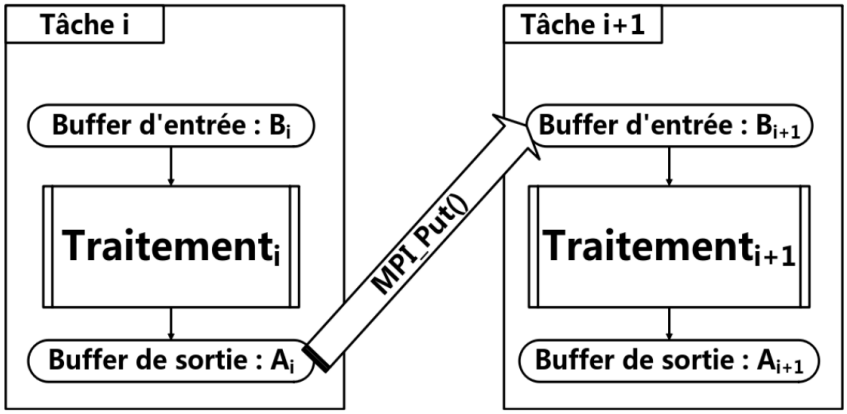

Figure 3.5: *Scénario de communication MPI-2 RMA en pipeline pour la fonction d'auto-corrélation Praat*

d'exploitation et OpenMPI 4.0.5 utilisé comme bibliothèque MPI (Message Passing Interface) exploitant C++ comme langage de programmation.

Le tableau 3.1 présente la configuration des variables telle qu'utilisée dans notre implémentation.

3.3.6 Expérimentations et résultats

Notre expérience a été réalisée sur un corpus de 8 paires minimales de mots de la langue *Kóló* enregistrés avec *Audacity*[4] à une fréquence de 8000 Hz. Pour valider notre implémentation, nous avons évalué le pourcentage de F_0 bien estimés (F_0 pour lesquels le pourcentage d'erreur entre le résultat de notre implémentation et celui produit par le logiciel Praat est inférieur à 10%). La précision obtenue pour les différents enregistrements est donnée dans le tableau 3.2. D'après cette évaluation, notre implémentation présente une précision très satisfaisante (voire excellente).

L'expérimentation a été réalisée sur un fichier audio intégrant toutes les paires minimales enregistrées pour un nombre total de 2058 trames. Nous avons d'abord exécuté la version séquentielle de l'algorithme sur toutes les trames et nous avons pris le temps d'exécution du traitement réel qui nous a donné 455,363 ms. Puis nous avons fait de même en utilisant la version parallèle et nous avons obtenu un temps d'exécution réel de 289,79 ms, ce qui donne un speedup de **1,571** selon l'équation 3.10 .

$$Speedup = \frac{Temps\ d'execution\ sequentielle}{Temps\ d'execution\ parallele} \tag{3.10}$$

Théoriquement, étant donné le nombre d'étages, nous devrions obtenir un facteur d'accélération proche du nombre d'étages, c'est-à-dire 7 pour notre cas. Le facteur d'accélération obtenu ici peut s'expliquer par le fait que l'échange de données entre deux étapes ne se fait pas par l'intermédiaire d'un registre comme dans le cas d'un pipeline matériel, mais par l'intermédiaire de la communication par passage de messages (MPI). La figure 3.6 montre le temps d'exécution moyen des différentes parties des étapes du pipeline en pourcentage mesuré avec Score-P[114] et analysé avec ParaProf[115]. Nous constatons que pour une tâche i, il faut en moyenne 34,3% du temps pour la primitive MPI_Win_Wait, qui correspond à la phase de synchronisation avec la tâche précédente (tâche $i-1$) ; 27,04% du temps est dédié à la primitive MPI_Win_Complete correspondant à la synchronisation avec la tâche suivante (tâche $i+1$) et 5,68% pour la primitive MPI_Put correspondant à l'écriture des données dans la mémoire de la tâche suivante ;

4. https ://www.audacityteam.org/

Algorithme 1 : Structure générale du code C++ de chaque tâche i

```
// Initialisation MPI
.....
// Déclaration des objets
double A[FrameLen] ; double B[FrameLen] ;
.....
// Création de la fenêtre de communication
MPI_Win_create (B, FrameLen * sizeof(double), sizeof(double), MPI_INFO_NULL,
MPI_COMM_WORLD, &win) ;
.....
// Routine de la tâche
while true do
    // Étape 1 : exposer les objets mémoire pour la tâche i-1
    destrank = i-1 ;
    MPI_Group_incl(comm_group, 1, &destrank, &group) ; // Produit un groupe
      avec les membres énumérés.
    MPI_Win_post(group, 0, win) ; // Démarre une époque d'exposition RMA
    MPI_Win_wait(win) ; // Termine une époque d'exposition RMA commencée avec
      MPI_Win_post.
    .....
    // Étape 2 : Réaliser le traitement
    Treat(B, A) ; // Traiter les données en B et mettre le résultat en A
    .....
    // Étape 3 : envoyer les résultats à la tâche i+1 avec la primitive MPI_Put
    destrank = i+1 ;
    MPI_Group_incl(comm_group, 1, &destrank, &group) ;
    MPI_Win_start(group, 0, win) ; // Démarre une époque d'accès RMA de MPI
    MPI_Put(A, FrameLen, MPI_DOUBLE, destrank, 0, FrameLen, MPI_DOUBLE,
      win) ; // Insérer des données dans la fenêtre de mémoire d'un processus distant
    MPI_Win_complete(win) ;
end
```

Tableau 3.1: *Paramètres de configuration de notre implémentation Praat ACF*

Paramètre	**Valeur**
Pas de temps	*par défaut*
Valeur minimale du pitch	75 Hz
Nombre maximal de candidats	4
Très précis	*off*
Seuil de silence	0,3
Plafond du pitch	500 Hz
Longueur de la fenêtre	0,04 ms

Tableau 3.2: *La précision de notre PDA implémenté exécuté sur les enregistrements de paires minimales de mots*

Mot 1		**Mot 2**	
Mot - signification	**Précision en %**	**Mot - signification**	**Précision en %**
bàm - gronder	96.29	*bám* - s'en faire	100.00
bòg - extraire	95.69	*bóg* - empiler	99.00
kòb - frôler	100.00	*kób* - rejoindre	100.00
màan - récompense	96.74	*máan* - carrefour	98.86
minkùd - sacs	95.19	*minkúd* - nuages	98.96
sēg - diminuer	91.46	*ség* - découper	95.06
yēm - connaître	94.37	*yém* - serrer	98.02
zàm - bon goût	96.33	*zám* - raphia	93.26

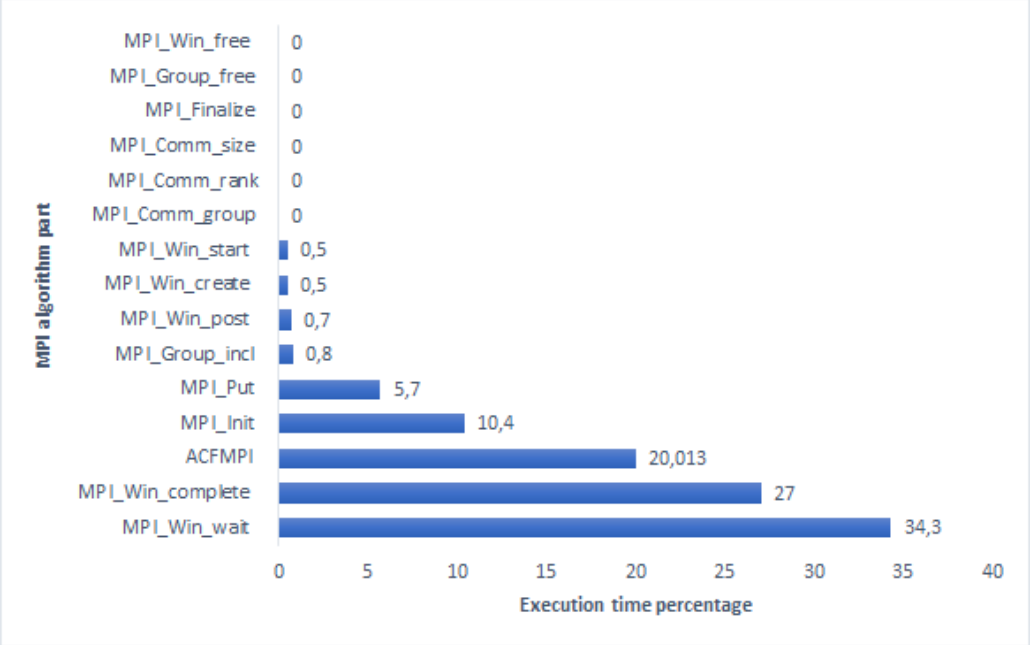

Figure 3.6: *Pourcentage du temps d'exécution moyen des différentes parties des tâches mesurées avec Score-P et analysées avec ParaProf.*

ce qui fait un total de 67,02% du temps au moins réservé à la communication entre les étages. De l'analyse faite ci-dessus, nous pouvons comprendre que cette faible accélération est due à un nombre élevé de communications entre les différentes étages du pipeline. Ainsi, l'intuition immédiate qui nous vient est qu'une implémentation matérielle de ce modèle MPI pourrait donner une meilleure accélération que celle obtenue dans cette première implémentation. C'est pourquoi pour plus d'efficacité de l'application sur un système embarqué, nous avons envisagé le déploiement de ce modèle parallèle sur une architecture reconfigurable, en particulier sur des FPGA qui offrent de nombreux avantages en termes de reconfiguration dynamique, de vitesse de traitement et de faible consommation en énergie. Cela consisterait à mettre en œuvre cette application sur architecture reconfigurable sous forme d'un système avec plusieurs éléments processeur appelé en anglais *MP-RSoCs (Multi-Processing Reconfigurable System on Chip)*. Dans la prochaine section nous présentons la technologie des MP-RSoCs, les avantages et intérêts qui nous motivent à les utiliser pour la mise en œuvre de notre application sur système embarqué.

3.4 Le MP-RSoC, un outil pour la conception d'applications sur systèmes embarqués

3.4.1 Notion de SoCs et MPSoCs

Un *SoC (System on Chip)* ou système sur puce, est un circuit intégré qui met en œuvre la plupart ou la totalité des fonctions d'un système électronique complet (pouvant comprendre de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interface, une logique spécialisée, des bus et d'autres fonctions numériques). L'architecture d'un SoC est généralement adaptée à l'application plutôt que d'être une puce à usage général. En fonction des applications, pour plus d'efficacité, il est souvent nécessaire de faire recours à plusieurs processeurs, d'où la naissance des *MPSoCs (Multi-Processing System on Chip)* qui sont des systèmes sur puce qui incorporent la plupart ou tous les composants nécessaires à une application et qui utilisent plusieurs processeurs comme composants du système.

3.4.2 Notion d'Architectures Reconfigurables

Pour la mise sur pied des SoCs, l'évolution de la technologie a permis d'avoir à disposition des *architectures reconfigurables* qui correspondent au fait qu'un dispositif en cours d'exploitation puisse être modifié pour effectuer une tâche différente. En d'autres termes, une architecture reconfigurable est un système sur puce capable de s'adapter aux besoins de l'application. La propriété d'être reconfigurable est appelée *reconfigurabilité*. La *reconfiguration* pour un système sur puce est le processus de changement de la structure d'un dispositif reconfigurable au cours de l'exécution. Un dispositif reconfigurable peut supporter la reconfiguration d'une partie de celui-ci, on parle alors de *reconfigurabilité partielle*. Et lorsqu'il est possible de modifier la fonctionnalité d'un dispositif reconfigurable pendant qu'une partie du dispositif est occupée à exécuter une tâche quelconque, on parle de *reconfigurabilité dynamique*.

3.4.3 Les FPGAs : technologie de mise en œuvre d'architectures reconfigurables

L'une des technologies d'architectures reconfigurables les plus connues et utilisées est la technologie des FPGA (Field Programmable Gate Array). Les FPGA sont des composants logiques de haute densité et reconfigurables qui permettent, après configuration, de mettre en œuvre des circuits numériques de traitement pouvant calculer ou générer des informations pour tout type d'application. Leur structure régulière (voir figure 3.7) en fait des éléments très performants pour les traitements bas niveaux réguliers. Les principaux éléments composant les FPGAs sont :

- **Les CLB (Configurable Logic Block) :** Ce sont des cellules constituées d'éléments logiques programmables où l'on trouve des bascules, des LUTs (Look-Up Table), des multiplexeurs et des portes logiques disposées sous forme matricielle, comme le montre la figure 3.7. Chaque cellule est identique aux autres et peut être reliée à ses voisines par le biais de bus d'interconnexion. Ces cellules peuvent être utilisées pour créer des fonctions logiques complexes mais aussi comme éléments de stockage de variables.
- **Les IOBs (Input Output Block) :** Ce sont des cellules d'entrées/sorties qui permettent d'interfacer le FPGA avec l'environnement extérieur.
- **Les ressources d'interconnexion des cellules :** Pour pouvoir réaliser des fonctions complexes à partir des cellules de base que représentent les CLBs, il est nécessaire de disposer de ressources d'interconnexion entre ces différentes cellules.

Depuis quelques générations de FPGAs, les fabricants ont ajouté aux ressources classiques, de nombreux éléments tels que :

- **Les blocs mémoire :** Les FPGAs peuvent se comporter comme un espace de stockage de variables, cet espace de stockage est réparti dans tout le FPGA. Ce mode de stockage est

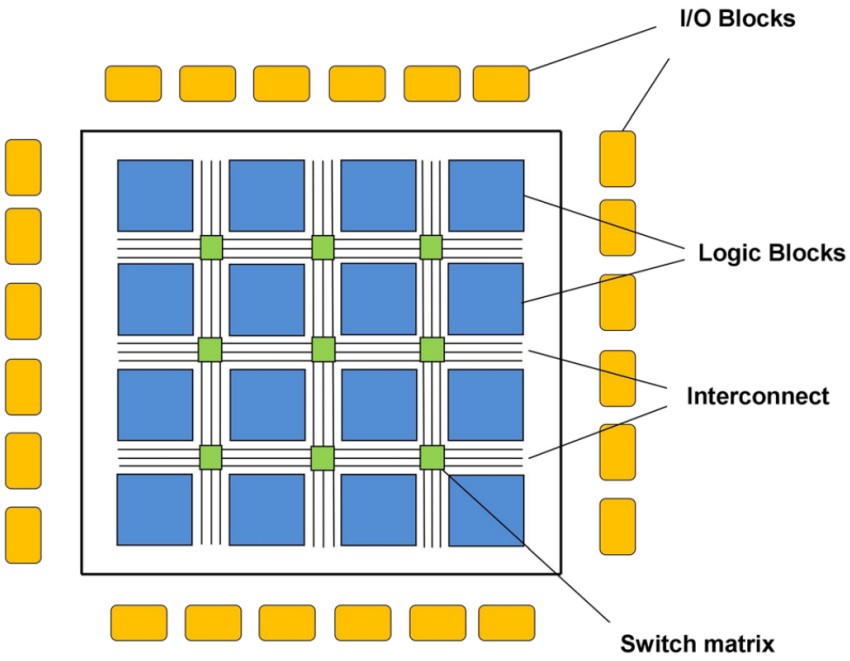

Figure 3.7: *Structure de base d'un FPGA[116]*

tout de même limité en termes d'espace disponible. C'est pourquoi, pour pouvoir stocker une quantité importante d'informations sans avoir à accéder à des mémoires externes, certains fabricants de FPGAs ont introduit des blocs mémoires à l'intérieur des FPGAs.

— **Les multiplieurs :** La logique présente dans les FPGAs permet de réaliser toutes sortes d'opérations arithmétiques (additions, multiplications, ...). Ces opérations, et en particulier les multiplications sont très coûteuses en termes de ressources logiques utilisées, d'où l'intérêt de disposer dans les FPGAs de multiplieurs câblés. De plus, le temps de calcul pour ces opérations est alors optimisé.

— **Les blocs processeur :** avec l'apparition des très grands FPGAs, il est possible d'intégrer des algorithmes complexes sur une seule puce. Certains FPGAs disposent aujourd'hui de cœurs de processeurs.

3.4.4 MP-RSoC

La croissance de la capacité des FPGA permet aux concepteurs de mettre en œuvre un système multiprocesseur complet dans un seul FPGA. Les FPGA sont équipés de blocs de mémoire sur puce, de périphériques et de circuits d'interconnexion en dur ou reconfigurables. La reconfigurabilité dynamique est l'un des points forts des systèmes multiprocesseurs basés sur les FPGA. Cette caractéristique permet aux systèmes multiprocesseurs d'être optimisés en termes de surface consommée sur le FPGA, et permet de gagner en flexibilité dans le système conçu.

Les caractéristiques ci-avant présentées à propos des MP-RSoCs, en font une solution technologique qui présente de nombreux intérêts et atouts que nous présentons dans la prochaine section.

3.5 Intérêts et atouts des MP-RSoC comme espace de solution

L'intérêt accordé aux MP-RSoCs se fonde sur ses nombreux avantages, et la multiplicité des domaines dans lesquels ils sont sollicités.

3.5.1 Avantages des MP-RSoCs

Les MP-RSOCs présentent un certain nombre d'avantages qui justifient leur attractivité comme solution pour des applications embarquées. Le tableau 3.3 présente une comparaison entre les plateformes de type microprocesseur, FPGA, ASIC et GPU[117]. Parmi ces avantages on distingue :

Flexibilité et reconfiguration

Le nombre de processeurs softcore pouvant être inclus n'est limité que par la capacité du FPGA. De plus, il est possible de configurer chaque processeur indépendamment en ajoutant du cache, des modules FPU, etc.

Réduction du délai de mise sur le marché

Le processus de conception ne comprend pas la fabrication du circuit intégré, ce qui permet de réduire considérablement le temps de conception.

Tableau 3.3: *Comparaison des architectures de type microprocesseur, FPGA, ASIC et GPU en fonction de différents paramètres[117].*

Paramètres	**Microprocesseur**	**FPGA**	**ASIC**	**GPU**
Flexibilité pendant le développement	Moyenne	Élevée	Très élevée	Faible
Flexibilité après le développement	Élevée	Élevée	Faible	Élevée
Parallélisme	Faible	Élevé	Élevé	Moyen
Performance	Faible	Moyenne	Élevée	moyenne
Consommation en énergie	Élevée	Moyenne	Faible	Élevée
Coût	Faible	Moyen	Élevé	Faible
Coût de mise en œuvre	Aucun	Aucun	Élevé	Aucun
Coût unitaire	Moyen	Élevé	Faible	Elevé
Délai de mise sur le marché	Faible	Moyen	Élevé	Moyen

3.5.2 Domaines d'application des MP-RSoCs

Les MP-RSoCs sont exploités dans plusieurs domaines dans le traitement de l'information. Dans cette section nous ne présenterons que quelques champs d'applications.

Systèmes de vision artificielle

Il existe de plus en plus de dispositifs dotés d'un système de vision artificielle. Les caméras de vidéosurveillance, les robots, voitures autonomes, etc ; en sont des exemples. Beaucoup de ces dispositifs ont besoin d'un système pour connaître leur position, reconnaître les objets dans leur environnement[118], reconnaître les visages des personnes[119], et être capable d'agir et d'interagir avec elles de manière appropriée.

Les systèmes d'imagerie médicale

Les FPGA sont de plus en plus fréquemment utilisés pour le traitement d'images biomédicales[120]. Ces systèmes de vision médicaux exigent de plus en plus une résolution et une capacité de traitement accrues, et même beaucoup doivent pouvoir être développés en temps

réel. Les avantages offerts par les FPGA et le traitement parallèle s'adaptent très bien à ces besoins.

Chiffrement et décryptage, cryptographie

Le parallélisme de calcul massif, la possibilité de configurer les unités de calcul en fonction de la largeur de bit nécessaire et la faible latence sont les principales raisons pour lesquelles les FPGA sont utilisés dans le domaine du chiffrement/déchiffrement[121] et de la cryptographie post-quantique[122].

Radioastronomie

La radioastronomie est la science qui est chargée d'étudier les phénomènes qui se produisent dans l'espace en captant le rayonnement électromagnétique de celui-ci. Comme les applications précédentes, elle nécessite le traitement d'une grande quantité d'informations dans lequel le FPGA peut apporter tout son potentiel[123].

Reconnaissance de la parole

La reconnaissance de la personne qui parle est une technique utilisée dans la sécurité, les systèmes de recherche d'informations, etc. et l'on s'attend à ce que son champ d'application augmente à l'avenir. Dans ce contexte, le FPGA est très efficace lorsqu'il s'agit de comparer la voix d'une personne avec des modèles précédemment stockés[124]. D'autres systèmes permettent d'implémenter des systèmes domotiques contrôlés par commandes vocales[125] et aussi des systèmes de communication multilingues[126].

Aéronautique et défense

Outre celles mentionnées précédemment, il existe une multitude d'applications aéronautiques et de défense[127] qui utilisent les FPGA en raison des bonnes caractéristiques qu'ils offrent.

Centre de données / Cloud

L'internet des objets (IoT), et le big data en général, génèrent une croissance exponentielle des données acquises et traitées, qui, avec l'analyse computationnelle de celles-ci à travers des techniques d'apprentissage profond de multiples opérations parallèles / simultanées, conduisent à une forte demande de capacité de calcul à faible latence, flexible et sécurisée, qui ne peut pas être résolue en ajoutant plus de serveurs, en raison de l'augmentation folle du coût en espace, consommation et argent. Dans ce contexte, les portes du monde des centres de données s'ouvrent massivement aux FPGA[128], en raison de leur capacité d'accélération du calcul, de leur flexibilité de configuration et de la sécurité que le matériel garantit contre le logiciel.

A la lumière des différents champs d'applications présentés ci-avant, nous comprenons que les FPGA ont une forte présence dans des applications complexes qui intègrent souvent beaucoup de traitements et sont parfois soumis à des contraintes de temps réel. Ainsi, dans ces domaines, la parallélisation possède une place de choix pour permettre aux concepteurs d'applications de répondre à la contrainte de temps réel ou du moins de délivrer une latence réduite pour des applications moins exigeantes. Les MP-RSoC constitués de plusieurs éléments processeurs susceptibles de s'exécuter simultanément viennent répondre à ce besoin de parallélisation d'applications sur puce. Dès lors, un MP-RSoC est donc considéré comme un système parallèle et il est nécessaire d'offrir au développeur les instruments qui lui permettraient de concevoir ce type de système aussi aisément que sur un environnement logiciel. En fonction des contraintes et exigences de l'application, les tâches de traitement peuvent être déployées sur environnement matériel et/ou

environnement logiciel. Dans le cas de l'usage des deux environnements, il revient à l'ingénieur de réaliser une conception qui prend en compte de façon conjointe les deux réalités ; on parle alors de *codesign*. La prochaine section présente la notion de codesign avec les MP-RSoC.

3.6 MP-RSoCs et Co-design/Co-conception matérielle-logicielle

3.6.1 Définition

Les systèmes embarqués se composent généralement de composants matériels et logiciels profondément intégrés. Dans certains systèmes, la partie matérielle est fixe, mais les exigences croissantes en matière de performance nécessitent de plus en plus que le matériel réponde à des exigences très particulières et qu'il soit précisément adapté au logiciel embarqué. Par conséquent, il n'est souvent plus suffisant d'utiliser des composants matériels préfabriqués. Lorsqu'un matériel dédié doit être développé pour un système embarqué, le développement du logiciel embarqué dépend de l'avancement du développement du matériel.

Le Codesign matériel/logiciel désigne les méthodologies de conception des systèmes électroniques qui exploitent les atouts du matériel et du logiciel. En général, la fonctionnalité de l'application est divisée en composants logiciels qui s'exécutent sur les cœurs de processeur et en composants matériels qui sont utilisés pour accélérer certaines parties de l'application ou pour fournir des interfaces à l'environnement.

3.6.2 Flot de Co-conception matérielle-logicielle

La conception commence par un ensemble de spécifications et de besoins sur les propriétés fonctionnelles et non fonctionnelles (sécurité, temps de réponse, précision, etc.) du système cible. La première étape consiste à déterminer l'algorithme approprié pour répondre à la *spécification fonctionnelle* si elle n'est pas donnée. Ensuite, les architectes système déterminent généralement une architecture matérielle qui répondrait aux exigences fonctionnelles tout en optimisant les objectifs de conception tels que la minimisation des coûts et/ou de l'énergie. La décision est généralement prise sur la base des informations de *profilage* des algorithmes à mettre en œuvre et de leur complexité de calcul, ainsi que des exigences et contraintes en matière de ressources. Elle est généralement influencée par l'expérience des ingénieurs sur la conception de systèmes similaires. La manière de *partitionner* (répartir les taches selon qu'elles seront implémentées du coté matériel ou logiciel) la fonctionnalité en composants logiciels et matériels est également déterminée manuellement à cette étape.

Après ce partitionnement Matériel/Logiciel pour l'architecture matérielle déterminée, on passe à la conception et implémentation des composants matériels et logiciels. Une fois les deux types de composants construits, l'ensemble du logiciel est intégré en exploitant une interface qui sert de canal de communication entre les deux parties. Ensuite, le système est soumis à des tests et à des vérifications visant à déterminer s'il satisfait aux exigences fonctionnelles et non fonctionnelles. S'il s'avère que l'architecture matérielle et les décisions de partitionnement matériel/logiciel doivent être modifiées, il est nécessaire de revenir au point de départ de la boucle de conception pour itérer le processus de conception.

Dans la conception des MP-RSoC, les composants logiciels sont déployés sur des processeurs qui peuvent être soit implantés sur la puce soit instanciés (Softcore) sur la surface reconfigurable. Les composants logiciels ainsi déployés sont appelés *taches logicielles*. En outre les composants matériels sont soit des IP matérielles réutilisées, soit des composants spécialisés conçus dans le cadre de l'application. Ces composants sont donc appelés de *taches matérielles*.

A la lumière de tout ceci, un MP-RSoC est donc, non seulement un système parallèle qui doit prendre en compte les modèles et standards manipulés pour l'efficacité dans la conception d'applications, mais aussi un système dont la conception doit prendre en compte l'approche de

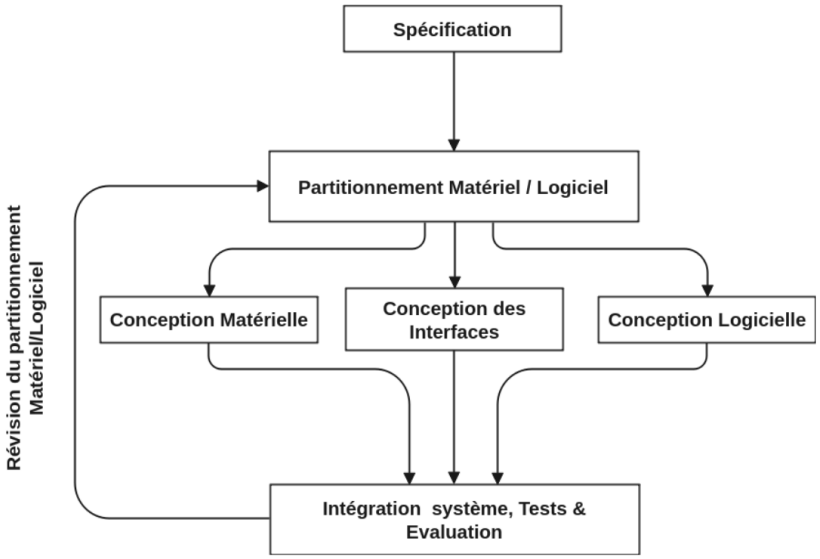

Figure 3.8: *Flot de Co-conception Matériel/Logiciel*

co-conception matérielle - logicielle car ils peuvent intégrer à la fois des CPU et des éléments de traitement de type accélérateur matériels. Nous avons vu à travers les différents avantages que les MP-RSoCs constituent un espace de solution de choix pour la conception d'applications embarquées sur puce ; cependant nous verrons dans la suite que la conception d'une application sur MP-RSoC peut se révéler être une tâche complexe pour les ingénieurs. Il est dès lors capital, de proposer aux ingénieurs des approches et outils qui leur permettent de concevoir des applications sur MP-RSoC en associant à la fois efficience en termes de performance (latence, surface et consommation en énergie) et facilité de conception. Ceci évoque donc le problème de productivité de conception. Dans la prochaine section, nous faisons une clarification de la notion de productivité de la conception des applications sur MP-RSoC.

3.7 Productivité de conception des applications parallèles sur MP-RSoC

Sous l'influence de la loi de Moore[129], qui stipule que le nombre de transistors sur les circuits intégrés double tous les deux ans, l'industrie des semi-conducteurs a connu une évolution fulgurante. Cela a offert aux concepteurs la possibilité de créer des systèmes plus rapides, plus grands et plus complexes en utilisant moins de surface. Cependant, l'un des principaux défis de la conception de systèmes électroniques est actuellement l'écart croissant en matière de productivité de la conception, comme le souligne l'ITRS (International Technology Roadmap for Semiconductors)[130] devenue IRDS (International Roadmap for Devices and Systems). L'écart de productivité de la conception fait référence à une augmentation plus rapide de la complexité des systèmes par rapport à la productivité des concepteurs de systèmes[131]. La figure 3.9 illustre ce phénomène ; en effet, la productivité de la conception des architectures matérielles a été améliorée ces dernières années en remplissant le silicium de composants multicœurs et de mémoire, fournissant ainsi des fonctionnalités uniquement avec des logiciels supplémentaires ; d'autre part, la productivité, en particulier pour les logiciels requis pour le matériel, ne double que tous les 5 ans. La flèche rouge résume le nouvel écart de conception incluant à la fois le matériel et le logiciel.

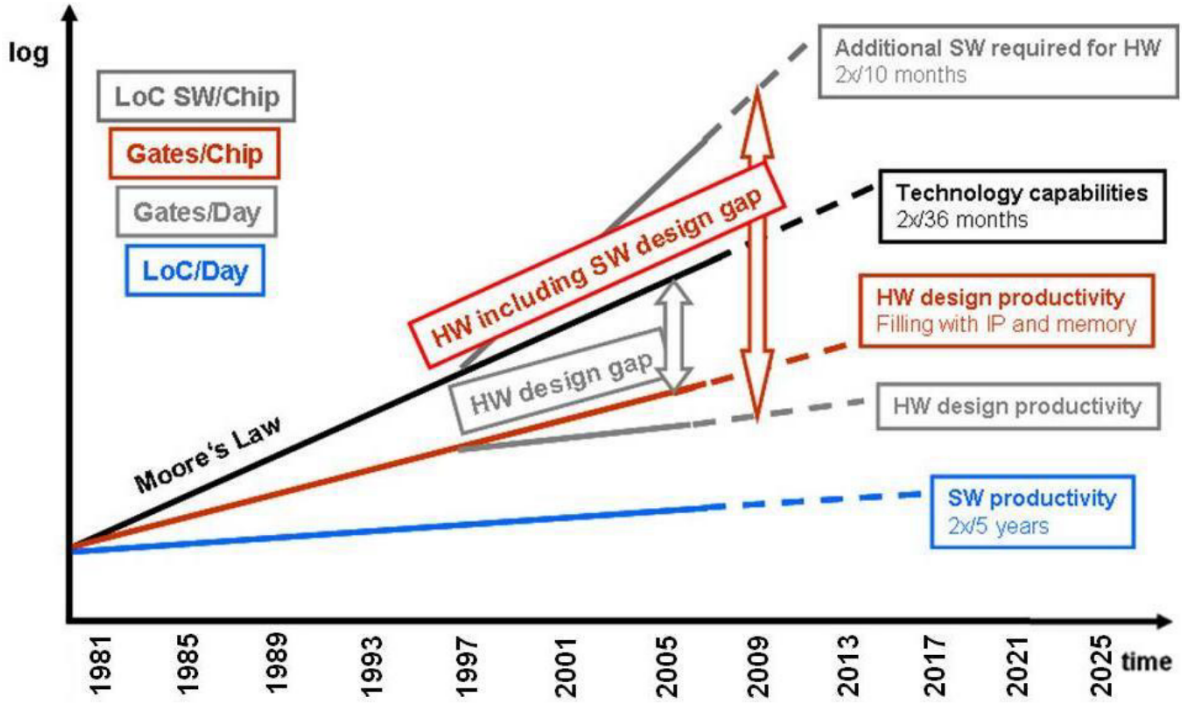

FIGURE 3.9: *Ecarts de productivité de la conception matérielle et logicielle en fonction du temps[130]*

3.7.1 Notion de productivité de conception

Maxime Pelcat et al.[132], indiquent que *"la productivité de la conception (Design Productivity/DP) est un compromis entre les efforts déployés pour construire un système et la qualité du système résultant de ces efforts de conception"*. Ils proposent une représentation générique de la productivité de la conception d'un système (voir figure 3.10) et stipulent que la productivité de la conception peut être augmentée par deux facteurs :

— lorsque l'effort de conception est réduit. Cela correspond à l'augmentation de l'efficacité de la conception à qualité égale du système,

— lorsque, pour un effort fixe, la qualité de l'implémentation est augmentée.

D'après Maxime Pelcat et al., l'effort de conception peut être quantifié par des mesures de coûts d'ingénierie non récurrents (Non-Recurring Engineering/NRE) tels que le temps de conception, le temps de test et de validation, le nombre de lignes de code à écrire, les dépenses budgétaires NRE, etc. La qualité de l'implémentation peut être mesurée par des coûts de propriété non fonctionnelle (Non-Functional Property/NFP) tels que la consommation d'énergie du système, sa latence, son débit, le coût de production, la surface de silicium, etc.

3.7.2 Augmenter la productivité de conception des applications parallèles sur MP-RSoC

La conception d'applications parallèles sur MP-RSoC induit de nombreuses problématiques auxquelles doivent faire face les ingénieurs. Taho Dorta et al.[133] ont mis en avant les problématiques les plus rencontrées et on distingue :

— la quantité limitée de ressources logiques, en particulier la quantité de mémoire sur puce ;

— la cohérence du cache est un autre point critique dans la conception des MP-RSoC. Normalement, les simples softcores utilisés dans de tels systèmes ne supportent aucun mécanisme pour garantir la cohérence du cache ;

— les communications efficaces sur la puce entre les différents cœurs ou unités de traitement sont nécessaires dans les MP-RSoC ;

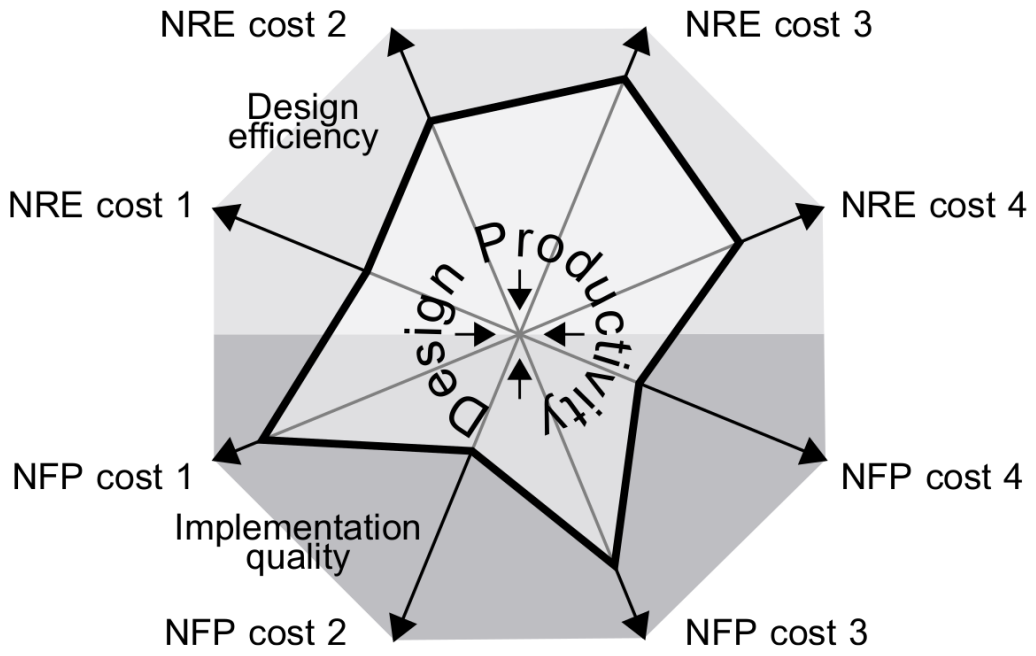

Figure 3.10: *Représentation de la productivité de la conception comme une combinaison des coûts d'ingénierie non récurrents (NRE) et des coûts des propriétés non fonctionnelles (NFP) du système.[132]*

— l'efficacité de conception des applications parallèles. Il est nécessaire de développer des techniques et outils pour la programmation parallèle sur MP-RSoC.

Toutes ces problématiques sont les principaux éléments qui constituent un obstacle pour une meilleure productivité des ingénieurs. Pour la réduction des coûts et du temps de mise sur le marché des produits et applications réalisées à l'aide des MP-RSoC, il est primordial de proposer aux ingénieurs, des outils et approches leur permettant d'accroître leur productivité.

Nous entendons ici par augmentation de la productivité de conception des applications parallèles sous MP-RSoC, le fait de proposer : des approches, méthodes, modèles et outils qui permettraient à un ingénieur de réduire l'effort de conception d'une application parallèle (en termes de temps de conception, temps de test et de validation, nombre de lignes de code à écrire, dépenses budgétaires NRE, etc.) et/ou d'augmenter étant donné un niveau d'effort de conception, la qualité de l'implémentation du système obtenu (en termes de consommation en énergie du système, latence, débit, coût de production, surface/quantité de ressources logiques consommées, etc.).

Dans le cadre de ces travaux, nous adressons le problème d'augmentation de la productivité de conception des applications parallèles sur MP-RSoC. Plus précisément, nous nous intéressons aux applications parallèles sur MP-RSoC structurées sous forme d'architecture à mémoire distribuée et qui exploitent le standard de communication MPI-2 RMA.

La question qui est traitée ici est celle de savoir, comment faciliter la conception et la mise en œuvre d'applications sur ce type de plates-formes. Plus spécifiquement, comment faciliter la conception d'applications parallèles à mémoire distribuées pour MP-RSoC par une approche de génération automatique de plateforme ; quelle méthodologie adopter ; et dans quelle mesure une telle approche peut intégrer la co-conception matérielle-logicielle.

Ces questions ont déjà été abordées par d'autres auteurs ; dans la prochaine section, nous parcourons l'état de l'art sur l'augmentation de la productivité de conception d'applications parallèles sur MP-RSoC.

3.8 État de l'art sur les approches d'augmentation de la productivité de conception d'applications parallèles sur MP-RSoC

Il existe plusieurs approches pour augmenter la productivité de conception des applications parallèles sur MP-RSoC. Dans ce travail nous avons mis un accent sur les approches qui consistent à réduire l'effort de conception fourni par l'ingénieur, en d'autres termes celles qui augmentent son efficacité de conception. Ainsi dans cette section, nous vous présentons les différentes approches que nous avons pû identifier en les étayant par des exemples d'outils rencontrés dans l'état de l'art.

3.8.1 Les approches basées sur les outils CAO (CAD tools)

Les outils d'automatisation de la conception électronique en anglais *Electronic Design Automation* (EDA) encore appelés outils de conception assistée par ordinateur en anglais *Computer Aided Design*(CAD) sont un groupe d'outils logiciels permettant de concevoir des systèmes électroniques tels que des circuits intégrés (ASIC), des cartes de circuits imprimés (PCB), ou du matériel reconfigurable comme les FPGA, etc[134]. En général, ces outils fonctionnent dans un flot de conception que les concepteurs de matériel et de systèmes utilisent pour concevoir et analyser le comportement du système entier.

Traditionnellement, le flot EDA sur FPGA commence par le niveau RTL (*Register Transfer Level*) qui n'est rien d'autre qu'une abstraction d'un circuit numérique composé d'éléments combinatoires et séquentiels. Le RTL est spécifié dans un langage de description du matériel en anglais *HDL (Hardware Description Language)*, comme Verilog ou VHDL.

Il est aussi possible de spécifier tout ou une partie d'une description RTL par l'utilisation de schémas, l'utilisation de blocs de propriété intellectuelle (IP), ou l'utilisation de langages électroniques de niveau système (*ESL : Electronic System Level*). La figure 3.11 montre un flot de conception typique. Les fournisseurs de FPGA regroupent les outils EDA dans des *suites logicielles de conception*, et il en existe plusieurs. Nous présentons dans la suite les solutions les plus connues et les plus utilisées dans l'état de l'art.

Quartus Prime

Quartus Prime (par Intel)[135] : il permet de décrire facilement les conceptions, de les traiter rapidement, de programmer facilement les dispositifs et de les intégrer à d'autres outils EDA standard. Le logiciel Intel Quartus Prime offre une gamme complète de fonctions à chaque phase du flot pour raccourcir le cycle de conception et obtenir les meilleures performances. Ce logiciel fournit un environnement de conception complet pour les conceptions FPGA et SoC de type Intel Agilex, Intel Stratix 10, Intel Arria 10 et Intel Cyclone 10 GX les plus avancées.

Vivado Design Suite

Vivado Design Suite[136] (par Xilinx) [5] : est conçue pour augmenter la productivité globale de la conception, de l'intégration et de la mise en œuvre de systèmes utilisant des dispositifs UltraScale, 7 series et Versal, Zynq UltraScale+ MPSoCs et, Zynq-7000 SoCs. Avec cet outil, il est possible d'accélérer la mise en œuvre de la conception grâce à des outils de placement et de routage qui optimisent de manière analytique des métriques de conception multiples et simultanées, telles que le timing, la congestion, la longueur totale des fils, l'utilisation et la puissance. De plus, il propose un flot qui s'articule autour de la philosophie de conception, de réutilisation et d'intégration des IP. Il permet de créer des modules IP, mais aussi de les packager pour une utilisation répétée et d'intégrer facilement des applications tierces . La propriété intellectuelle personnalisée

5. https ://www.xilinx.com/support/university/vivado.html

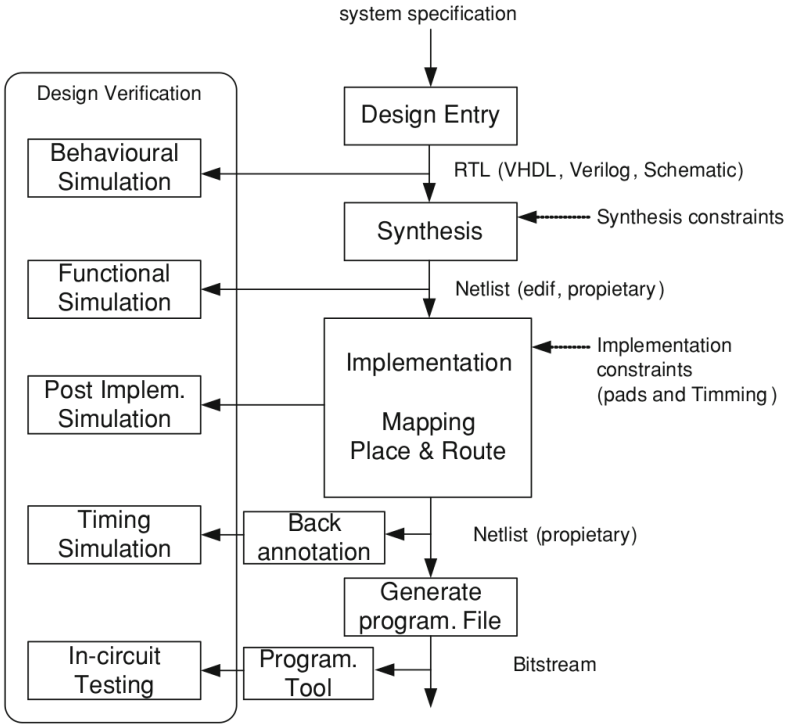

Figure 3.11: *Un flot de conception typique d'un EDA pour la conception de FPGA[134]*

est packagée pour être utilisée, conformément à IP-XACT[137], avant d'être disponible dans le catalogue de propriété intellectuelle.

Libero SoC Design Suite

Libero SoC Design Suite (par Microsemi)[138] : offre une productivité élevée grâce à ses outils de développement complets, faciles à apprendre et à adopter pour la conception avec les familles PolarFire, IGLOO2, SmartFusion2, RTG4, SmartFusion, IGLOO, ProASIC3 et Fusion[139] de Microsemi. La suite intègre la synthèse standard Synopsys Synplify Pro et la simulation Siemens ModelSim.

Lattice Diamond

Lattice Diamond (par Lattice Semiconductor)[140] est un logiciel de conception qui offre des outils de conception et d'implémentation pour les architectures FPGA de Lattice sensibles aux coûts et à faible consommation. Diamond est le remplaçant de nouvelle génération de ispLEVER[141], avec une exploration de la conception, une facilité d'utilisation, un flot de conception amélioré et de nombreuses autres améliorations. Cette combinaison de fonctionnalités nouvelles et améliorées permet aux utilisateurs de réaliser des conceptions plus rapidement, plus facilement et avec de meilleurs résultats qu'auparavant.

Tous ces outils permettent de concevoir des systèmes de type MP-RSoC au sein des FPGA, mais ils ont une limite liée au fait que la description de l'application nécessite de l'expertise dans la description des systèmes en langage matériel. En outre pour des besoins de validation, la simulation et la visualisation des signaux du système sont souvent très fastidieuses ; plus l'application est complexe, plus il y a d'objets et plus la simulation et la visualisation des signaux sont coûteuses en temps.

3.8.2 Les approches basées sur la ré-utilisabilité des IPs

Les IPs jouent un rôle important dans l'augmentation de la productivité de la conception en permettant aux concepteurs et aux outils de conception de réutiliser des composants préconçus appelés propriété intellectuelle ou IP. Les bibliothèques fournissant différentes alternatives d'implémentation facilitent encore plus la sélection du circuit optimal à partir d'un choix de circuits fonctionnellement équivalents, qui offrent différents compromis entre performance, surface et latence.

Les types de composants IP

Il existe trois types d'IP standard : *soft*, *firm* ou *hard*.

- **Les cœurs d'IP soft**, sont des implémentations dans un langage HDL sans optimisation poussée pour l'architecture cible. Ils ont généralement des performances moyennes en termes d'utilisation des ressources et de temps d'exécution.
- **Les cœurs d'IP firm**, sont également des implémentations HDL mais ont été optimisés pour une technologie FPGA cible, ils autorisent des optimisations supplémentaires lors du placement et du routage sur le support physique.
- **Les cœurs d'IP hard**, sont des IP optimisés pour un composant FPGA spécifique.

Les fournisseurs des IPs

Il existe de nombreux fournisseurs d'IPs à travers le monde. Par catégorie on distingue :

- **Les sociétés d'études et de conception** : Sociétés sans fonderie dont le profit vient des droits sur les licences, comme *DSP Group* (IP pour les télécoms) ou *ARM* (processeurs RISCs) avec des IPs soft, de simulation et synthèse.
- **Les sociétés de semi-conducteurs** qui peuvent fournir des IPs hard en plus des Ips soft (Motorola, TI, Lucent (actuellement Alcatel-Lucent [6]), Xilinx, Altera (Intel), LSI LOGIC (Avago), STM. . .)
- **Les fournisseurs d'outils de CAO** qui fournissent des IPs soft uniquement (Mentor Graphics (Siemens), Cadence [7], Synopsys [8]. . .)
- **Fournisseurs de blocs d'IP open source** : Les laboratoires des universités et les communautés de partage des conceptions *gateware* [9] telles que OpenCores [10], LibreCores [11], etc.

Malgré le potentiel de la réutilisation pour augmenter la productivité, cette approche présente des limites[142]. Dans la plupart des environnements et méthodologies de conception actuels, pour réussir à réutiliser l'IP, le concepteur doit (1) trouver et sélectionner manuellement l'IP, (2) comprendre les détails de sa mise en œuvre et (3) comprendre l'interface et le protocole de synchronisation utilisés afin de l'intégrer dans un système global. Les circuits de contrôle et d'interface doivent souvent être générés manuellement. Il s'agit d'un processus complexe et fastidieux qui doit être réalisé très rapidement afin que la réutilisation soit viable.

En outre, les IPs proviennent souvent de nombreuses sources et se présentent sous de nombreux formats, ce qui fait de la réutilisation une perspective peu attrayante. Or, pour qu'un processus de réutilisation soit réalisable, l'ensemble du processus ne doit pas nécessiter plus de 30 % de l'effort requis pour créer la même IP à partir de zéro[143].

6. https ://www.al-enterprise.com/en
7. https ://www.cadence.com/en_US/home.html
8. https ://www.synopsys.com/
9. *Gateware* comprend la description (du comportement, de la structure et/ou des connexions) de portes logiques numériques, une abstraction de haut niveau de celles-ci et/ou leur mise en œuvre dans des dispositifs logiques (re)configurables (tels que les FPGA et les ASIC).
10. https ://opencores.org/
11. https ://www.librecores.org/

Dans le contexte de la conception des applications parallèles, cette approche peut exploser en termes de complexité car le développeur doit prendre en charge différentes problématiques à savoir : les tâches de traitement, l'infrastructure de communication, le modèle de programmation parallèle, la synchronisation, etc. Une approche basée uniquement sur la ré-utilisabilité des IPs devient donc peu attrayante.

3.8.3 Approches basées sur l'utilisation des générateurs de circuit

Les FPGA sont utilisés pour intégrer des accélérateurs matériel qui peuvent intégrer des opérations arithmétiques optimisés. Bien qu'il existe une vaste littérature sur les circuits arithmétiques optimisés, le choix de l'implémentation optimale est un processus long et fastidieux, qui prend un temps précieux aux concepteurs.

Une catégorie de solution pour améliorer la productivité dans ce domaine consiste à concevoir des *générateurs de cœurs arithmétiques*. Cette catégorie d'outil consiste à générer des opérateurs arithmétiques (en langage HDL) en offrant au développeur la possibilité de paramétrer chaque opérateur arithmétique le plus finement possible en termes de taille (nombre de bits en entrée et en sortie), d'algorithme utilisé, de paramètres de l'opérateur lui-même (par exemple la constante pour un multiplieur par une constante), de FPGA cible avec ses spécificités architecturales, de fréquence à laquelle on souhaite faire fonctionner l'opérateur, etc. De nombreuses solutions sont proposées dans ce domaine, ici nous n'allons en présenter que quelques-unes.

HMS Multiplier

HMS Multiplier Generator [12] de M. Själander et al.[144]. Il s'agit d'un générateur de multiplicateurs capable de générer plusieurs types de multiplieurs. Il existe actuellement cinq types différents de multiplieurs qui peuvent être générés : Radix2 non signé, Baugh-Wooley, Booth modifié, Baugh-Wooley à double précision, Booth modifié double précision.

FloPoCo (Floating-Point Cores)

FloPoCo (Floating-Point Cores)[145] est un framework C++ open-source pour générer des chemins de données arithmétiques. FloPoCo fournit une interface en ligne de commande qui saisit les spécifications des opérateurs, et produit du VHDL synthétisable. Chaque générateur de chemin de données dans FloPoCo est une classe C++. Le choix d'un générateur de VHDL écrit en C++ permet d'inclure dans celui-ci des optimisations de l'architecture générée de plus en plus poussées tels que l'optimisation du pipeline. Il propose actuellement une vingtaine d'opérateurs, des plus simples comme les décaleurs ou les additionneurs d'entiers aux plus complexes comme les opérateurs calculant une exponentielle ou un logarithme en virgule flottante. Il constitue même un framework de recherche sur les cœurs arithmétiques matériels, incluant entre autres : l'arithmétique LNS[146], les générateurs de nombres aléatoires [147], les fonctions élémentaires[148], les opérateurs spécialisés tels que la multiplication et la division constantes[149], diverses techniques d'optimisation spécifiques aux FPGA [150], et plus récemment les transformées et les filtres de traitement du signal[151]. La figure 3.12 montre l'interface typique d'un générateur d'opérateur implémenté dans FloPoCo. Elle distingue la spécification fonctionnelle, qui décrit l'opération à implémenter, de la spécification de sa performance.

RTLGen

G. Rocha et al. proposent RTLGen [153], un framework pour à la fois généraliser et explorer différentes compositions de multiplieurs binaires parallèles afin de rechercher les choix de compromis dans des axes multiples. Il s'agit d'un framework automatique de génération, de vérification,

12. https ://www.sjalander.com/research/multiplier/

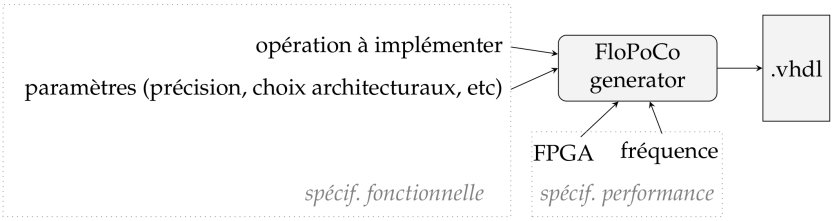

Figure 3.12: *L'interface typique d'un générateur d'opérateurs dans FloPoCo [152]*

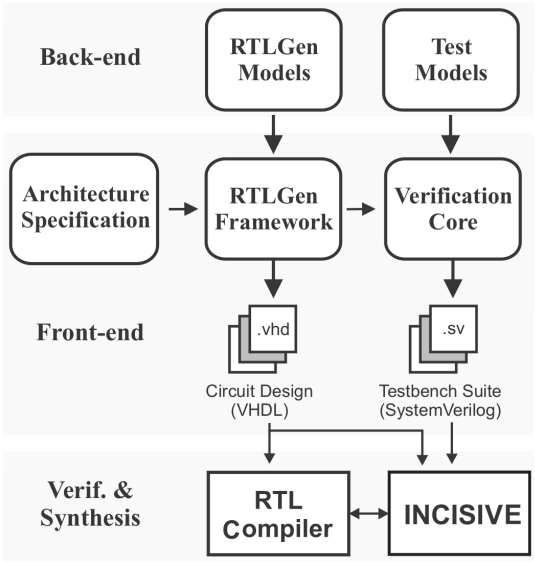

Figure 3.13: *Flot de génération de circuit avec RTLGen*

de synthèse et d'analyse de la qualité des résultats (QoR) des cœurs arithmétiques. La principale contribution de ce travail est de proposer un environnement permettant d'explorer le compromis pour différents paramètres : chemins temporels cibles, largeurs de bits, codeurs de génération de produits partiels, arbres de compression, topologies d'additionneurs à propagation de retenue et circuits d'additionneurs finaux, afin de vérifier le compromis d'analyse QoR pour la mise en œuvre de multiplicateurs binaires parallèles câblés. La figure 3.13 illustre le flot de génération pour un ensemble donné de paramètres d'architecture en utilisant le cadre VHDLGen. Le moteur back-end fournit les modules de construction de base qui permettent l'intégration de nouveaux composants tels que les codeurs de produits partiels, les arbres de compression et les additionneurs à propagation de retenue. Une fois que les modules de construction de base sont décrits à l'aide du framework, le moteur front-end fournit une vue simplifiée de ces modules, permettant la génération de diverses architectures arithmétiques.

Les générateurs de circuits constituent une solution très efficace sur le ratio temps de conception - qualité des résultats ; dans la mesure où elle permet la conception rapide des cœurs de traitement très optimisés en un temps limité. Cependant une telle solution nécessite une bonne connaissance des algorithmes manipulés par ces cœurs de traitement pour mieux définir les paramètres exploités par le générateur. Par rapport à la conception des applications parallèles, cette catégorie de solution laisse totalement les taches de d'intégration des composants, la construction de l'infrastructure de d'interconnexion, la communication, et la synchronisation à la charge du développeur.

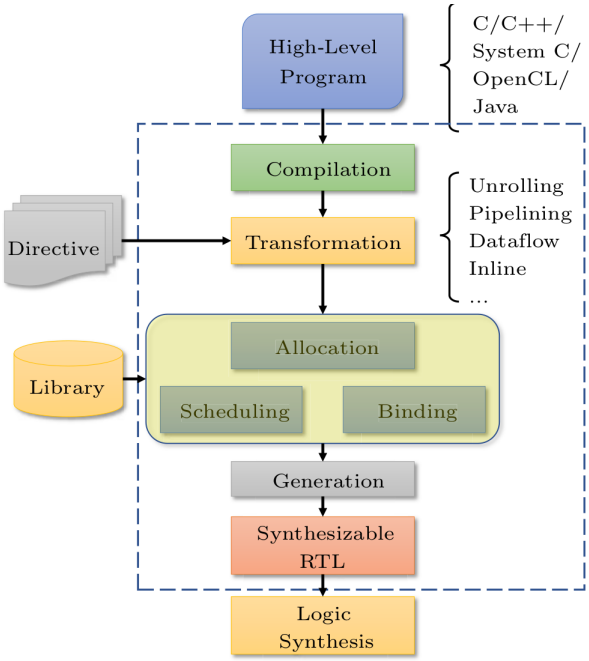

Figure 3.14: *Flot typique de synthèse de haut niveau (HLS).*

3.8.4 Approches basées sur la synthèse de haut niveau/HLS

Les outils de synthèse de haut niveau en anglais High-Level Synthesis (HLS) transforment une spécification de haut niveau non cadencée (ou partiellement cadencée) en une mise en œuvre entièrement cadencée [154]. Ils génèrent de façon automatique ou semi-automatique une architecture personnalisée pour implémenter efficacement la spécification. Outre les bancs de mémoire et les interfaces de communication, l'architecture générée est décrite au niveau du RTL et contient un chemin de données (registres, multiplexeurs, unités fonctionnelles et bus) et un contrôleur, comme l'exigent la spécification donnée et les contraintes de conception.

Comme le montre la figure 3.14, la tâche d'un outil de HLS comprend les étapes suivantes :

— Compilation de la description en langage de haut niveau ;
— Transformation de la spécification fonctionnelle en structures de contrôle matérielles selon les directives du concepteur ;
— Allocation des ressources matérielles (composants de stockage, unités fonctionnelles, bus, etc.) ;
— L'ordonnancement des opérations avec des cycles d'horloge dédiés ;
— Lier les opérations à des unités fonctionnelles personnalisées ou intégrées ;
— Lier les variables aux composants de stockage (LUTs, flip-flops, ou BRAMs) ;
— Lier les transferts de données à des bus spécifiques ;
— Générer la description RTL.

Les sous-tâches de HLS ont des méthodes intégrées pour faciliter l'implémentation rapide de programmes parallèles dans les FPGA. Parmi les étapes citées ci-avant, l'ordonnancement est la tâche la plus importante du processus de HLS. Il planifie les opérations, l'accès à la mémoire et aux interfaces. L'ordonnancement doit garantir que la conception utilise des unités fonctionnelles plus rapides pour les opérations sur le chemin critique et des unités plus lentes pour les opérations en dehors du chemin critique.

L'objectif de l'ordonnancement basé sur les performances est de maximiser l'utilisation des ressources matérielles à l'aide de techniques d'extraction du parallélisme. Il existe plusieurs méthodes d'extraction du parallélisme dans l'approche HLS, et on distingue :

— **Le déroulage de boucle (Loop Unrolling/LU)** est la méthode qui permet à des

itérations successives de boucle d'être exécutées en parallèle. S'il y a trop d'itérations de boucles pour qu'elles tiennent dans un seul FPGA, un LU partiel peut être appliqué. Le déroulage partiel peut également être utilisé pour négocier la surface, la puissance et les performances de la conception résultante.

— **La fusion de boucles (Loop merging/LM)** s'applique aux boucles séquentielles et crée une seule boucle avec la même fonctionnalité que les boucles d'origine. Cette transformation est utilisée pour réduire la latence et la consommation d'espace dans une conception en permettant l'exécution parallèle, lorsque cela est possible, de boucles qui s'exécuteraient normalement en série.

— **Le pipelining automatique (Automatic Pipelining/AP)** est une optimisation permettant d'augmenter le débit de production des résultats. Des registres sont ajoutés dans la conception à des positions appropriées pour minimiser la logique combinatoire entre les registres et maximiser la vitesse d'horloge de la puce.

— **Le pipelining de boucle (Loop Pipelining/LP)** permet d'augmenter le débit d'une boucle (ou de diminuer sa latence globale) en lançant l'itération suivante de la boucle avant que l'itération actuelle ne soit terminée. Le chevauchement de l'exécution des itérations suivantes d'une boucle exploite le parallélisme entre les itérations de la boucle. Le nombre de cycles entre les itérations de la boucle est appelé l'intervalle d'initiation.

— **La synthèse hiérarchique (Hierarchical Synthesis/HS)** est une autre façon d'exposer le parallélisme. Alors que le déroulement des boucles exploite le parallélisme au niveau des instructions et que la fusion des boucles exploite le parallélisme au niveau des boucles, la hiérarchie exploite le parallélisme au niveau des fonctions (niveau des tâches). La SH peut simplifier considérablement les tâches de conception et d'intégration.

Dans la littérature, plusieurs outils utilisent ces méthodes d'extraction de parallélisme pour accélérer les applications pour FPGA par HLS. Numan et al. [155] font un état de l'art sur les outils de synthèse de haut niveau les plus récents. Lan Huang et al. [156] passent en revue la littérature publiée depuis 2014 sur l'optimisation des performances des outils de HLS. Cieszewski et al.[157] font une revue des outils HLS qui mettent en œuvre les méthodes d'exploitation du parallélisme citées ci-avant.

Vivado HLS

Vivado HLS[158] est un outil commercial de HLS fourni par Xilinx pour ses propres FPGA. Il est basé sur AutoPilot[159]. Vivado HLS fournit un environnement de conception permettant de générer des descriptions RTL en VHDL et Verilog à partir de codes synthétisables en langage de haut niveau C, C++, SystemC ou OpenCL. Vivado HLS est basé sur l'infrastructure du compilateur LLVM[160], qui compile le code en langage de haut niveau (*High-Level Language, HLL*) en une représentation de code connue sous le nom de représentation intermédiaire LLVM ou LLVM-IR. Le LLVM-IR passe ensuite par une série de tâches de compilation standard, notamment l'élimination des codes inactifs, la propagation des constantes et le déroulement des boucles, ainsi que des optimisations spécifiques au matériel telles que l'optimisation de la largeur de bit pour réduire la complexité et la redondance du code, maximiser la localité des données et exposer le parallélisme. Vivado HLS utilise la représentation intermédiaire modifiée pour effectuer des optimisations centrées sur la synthèse et l'interconnexion pendant les phases d'ordonnancement des opérations et de liaison des ressources en tenant compte des contraintes spécifiées par l'utilisateur. La représentation intermédiaire est synthétisée en implémentations RTL pour les FPGA de Xilinx. Le RTL généré peut être enregistré sous forme d'une bibliothèque IP pour une utilisation ultérieure. Vivado HLS fournit également des fonctionnalités permettant de vérifier la fonctionnalité du RTL par rapport à la description HLL en utilisant un testbench.

FPGA SDK for OpenCL

Le Open Computing Language (OpenCL) est un framework pour la programmation d'applications parallèles pour des plates-formes hétérogènes comprenant des processeurs, des GPU et des FPGA. Il s'agit d'un standard ouvert maintenu par Khronos Group [161]. Intel supporte OpenCL à travers le logiciel FPGA SDK for OpenCL[162]. Au lieu d'exécuter des threads parallèles de fonctions coûteuses sur plusieurs cœurs, ce compilateur génère des circuits matériels profondément pipelinés qui peuvent être mis en œuvre sur des FPGA de type Intel. L'outil transforme OpenCL en Verilog et en bibliothèques d'exécution pour la partie du système fonctionnant sur le processeur. Il utilise LLVM-Clang[163] pour analyser les constructions OpenCL et produire une représentation intermédiaire. La représentation intermédiaire passe par une série d'optimisations comprenant l'élimination des branches, la fusion des boucles et la vectorisation automatique. Les utilisateurs guident l'optimisation en insérant des annotations pour le déroulement des boucles, le pipelining et le streaming des données. Le compilateur applique automatiquement ces optimisations et traduit la représentation intermédiaire optimisée en Verilog.

LegUp

LegUp[164] est un compilateur HLS développé par des chercheurs de l'Université de Toronto, et maintenant soutenu par LegUp Computing Inc. Il prend en entrée un programme en langage C et fonctionne en deux modes : (1) le mode uniquement matériel, dans lequel l'ensemble du code C d'entrée est synthétisé en RTL ou (2) le mode logiciel-matériel où il synthétise le programme en un système hybride comprenant un processeur (un processeur logiciel MIPS ou ARM) et un ou plusieurs accélérateurs matériels. Dans ce dernier cas, l'utilisateur désigne les fonctions C à implémenter comme accélérateurs. La communication entre le processeur et les accélérateurs se fait par le biais de l'interface bus sur puce à mémoire mappée. LegUp supporte les FPGA fournis par Intel, Xilinx, Lattice, Microsemi et Achronix. Pour la synthèse matérielle, la plupart du langage C est supportée, à l'exception de l'allocation dynamique de mémoire et de la récursivité. LegUp est construit en utilisant le framework LLVM avec Clang comme front-end. Comparé à d'autres outils HLS, LEGUP possède plusieurs caractéristiques uniques. Il prend en charge Pthreads et OpenMP, où les threads logiciels parallèles sont automatiquement synthétisés dans du matériel fonctionnant en parallèle. La réduction automatique de la largeur des bits peut être invoquée pour réduire la largeur des chemins de données en fonction de l'analyse (statique) de la plage des variables et des masques de bits au moment de la compilation. L'analyse des chemins multicycles et la suppression des registres sont également prises en charge, LEGUP éliminant les registres sur certains chemins dont l'exécution nécessite plus d'un cycle, générant ainsi des contraintes pour le back-end du flot.

Le compilateur Intel HLS

Le compilateur Intel HLS[165] transforme une description matérielle de haut niveau écrite en C++ en une description RTL pour les FPGA Intel. En plus des options de compilation standard, le compilateur effectue des tâches d'optimisation telles que le déroulement des boucles, la dépendance des données et le pipelining sur la base des annotations fournies par l'utilisateur. Il permet à l'utilisateur d'explorer les architectures matérielles, notamment les interfaces, le parallélisme, les mémoires, les chemins de données et les boucles à l'aide d'attributs et d'annotations spécifiques. Comme Vivado HLS, Intel HLS Compiler facilite également la génération des IP réutilisables pour l'intégration dans les systèmes, et réduit le temps de développement des FPGA. L'outil prend en charge la vérification du testbench logiciel par rapport au RTL généré. Il est inclus dans la suite logicielle de conception Intel Quartus Prime[135].

Catapult HLS

La plateforme Catapult HLS[166] a été initialement développée par Mentor Graphics sous le nom de Catapult-C[167], elle est actuellement une propriété de Siemens Digital Industries Software. Grâce à Catapult HLS, les développeurs peuvent définir le matériel à l'aide d'un sous-ensemble de C++ et SystemC et générer une description Verilog et VHDL optimisée pour les FPGA et les ASIC. Catapult prend en charge la plupart des constructions C++, notamment les classes, les structures, les tableaux et les pointeurs vers des objets alloués de manière statique. Toutefois, il ne prend pas en charge l'allocation dynamique de la mémoire. Les développeurs de matériel peuvent spécifier le parallélisme, le débit et la configuration de la mémoire à des niveaux d'abstraction élevés en utilisant des attributs et des annotations ; cependant, une compréhension approfondie du matériel sous-jacent est nécessaire pour obtenir un bon résultat. Au cours de la synthèse matérielle, Catapult effectue un certain nombre d'optimisations, dont le déroulement de boucle, la fusion de boucle et le pipelining. Catapult peut générer automatiquement une infrastructure de simulation pour vérifier le RTL généré par HLS par rapport au code source HLL original. Catapult HLS permet également d'incorporer les changements de spécification et le portage vers une technologie différente grâce à la séparation de la fonctionnalité de la conception et des détails de l'implémentation. Le RTL peut simplement être régénéré sur la base du modèle HLS modifié et des nouvelles contraintes.

Il existe beaucoup d'autres solutions HLS autant propriétaires que Open source référencées dans les articles de revue de la littérature cités ci-avant. Les outils HLS permettent de gagner en productivité de conception en générant des circuits pour FPGA correspondant à la description d'une application faite en langage de haut niveau. Cependant, ceux-ci ne prennent pas (si oui très rarement) en compte les modèles et standards de programmation parallèle connus par la communauté de développeurs d'applications parallèles tels que OpenMP et MPI. Seul l'outil LEGUP prend en compte le modèle OpenMP et les threads, mais ne prend pas en compte le standard MPI et plus spécifiquement le standard MPI-2 RMA qui nous intéresse dans le cadre de ce travail.

Autre : SDMPSoC (Software-defined MPSoCs)

SDMPSoC (Software-defined MPSoCs)[168] est un environnement de développement pour MPSoCs hétérogènes qui augmente la productivité et minimise la complexité de conception des MPSoCs basés sur FPGA. Cet environnement consiste en un flot automatique qui analyse un programme basé sur MPI pour construire un MPSoC approprié. Ce MPSoC est capable d'exécuter efficacement le programme MPI, car il est composé de processeurs MicroBlaze (MB) avec des accélérateurs optionnels. MPI est utilisé pour définir différents processus dans un seul fichier de programme. Chaque processus du programme MPI est exécuté par un élément de traitement (Processing Element/PE) qui peut être soit un processeur MB, soit un module matériel. Les PE échangent des données par l'intermédiaire d'un réseau maillé 2D sur puce (NoC) fournissant une infrastructure de communication évolutive. L'utilisateur peut définir le type de PE pour un processus en utilisant des contraintes. Si un module matériel a été sélectionné, SDMPSoC adapte automatiquement le programme MPI pour qu'il puisse être synthétisé en module matériel à l'aide de Vivado HLS. Si un processeur MB a été sélectionné, SDMPSoC adapte le programme MPI pour qu'il puisse être compilé pour un processeur MB. Chaque processeur MB peut également être équipé d'un ou plusieurs modules matériels pour accélérer le processus s'exécutant sur ce processeur MB. Les fonctions définies dans le programme MPI peuvent être marquées par des pragmas et, par conséquent, synthétisées en modules matériels pour les processus qui exécutent ces fonctions. Vivado HLS est utilisé pour synthétiser ces fonctions. Les pilotes de périphériques qui gèrent les transferts de données entre le processeur MB et les modules matériels sont automatiquement insérés par SDMPSoC dans les programmes pour les processeurs MB. Le flux est

automatisé par des scripts Python, qui génèrent des scripts TCL pour le MPSoC et les modules matériels ainsi que les programmes pour les processeurs MB. De cette manière, la programmation et la conception sont considérablement simplifiées, ce qui permet de réduire le temps de développement.

L'outil SDMPSoC permet de générer un MPSoC à partir d'un programme décrit en langage de haut niveau. En outre, l'architecture obtenue permet la communication MPI entre les différents éléments de traitement (PE) ; cependant, le modèle de communication MPI utilisé est de type bilatéral qui s'avère souvent être moins efficace que la communication unilatérale implémentée via le standard MPI-2 RMA.

Les approches HLS améliorent la productivité de mise en œuvre d'applications sur FPGA mais celles-ci ne prennent pas toujours en compte la mise en œuvre des modèles parallèles ; quand bien même c'est les cas, les solutions HLS n'intègrent pas le paradigme MPI-2 RMA. La prochaine section présente une autre catégorie d'approche de solution que sont les Overlays.

3.8.5 Approches basées sur les FPGA Overlay

D'après [169], *"un FPGA overlay est une architecture virtuelle reconfigurable qui se superpose à la fabrique physique configurable du FPGA "*. En d'autres termes, un FPGA overlay est une couche virtuelle d'architecture qui se situe conceptuellement entre l'application utilisateur et le FPGA physique sous-jacent, comme le montre la figure 3.15. Avec cette couche supplémentaire, les applications utilisateur ne seront plus implémentées directement sur le FPGA physique. Au lieu de cela, l'application sera ciblée vers l'architecture overlay, quel que soit le FPGA physique. Une étape distincte permettra ensuite de traduire cette architecture overlay, ainsi que l'application qui s'y exécute, sur le FPGA physique. Cette catégorie d'outils présente donc de nombreux avantages, en termes de virtualisation, de réduction du temps de compilation, d'amélioration des capacités de débogage, etc.

Un FPGA overlay peut être soit configuré spatialement en anglais spatially configured (SC), soit multiplexée dans le temps en anglais time-multiplexed (TM), en fonction de sa configurabilité au cours du fonctionnement. Si un FPGA overlay possède des unités fonctionnelles (functional units / FU) auxquelles sont attribuées des tâches fixes, on parle d'overlay de type SC. Si un FPGA overlay peut changer le fonctionnement de ses unités fonctionnelles sur une base cycle par cycle, l'overlay est appelée est de type TM. L'interconnexion entre les unités fonctionnelles des superpositions SC et TM peut être fixe ou reconfigurable au moment de l'exécution. Selon la granularité (à laquelle le matériel peut être programmé) d'une architecture overlay, les overlays SC et TM peuvent être classés en overlays à grain fin et à gros grain. Les overlays à grain fin peuvent fonctionner au niveau des bits tout comme les FPGA physiques, mais offrent une programmabilité et une configurabilité à un niveau d'abstraction plus élevé que les FPGA physiques[170].

La revue faite dans [171] met en évidence le fait que de nombreux outils d'overlay ont été développés dans les deux classes (SC/TM). Dans cette section, nous n'allons nous intéresser qu'aux overlays qui traitent des applications parallèles.

Le travail effectué dans le document in[172] met en évidence une liste de certains overlays de traitement parallèle antérieurs, dont DRAGON [13][173] qui est l'outil sur lequel porte le travail mentionné.

Le réseau systolique de calcul et de mémoire (systolic computational-memory array / SCMA)

Pour les simulations numériques scientifiques qui nécessitent un rapport relativement élevé entre l'accès aux données et la communication, l'évolutivité de la bande passante de la mémoire

13. Dynamically Re-programmable Architecture for Gather/scatter Overlay Nodes

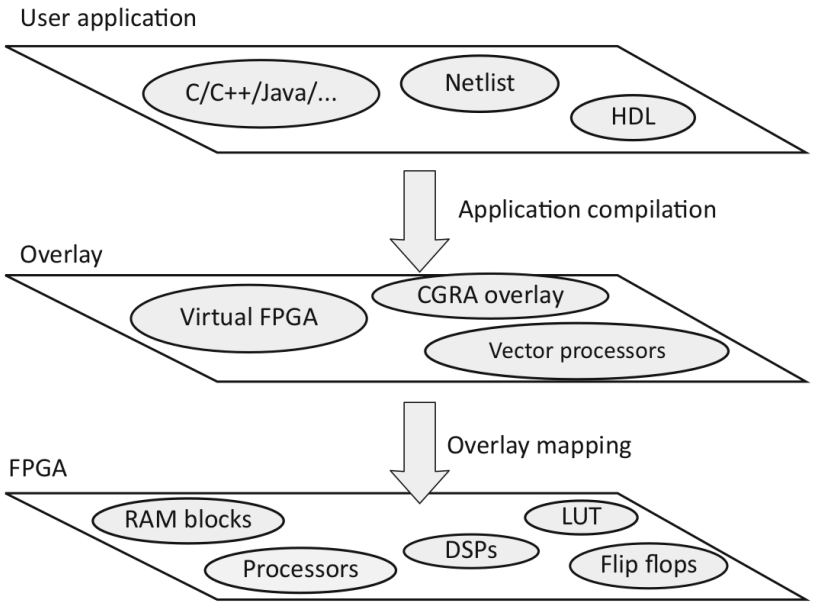

Figure 3.15: *Utilisation d'un overlay pour former une approche à deux niveaux de développement d'applications sur FPGA [169].*

est la clé de l'amélioration des performances. Les machines de calcul personnalisées (custom-computing machines/CCM) sont donc l'une des approches prometteuses pour fournir des structures tenant compte de la bande passante et adaptées aux applications individuelles. Dans [174] il est proposé un réseau de FPGA évolutif doté d'un mécanisme de réduction de la bande passante (bandwidth-reduction mechanism / BRM) pour mettre en œuvre des CCM performants et économes en énergie pour les simulations scientifiques basées sur les méthodes de différences finies. Avec une matrice de FPGA, KENTARO et al. construisent un réseau systolique de calcul et de mémoire (systolic computational-memory array / SCMA), qui est doté d'un minimum de programmabilité afin de fournir une flexibilité et une productivité élevée pour divers noyaux de calcul et calculs de limites. Le réseau systolique[175] est une disposition régulière de nombreux éléments de traitement (PE) dans un réseau, où les données sont traitées et circulent de manière synchrone dans le réseau entre voisins. Ce type de matrice convient au pipelining et au traitement parallèle dans l'espace, les données d'entrée traversant la matrice, ce qui permet d'obtenir des performances arithmétiques évolutives en fonction de la taille de la matrice. Cependant, l'accès à la mémoire externe peut encore constituer un goulot d'étranglement pour l'amélioration des performances des calculs gourmands en mémoire. KENTARO et al. pensent que l'approche de la mémoire computationnelle est l'une des solutions à ce problème. Cette approche est similaire à la RAM computationnelle ou au concept de *"traitement en mémoire"*[176], où la logique de calcul et la mémoire sont disposées très près l'une de l'autre. Dans cette architecture SCM, la matrice entière se comporte comme une mémoire, non seulement en stockant des données mais aussi en effectuant elle-même des opérations à virgule flottante sur celles-ci. La mémoire est partitionnée en mémoires locales décentralisées vers les PE, qui effectuent simultanément des calculs avec les données stockées dans leurs mémoires locales. Cette structure permet à la bande passante de la mémoire interne du réseau d'être large et adaptable à sa taille sans le goulot d'étranglement de l'accès à la mémoire externe. La figure 3.16 illustre la structure de cette architecture. Le chemin de données des PE est composé d'un séquenceur, d'une mémoire locale et d'une unité MAC (Multiplication and ACcumulation). La mémoire locale stocke toutes les données nécessaires pour le sous-réseau alloué au PE, ainsi que les résultats temporels ou intermédiaires des calculs. Pour décrire les microprogrammes qui effectuent des calculs numériques avec la partie HW susmentionnée, les auteurs ont défini un langage d'assemblage basé sur les exigences sui-

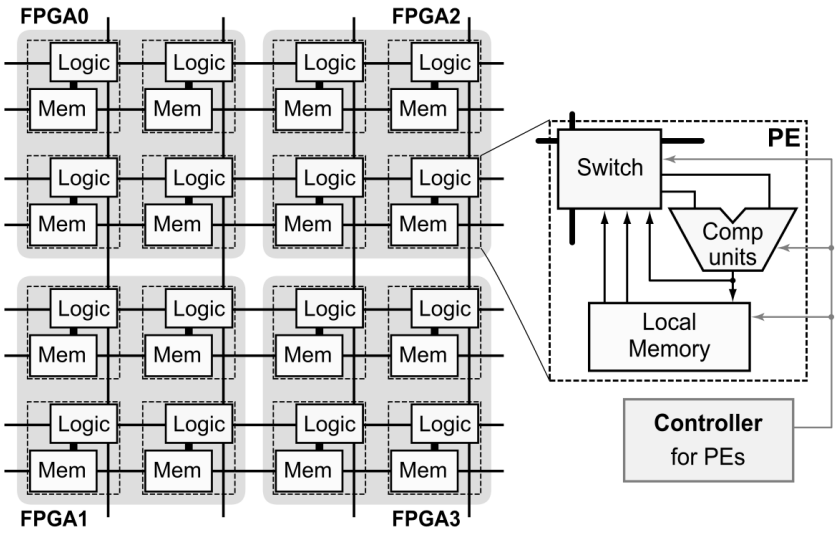

Figure 3.16: *Réseau de mémoire de calcul systolique sur un réseau FPGA 2D*

vantes : (1) L'unité MAC prend les deux entrées de la mémoire ou des FIFOs. (2) L'unité MAC multiplie, puis ajoute ou accumule. (3) La sortie de l'unité MAC est écrite dans la mémoire et/ou les FIFO. (4) Les calculs sont répétés par des boucles imbriquées.

Pour démontrer la faisabilité et évaluer quantitativement les performances, les auteurs ont conçu et mis en œuvre le SCMA de 192 éléments de traitement sur deux FPGA ALTERA Stratix II. Le SCMA implémenté fonctionnant à 106 MHz a une performance de pointe de 40,7 GFlops en simple précision. Ils démontrent que le SCMA atteint des performances soutenues de 32,8 à 35,7 GFlops pour trois calculs de référence avec une utilisation élevée des unités de calcul. Le SCMA double FPGA permet un calcul 6 à 29 fois plus rapide que le calcul logiciel avec le processeur Pentium4 fonctionnant à 3,4 GHz.

Cette solution est intéressante en termes de performances, mais elle ne peut répondre qu'à des applications qui sont modélisables sous forme d'architectures systoliques.

reMORPH

reMORPH[177] est un FPGA overlay composé de modules à gros grains (coarse grained modules/CGRM) connectés entre eux par des liens de communication malléables. L'architecture de communication semi-systolique à mémoire partagée de type "near neighbor" a été conçue pour exploiter la reconfiguration partielle en cours d'exécution présente dans les FPGA de Xilinx afin d'obtenir un avantage significatif en termes de performances et de surface, tout en travaillant avec des coûts de reconfiguration très faibles. L'élément de calcul de base de reMORPH possède un pipeline à 5 étages utilisant le DSP48E comme ALU (Arithmetic Logic Unit) et possède 512 registres de données adressables ainsi que 512 registres d'instructions. Le micro-séquenceur travaille sur un format d'instruction expresse permettant au processeur d'être construit avec une empreinte très faible d'environ 200 slice LUTs [14]. Toute application (ré)utilise autant de blocs que nécessaire pour répondre aux contraintes de performance - le code exécuté dans les processeurs ainsi que le réseau d'interconnexion entre les blocs sont modifiés au moment de l'exécution de l'application selon un programme déterminé statiquement. La figure 3.17 illustre la structure de reMORPH.

reMORPH a une structuration à mémoire partagée, ce qui signifie qu'elle partage les défauts

14. La LUT(LookUp Table) est le bloc de construction de base d'un FPGA capable d'implémenter n'importe quelle fonction logique de N variables booléennes. Essentiellement, cet élément est une table de vérité dans laquelle différentes combinaisons d'entrées mettent en œuvre différentes fonctions pour produire des valeurs de sortie.

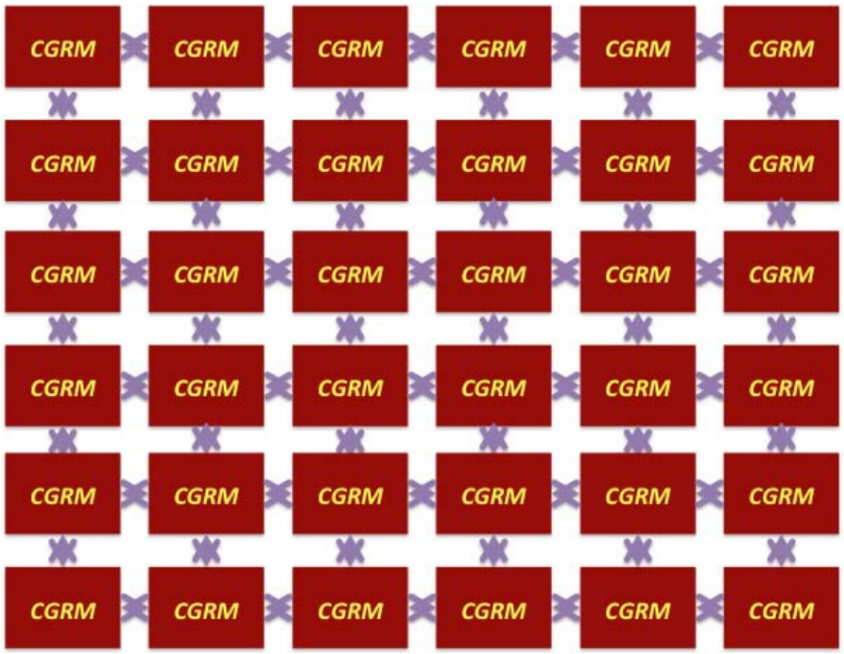

Figure 3.17: *La structure de l'architecture reMORPH*

liés à ce type de structuration comme précisé dans la section 3.2.2 . En outre le processus de déploiement d'applications n'est pas totalement automatique.

Scalable Streaming-Array (SSA)

SSA[178] est une machine de calcul personnalisée pour accélérer le calcul itératif des stencils [15] avec plusieurs FPGA. C'est une architecture évolutive de réseau en continu pour le calcul de stencils à haute performance sur FPGA. L'architecture est conçue sur la base du concept de programmation spécifique à un domaine, où une machine de calcul personnalisée est programmable avec la fonctionnalité minimale requise pour le domaine de l'algorithme. Cette machine est basée sur une architecture, qui exploite le parallélisme spatial et temporel du calcul de stencils en continu. L'approche du pipelining est utilisée sur des itérations successives pour permettre un calcul profondément pipeliné avec plus de FPGA. SSA sur plusieurs FPGA connectés dans un réseau linéaire construit un pipeline profond avec de nombreux étages SIMD (single-instruction stream and multiple-data stream). Comme la profondeur d'un pipeline n'influence pas le débit, on peut augmenter les performances de calcul avec une bande passante mémoire constante en connectant plus de FPGA pour un SSA plus long. L'architecture SSA programmable peut effectuer divers calculs de stencil avec des opérations en virgule flottante. La structure des éléments processeurs (Processing Element/PE), se compose d'une mémoire tampon pour la mise en mémoire tampon cyclique, de la mémoire constante pour stocker les coefficients du calcul du stencil et de l'unité de multiplication et d'accumulation en virgule flottante (Floating-point Multiply-and-Accumulate unit/FMAC). Le chemin des données est pipeliné avec les huit étapes suivantes : la récupération des instructions (IF), la lecture de la mémoire (MR), les exécutions 1 à 5 (EX1 à EX5) et l'écriture de la mémoire (MW). Les figures 3.18 et 3.19 illustres respectivement la structure de SSA à l'intérieur d'u FPGA et la structure globale sur plusieurs FPGAs.

Cette architecture présente un grand avantage en termes d'efficience. Cependant il se pourrait qu'elle ne soit pas adaptée pour des applications qui ne peuvent pas être structurées sous forme

15. Un stencil (pochoir en français) est un motif (pattern) de calcul que l'on applique à une cellule afin de mettre à jour les éléments d'un tableau multidimensionnel. Une opération sur un tableau où chaque élément des données en sortie est une somme pondérée d'un ensemble d'éléments voisins des données en entrée.

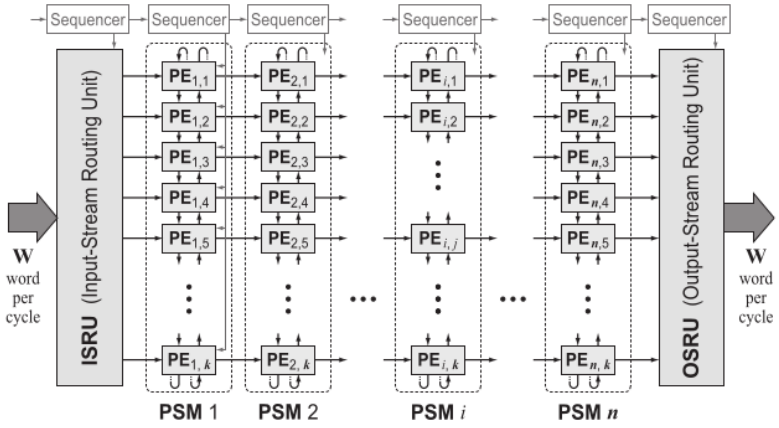

Figure 3.18: *Vue d'ensemble du SSA sur chaque FPGA*

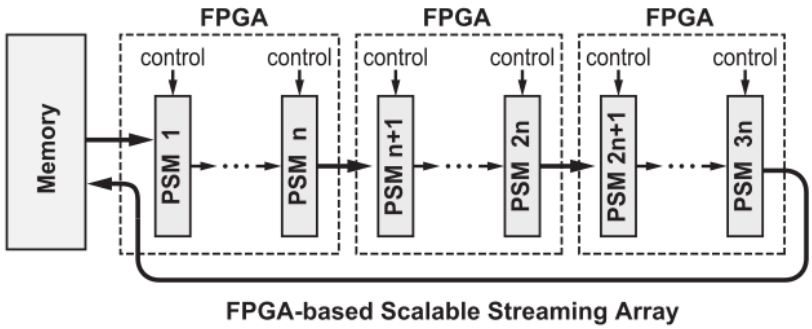

Figure 3.19: *Architecture SSA*

de stencil.

GRVI Phalanx

GRVI est un processeur logiciel RISC-V RV32I efficace sur FPGA. Phalanx est un framework de processeurs parallèles et de réseaux d'accélérateurs. GRVI Phalanx[179] est un framework pour la construction d'accélérateurs. Un concepteur peut utiliser n'importe quelle combinaison de logiciels, de logiciels accélérés ou d'accélérateurs, regroupés en grappes. Dans un cluster, les cœurs peuvent être couplés directement ou communiquer par le biais de la mémoire partagée. Entre les clusters et les cœurs d'interface externes, un large NOC Hoplite[180] transporte les données sous forme de messages. Dans [179] il a été construit un Phalanx de 400 GRVI implémenté dans un Kintex UltraScale KU040. Il comporte 10 lignes et 5 colonnes de clusters (c'est-à-dire sur un NOC Hoplite 10x5) ; chaque cluster comporte 8 PE partageant 32 Ko de RAM. Cette conception utilise 73% des LUTs du dispositif et 100% de ses BRAMs. Le NOC Hoplite de 300 bits de large utilise 6% des LUTs du dispositif ; son coût amorti est inférieur à 40 LUTs/PE. La taille totale de chaque processeur, de sa part de l'interconnexion de la grappe et du routeur Hoplite est d'environ 440 LUT.

La structuration des PE à l'intérieur d'un cluster est à mémoire partagée cela signifie qu'il faut des mécanismes de synchronisation entre PE, et la scalabilité sur les transactions en mémoire est limitée tel que précisé dans la section 3.2.2.

VectorBlox MXP

Le processeur matriciel VectorBlox MXP[181], est un processeur logiciel (soft core) basé sur FPGA et capable d'une exécution hautement parallèle. Entièrement programmé en C, le MXP

est capable d'exécuter des algorithmes logiciels parallèles aux données à des vitesses similaires à celles du matériel. La conception paramétrée du MXP permet à l'utilisateur de spécifier la quantité de parallélisme requise, allant de 1 à 128 ALU parallèles ou plus. Les principales caractéristiques du MXP comprennent une mémoire scratchpad à accès parallèle pour contenir les données vectorielles et des moteurs DMA et scatter/gather à haut débit. Pour offrir des performances extrêmes, le processeur est extensible avec des instructions vectorielles spécifiques et des filtres DMA personnalisés. Enfin, le MXP s'intègre parfaitement aux flux de développement existants d'Altera et de Xilinx, simplifiant ainsi la création et le déploiement de systèmes.

Cet outil est limité, parce que toutes les applications ne peuvent pas être facilement vectorisées. Les algorithmes qui effectuent des opérations différentes sur chaque élément du vecteur, en particulier si elles dépendent de la valeur de l'élément ou d'éléments voisins, sont également très difficiles à implémenter sur processeur vectoriel. De même, les algorithmes qui effectuent des mises à jour séquentielles des données sont difficiles à paralléliser. Enfin, les algorithmes qui nécessitent des valeurs provenant de voies vectorielles adjacentes peuvent être difficiles à vectoriser.

Octavo

Octavo[182] amélioré dans [183] est un processeur (soft-processor) capable de fonctionner à la vitesse maximale autorisée de la puce FPGA sous-jacente qui, dans les dispositifs Stratix IV, est limitée à 550 MHz. Pour atteindre cette limite, Octavo possède 10 étages de pipeline et des instructions de 36 bits, comprenant un opcode de 4 bits et trois adresses de 10 bits : deux sources, une destination, avec 2 bits (sur 36) inutilisés. Octavo n'a pas de fichier de registre ; il lit/écrit les opérandes entiers de 36 bits directement à partir de la mémoire RAM. Il ne supporte que l'adressage direct, ce qui, avec des opérandes de 10 bits, implique 1024 mots d'espace mémoire adressable par opérande. Enfin, Octavo multithreads son pipeline avec 8 threads indépendants, avec des instructions de threads émises dans un ordre fixe round-robin pour éviter tout risque de RAW ou de pénalités de branchement dans chaque thread. Chaque instruction thread se termine complètement avant que la suivante ne commence. Octavo est hautement paramétré, ce qui permet d'explorer les compromis entre la largeur du chemin de données et de la mémoire, la profondeur de la mémoire et le nombre de contextes de threads pris en charge.

Dans le travail [184], une comparaison est faite en termes de micro-architecture, performances et de surface de deux overlay de processeurs logiciels : Octavo et MXP. Pour mesurer les écarts de surface et de performance de ces overlay par rapport au matériel FPGA sous-jacent, Laforest et al. comparent les implémentations directes sur FPGA des micro-benchmarks écrits en C synthétisés avec l'outil LegUp HLS et également écrits en Verilog HDL. Octavo s'avère être efficace autant en consommation en énergie qu'en performance sur le bentch mark utilisé comparé à des implémentations HLS.

Toutefois, Octavo ne manipule pas le standard MPI qui est celui qui nous intéresse dans le cadre de ce travail, et qui est très utilisé pour la parallélisation de logiciels.

2GRVI Phalanx

2GRVI (et son prédécesseur, GRVI) sont des processeurs RISC-V 64 bits (resp. 32 bits) efficaces sur FPGA. Phalanx est un framework d'overlay de processeurs parallèles et de réseaux d'accélérateurs. Des groupes de PE et de cœurs d'accélérateur forment des clusters de calcul à mémoire partagée. Les clusters, la DRAM, les NIC et les autres contrôleurs d'E/S communiquent par passage de messages sur un soft NoC Hoplite de type tore optimisé sur FPGA. 2GRVI Phalanx [185] est la version 2 de GRVI Phalanx[179], un overlay de processeur parallèle pour simplifier le développement d'accélérateurs. 2GRVI Phalanx prend en charge une approche de type logicielle. Une application C++ ou OpenCL s'exécute sur les processeurs logiciels de l'overlay. Ensuite, des instructions personnalisées, des cœurs d'accélérateur ou des mémoires peuvent être

introduits pour accélérer les goulots d'étranglement. Ces travaux remanient le GRVI Phalanx afin de tirer parti des nouveaux FPGA de Xilinx avec 460 Go/s de DRAM HBM2, et de fournir une expérience de programmation parallèle familière via un modèle de programmation et des outils de type OpenCL. Le nouveau système est le premier SoC RV64I kilocore et le premier multiprocesseur RISC-V avec un système de mémoire HBM2[186].

2GRVI Phalanx ne manipule donc pas le standard MPI qui nous intéresse dans le cadre de ces travaux et qui est très connu des développeurs d'applications parallèles

DRAGON

DRAGON (Dynamically Reconfigurable Architecture for Gather-scatter Overlay Nodes)[173] est un overlay qui offre à la fois la possibilité d'utiliser des unités d'exécution en virgule flottante double précision et en nombres entiers 64 bits, ainsi qu'un modèle de mémoire de diffusion efficace et une interconnexion Tore 4D évolutive. L'architecture DRAGON proposée, adopte la plupart des concepts de traitement parallèle, tels que le parallélisme en pipeline, un modèle SIMD (Single Instruction Multiple Data) ainsi qu'une approche de conception VLIW (Very large Instruction Word). L'architecture DRAGON se compose de deux éléments principaux : l'accélérateur et le contrôleur (en cours de développement). L'élément de traitement (PE) est la partie la plus importante de l'accélérateur. Il s'agit d'un processeur personnalisé, polyvalent et efficace, qui prend en charge les opérations en nombres entiers 64 bits et en virgule flottante à double précision. Le PE est conçu pour prendre en charge uniquement les instructions de calcul et les opérations de déplacement de données. Cela signifie qu'aucun branchement ou exécution de code conditionnel n'est pris en charge dans un PE, sauf dans le cas où des masques sont utilisés pour charger sélectivement des données de la BM/Broadcast Memory (mémoire de diffusion) vers la LM/Local Memory (mémoire locale) ou les stocker en retour. La micro-architecture du PE comporte un pipeline à 6 étages, à savoir un étage de décodage de registre, 3 étages d'exécution, un étage de mémoire et un étage de réécriture. L'accélérateur DRAGON a une architecture modulaire qui regroupe plusieurs modules appelés SuperClusters (SCs). Chaque SC est constitué de plusieurs BC (Broadcast Cluster) comme le montre la figure 3.20 a). Un BC consiste en un cluster relativement petit de 16 PE qui sont étroitement connectés par un schéma de type Tore 2D. Ces PE peuvent échanger des données avec la BM via un BMC (Broadcast Memory Controller) programmable. L'interface externe de la BM est connectée au contrôleur DRAGON par une unité DMA (Direct Memory Access) qui gère les transferts de données entre l'accélérateur et le contrôleur, comme le montre la figure 3.20 b). De l'autre côté, l'interface interne est connectée à la BMC qui assure la diffusion des données de la BM aux 16 PE par le biais du signal de données BC de la BMC, comme le montre la figure 3.20 b). Fondamentalement, une partie du flux d'instructions est dédiée au BMC, pour lui permettre de prélever des données de la mémoire centrale, afin de les préparer à être diffusées vers les postes de travail chaque fois qu'une instruction de calcul implique un opérande de la mémoire centrale. Lorsque les PE doivent stocker des données dans la mémoire centrale, leur sortie est enregistrée par l'échantillonneur, puis sérialisée vers le BMC, qui gère à son tour les opérations d'écriture dans la mémoire centrale.

DRAGON ne dispose pas d'un modèle de programmation standard connu par les développeurs d'applications parallèles et donc nécessite des efforts d'apprentissage pour une prise en main effective. En outre DRAGON est spécialisé pour les modèles parallèles de type SIMD et VLIW, il serait par conséquent moins adapté pour tout autre modèle parallèle.

Les différents FPGA Overlay présentés ci-avant adressent la problématique de la conception d'applications parallèles sur MP-RSoC, mais ne correspondent pas aux contraintes et exigences du modèle parallèle de notre application. Globalement, ils sont soit très spécialisés pour un type de modèle parallèle (à l'instar de SSA qui prend en compte les modèles basés sur le pattern stencil), soit ne prennent pas en compte la structuration à mémoire distribuée avec la communication

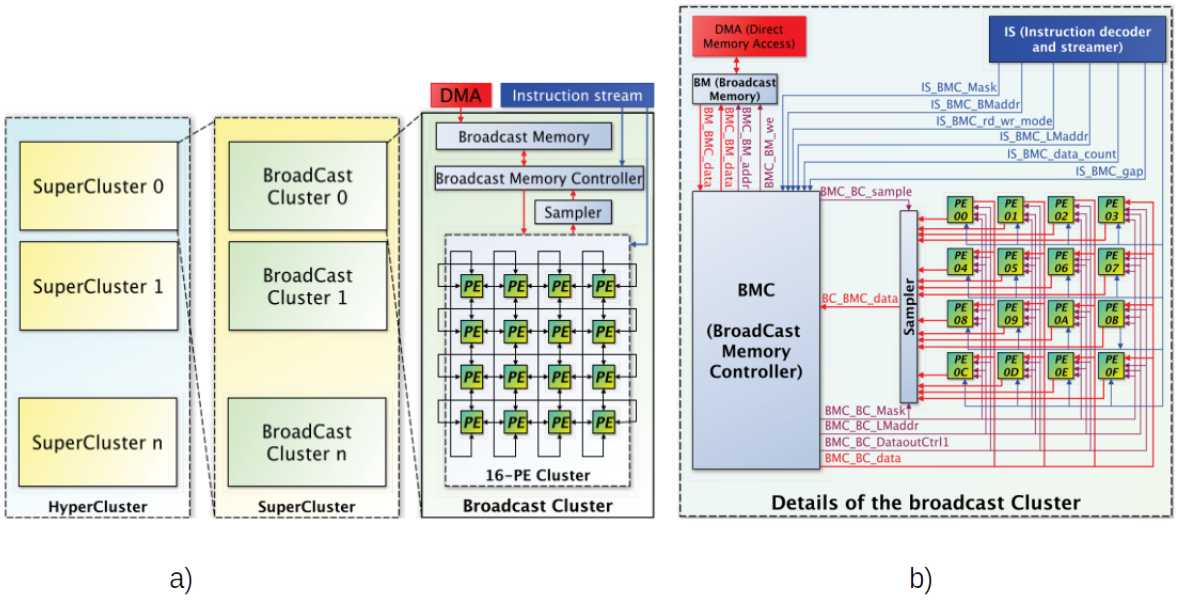

Figure 3.20: *a) Aspect modulaire de l'architecture DRAGON proposée, b) Détails des flux d'instructions et de données à l'intérieur d'un cluster de diffusion (Broadcast Cluster).*

basée sur le paradigme MPI-2 RMA qui est connu pour son efficacité.

3.9 MATIP : support d'une approche de mise en œuvre productive d'applications parallèles MPI-2 RMA sur MP-RSoC

Nous aimerions avoir une approche qui permettrait à un ingénieur de concevoir une application parallèle avec une structuration à mémoire distribuée avec une communication basée sur le paradigme MPI-2 RMA. L'idée est de libérer le développeur d'un maximum de problématiques rencontrées pour la mise en œuvre d'une application parallèle MPI-2 RMA sur MP-RSoC. En outre, une telle approche devrait être basée sur un langage de haut niveau manipulé par les développeurs d'applications parallèles sur environnement logiciel. Dans une telle approche, il est question de disposer d'une méthodologie pertinente et efficace qui puisse permettre à un ingénieur de concevoir des applications parallèles MPI-2 RMA à partir d'une description faite en langage de haut niveau comme le C, C++, Java, etc. L'idée étant de partir de cette description de haut niveau pour générer une plateforme distribuée où les éléments processeurs communiquent par MPI-2 RMA. Cela nécessiterait une infrastructure réseau sur puce dotée d'une API de communication MPI ; ou alors un outil de génération et de configuration automatique d'une telle infrastructure.

Des solutions existent basées sur un réseau sur puce qui interconnectent des processeurs communiquant par MPI ou alors une approche mixte processeurs/accélérateurs matériels. Cependant, l'usage des processeurs ajoute un aléa d'exécution séquentielle des instructions liées à chaque tâche. Notre idée est donc de pouvoir générer un réseau d'unités de traitement de type accélérateur matériel pour plus d'efficience en termes de latence, surface de FPGA, et consommation en énergie.

L'architecture générée devrait intégrer une communication de type MPI-2 RMA entre tâches. Cela implique la génération d'un réseau sur puce dotée d'une API de communication MPI-2 RMA qui pourra permettre les échanges d'informations entre tâches pour la réalisation de l'application. Dans l'état de l'art, il existe des travaux qui proposent ce genre d'infrastructure de communication ; notamment : la plateforme proposée par Uhlendorf et al. [187], MPI HAL

de Minhass et al.[188], SOC-MPI de Mahr et al. [189], TMD-MPI de Saldana et al.[190], etc. La solution qui a retenu notre attention est celle de Christian Gamom et al. [191], qui proposent MATIP [16], une plateforme qui permet le déploiement de tâches matérielles sur MP-RSoC avec une infrastructure de communication et une API MPI. C'est d'ailleurs la seule qui implémente le standard de communication MPI-2 RMA. En outre, MATIP est au niveau RTL (décrit en VHDL) donc serait compatible avec des tâches matérielles décrites en VHDL. MATIP constitue la couche intermédiaire sur laquelle peuvent s'intégrer des tâches matérielles générées à partir de la description de l'application faite dans un langage de haut niveau. MATIP correspond à un FPGA Overlay.

MATIP n'intègre que des taches matérielles, mais les parties séquentielles d'une application s'exécutent efficacement sur un processeur. Il est nécessaire de réfléchir à adjoindre à MATIP des processeurs.

Nous proposons d'utiliser une solution de type FPGA Overlay basée sur la synthèse de haut niveau et qui intègre la co-conception matérielle logicielle.

Dans le but de développer le modèle parallèle d'un algorithme d'extraction des caractéristiques pour la reconnaissance des langues à tons sur système embarqué, dans cette thèse nous proposons une approche de mise en œuvre productive d'applications parallèles MPI-2 RMA qui exploite MATIP comme FPGA Overlay basé sur la synthèse de haut niveau et qui intègre la co-conception matérielle logicielle. Dans les prochains chapitres nous décrivons les différents éléments qui constituent cette approche en termes d'étapes, méthodes, outils, etc.

3.10 Résumé du chapitre

Les systèmes embarqués correspondent au type de plate-forme privilégié pour le déploiement de système de reconnaissance vocale, car très souvent utilisées comme interface homme machine pour l'interaction avec des applications qui exploitent cette technologie. Pour des besoins d'efficacité et de réactivité, ces applications sont souvent régies par un impératif de temps réel. La parallélisation est une approche viable pour pouvoir répondre à cette contrainte. A travers la mise en œuvre d'un modèle parallèle de notre application sur environnement logiciel, nous avons compris que l'architecture matérielle joue un rôle important pour avoir de meilleures performances. Parmi les architectures des systèmes embarqués, les systèmes reconfigurables constituent une plateforme qui présente de nombreux atouts pour la mise en œuvre d'applications sur systèmes embarqués, notamment la dynamicité, la performance, la consommation en énergie. Les systèmes parallèles sur architecture reconfigurables que sont les MP-RSoC constituent un espace de solution très attrayant non seulement de par la multitude d'avantages qu'ils présentent en termes de flexibilité, temps réduit de mise sur le marché, coût et évolutivité ; mais aussi du fait de la diversité de leurs cas d'utilisation et de la multitude des domaines dans lesquels ils sont exploités. Toutefois leur caractère parallèle et hétérogène crée de nombreuses problématiques à lever par les concepteurs d'applications ; en termes de limite des ressources, synchronisation entre processeurs, cohérence de cache, communications efficaces, et la mise sur pied des outils et stratégies de programmation parallèle. C'est la raison pour laquelle, dans cette thèse, nous adressons le problème de productivité de conception des applications parallèles sur MP-RSoC. Où il est question de voir comment faciliter la conception et la mise en œuvre d'applications parallèles à mémoire distribuée sur MP-RSoC par une approche de génération automatique de plateforme, tout en intégrant une approche de co-conception matérielle/logicielle. Nous avons parcouru l'état de l'art des approches et outils associés qui peuvent augmenter la productivité de conception des applications parallèles sur MP-RSoC. Cependant très peu d'approches et outils prennent en compte les applications parallèles avec une structuration à mémoire distribuée et exploitant le standard de communication MPI-2 RMA. La spécification des attentes vis-à-vis de l'approche qui puisse

16. MPI Application Task Integration Platform

aider l'ingénieur à mettre en œuvre de façon productive une application parallèle MP-2 RMA sur MP-RSoC nous a permis d'identifier l'outil MATIP comme support d'une approche de FPGA Overlay basé sur la synthèse de haut niveau. Dans la suite de ce travail, nous vous proposons une approche d'augmentation de la productivité de conception d'applications parallèles à mémoire distribuée et exploitant le standard MPI-2 RMA. Dans le chapitre suivant, nous vous présentons l'essentiel de la méthodologie et les outils qui sous-tendent notre proposition de solution.

Chapitre

4 Proposition d'une approche de facilitation de la co-conception d'applications parallèles MPI sur MP-RSoC

Sommaire

4.1 Introduction

Les applications parallèles étant complexes, les ingénieurs ont besoin d'approches, méthodes et outils leur permettant de surmonter les différents challenges sous-jacents à la conception de ce type d'application, améliorant ainsi leur productivité. C'est le cas dans le cadre de ce travail, où nous aimerions mettre en œuvre dans un système embarqué (sur architecture reconfigurable) un modèle parallèle d'un algorithme d'extraction des caractéristiques pour la reconnaissance des langues à tons. Dans le présent chapitre, nous proposons une approche pour la conception

d'applications parallèles à mémoire distribuée, au standard de communication MPI-2 RMA sur MP-RSoC. Nous utilisons pour cela l'Overlay basé sur l'outil MATIP[191] associé à une approche de synthèse de haut niveau. Dans la section 4.2 nous présentons l'idée de notre approche, nous présentons l'outil MATIP et l'analysons. Dans la section 4.3 nous décrivons les différentes étapes de notre approche ; puis dans la section 4.4, nous en démontrons l'efficacité en effectuant l'implémentation du modèle parallèle de l'algorithme d'extraction des caractéristiques.

4.2 HLS+MATIP : une approche de conception productive d'applications parallèles sur MP-RSoC

Nous présentons une méthodologie permettant à un ingénieur de concevoir des application parallèles MPI-RMA à partir d'une description faite en langage de haut niveau comme le C, C++, Java, etc. L'idée étant de partir de cette description de haut niveau pour générer une plateforme distribuée où les éléments processeurs communiquent par MPI-2 RMA. Spécifiquement, il s'agit d'une approche de FPGA Overlay basée sur la synthèse de haut niveau et qui intègre la co-conception matérielle logicielle. Pour détailler afin de mieux comprendre cette approche, dans la prochaine section, nous présentons la plateforme MATIP.

4.2.1 Présentation de la plateforme MATIP

MATIP est un environnement conçu par R. Christian GAMOM[191] pour déployer des tâches matérielles dans un MP-RSoC avec une configuration à mémoire distribuée. MATIP est structuré selon une architecture en couches inspirée des travaux décrits par Pavel Zaykov dans [192]. Cette architecture est composée de trois couches : la couche d'interconnexion qui réalise les liens physiques permettant le transit de l'information entre les tâches ; la couche de communication qui offre des services de communication entre les tâches et la couche applicative qui regroupe les différentes tâches qui constituent l'application.

Dans MATIP, il a été choisi comme interconnexion, un réseau dynamique simple constitué d'un seul commutateur ou crossbar qui ne nécessite pas de routage complexe et qui permet de réaliser un réseau compatible avec la reconfiguration dynamique.

La couche de communication est constituée d'un composant qui recrée une version matérielle de l'environnement de communication MPI-2 et exploite le réseau. Ce composant est appelé MPI-HCL (Message Passing Interface Hardware Communication Layer). Le composant de communication MPI-HCL est un processeur qui exécute les primitives MPI fournies sous forme de micro-instructions. Ce processeur est logiquement composé de deux parties : l'une pour exécuter les primitives demandées par la tâche matérielle locale, appelée Core 1 et l'autre pour traiter les primitives initiées par les tâches matérielles distantes à travers le réseau d'interconnexion, cette partie est appelée Core 2.

La couche applicative a pour but de faciliter l'intégration d'une tâche matérielle avec la plateforme MATIP. C'est sur cette couche que le concepteur interagit avec la plate-forme MATIP pour déployer son application. Pour déployer une application sur MATIP, le concepteur devra décomposer son application sous forme de modules, chaque module réalisant une fonctionnalité, ou un traitement requis que l'on appelle dans l'environnement MATIP un *tâche matérielle*. Ainsi, le concepteur dispose d'un modèle appelé TIC (Task Integration Component) qui est l'environnement qui permet à cette tâche matérielle de communiquer avec l'extérieur. La plateforme MATIP définit un modèle pour la réalisation de tâches matérielles communicantes, associant à une tâche utilisateur, une mémoire et des primitives de communication, pour envoyer et recevoir des messages basés sur MPI. La figure 4.1 présente un résumé de l'architecture de la plate-forme MATIP.

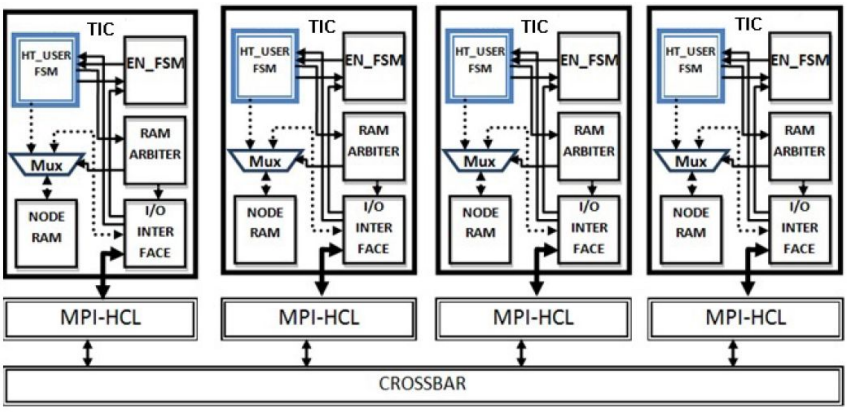

Figure 4.1: *Architecture de la plateforme MATIP*

4.2.2 Caractéristiques de MATIP en tant que FPGA Overlay

MATIP est une plateforme qui intègre le modèle d'architecture parallèle à mémoire distribuée. Sa structure en couches permet d'encapsuler tous les détails en termes de communication et d'interconnexion. De plus, grâce à son composant TIC qui est l'instance de la couche applicative, le développeur n'a qu'à décomposer l'application en tâches ou modules, identifier les échanges entre les modules de l'application et établir un scénario des communications, implémenter et intégrer chaque tâche dans le composant TIC. Et le scénario de communication est implémenté dans ce composant par l'invocation des MVPs [1](MPI_Put, MPI_Get, MPI_Send, MPI_Receiv, etc.) très proches des primitives MPI.

De plus, dans MATIP, la primitive MPI-2 RMA pour le chargement dynamique de tâches (*MPI_Comm_spawn*) a été implémentée. Cette primitive permet à un processus/tâche MPI de générer un certain nombre d'instances du processus MPI nommé. MATIP prend en compte deux types de tâches : les tâches statiques qui sont instanciées durant toute l'exécution de l'application et les tâches dynamiques qui sont instanciées/supprimées pendant l'exécution. Cette caractéristique de MATIP permet au développeur, en fonction des contraintes et de ses besoins, d'obtenir au final une architecture à tâches fixes qui correspond à un Overlay spatialement configuré (Spatially Configured/SC) ou une architecture qui peut modifier le contenu de ses tâches ou unités fonctionnelles en cours d'exécution qui correspond à un Overlay temporellement multiplexé (Time-Multiplexed /TM).

De plus, cette plateforme est entièrement implémentée en VHDL de sorte que, après synthèse, l'application résultante peut être déployée sur n'importe quel FPGA. Le tableau 4.1 résume les caractéristiques de MATIP en tant que FPGA Overlay.

Tableau 4.1: *Les caractéristiques de MATIP en tant que superposition FPGA.*

Caractéristique	**Valeur**
Type d'applications	*Parallèle MPI*
Modèle parallèle	Mémoire distribuée
Type d'Unité Fonctionnelle	*SC & TM*
Niveau d'abstraction	*RTL*
Compatibilité FPGA	*indépendant des plateformes et des constructeurs*
Interconnexion	*NoC*
Nombre maximum d'Unités Fonctionnelles	*16*

1. MPI Vhdl Primitives

Le tableau 4.2 [2] montre le positionnement de MATIP par rapport aux autres FPGA Overlay qui traitent des applications parallèles dans l'état de l'art. Par rapport à l'état de l'art, MATIP apporte de nombreuses innovations sur les overlays de traitement parallèle mentionnés dans [173]. En effet, il permet l'intégration des PE de type accélérateur matériel . De plus, grâce à l'implémentation de la reconfiguration dynamique, il offre la possibilité d'être utilisé aussi bien comme Overlay SC que comme Overlay TM. En outre, c'est le seul Overlay de cette liste qui aborde le modèle parallèle à mémoire distribuée avec une communication via le standard MPI-2 RMA.

Tableau 4.2: *La situation de MATIP par rapport à l'état de l'art*

Couverture	**Type de PE**	**FUs SC/TM**	**Topologie**	**Communication**
DRAGON[173]	Processeur	SC	Mesh/Torus	Mémoire partagée
GRVI Phalanx[179]	Processeur RV32I	TM	Hopelite NoC	Mémoire partagée + MP
SSA[178]	FMAC	SC	Torus+Mesh	Stream
SIMD-Octavo[182]	processeur logiciel	TM	Mesh	Mémoire partagée
reMORPH[177]	CGRM	TM	Mesh 2D	Mémoire partagée
SCMA[174]	SCM	SC	2D Mesh	Mémoire partagée
MATIP[191]	**Module matériel**	**SC/TM**	**NoC**	**Mémoire distribuée**

4.3 Présentation du flot de conception HLS-MATIP

4.3.1 Principe de fonctionnemment

Nous proposons dans le cadre de ces travaux de thèse, d'utiliser les outils de synthèse de haut niveau comme environnement front-end pour la conception de l'application utilisateur et à exploiter MATIP comme architecture d'Overlay FPGA. L'approche HLS permet au concepteur de se concentrer sur les aspects algorithmiques et fonctionnels sans avoir à se préoccuper des aspects matériels. Les modules matériels synthétisés par l'outil HLS sont intégrés dans MATIP. La figure 4.2 illustre le principe de cette approche. La figure 4.3 décrit la version semi-automatique du flot de conception qui met en œuvre cette approche.

4.3.2 Conception du modèle parallèle

Comme nous l'avons dit dans la section 3.2.4, la clé du traitement parallèle est la simultanéité exploitable sur un problème qui peut être décomposé en sous-problèmes. Ces sous-problèmes sont transformés par notre approche en modules matériels qui seront déployés dans MATIP comme des tâches matérielles statiques ou dynamiques communicantes pour réaliser l'application. Décomposer et structurer ces tâches de manière à être efficace compte tenu du problème est un défi pour le développeur. C'est la raison pour laquelle dans cette méthodologie, nous recommandons une méthode qui existe dans l'état de l'art pour l'atteinte de cet objectif. Il existe plusieurs méthodes dans la littérature, celle que nous recommandons est la méthode PCAM de Foster[103], que nous avons décrite dans la section 3.2.4. Dans la suite de cette section nous allons donner des indications pour contextualiser cette méthodologie par rapport à notre flot de conception.

Le partitionnement

L'étape de partitionnement d'une conception a pour but d'exposer les possibilités d'exécution parallèle. Étant donné qu'il s'agit d'une étape de conception algorithmique, il n'ya aucun raffinement à faire ici par rapport à ce que nous avons présenté plus haut.

2. MP(Message Passing), FMAC(Floating-point Multiply Accumulate Unit), CGRM (Coarse GRained Modules), SCM (Systolic Computational Memory)

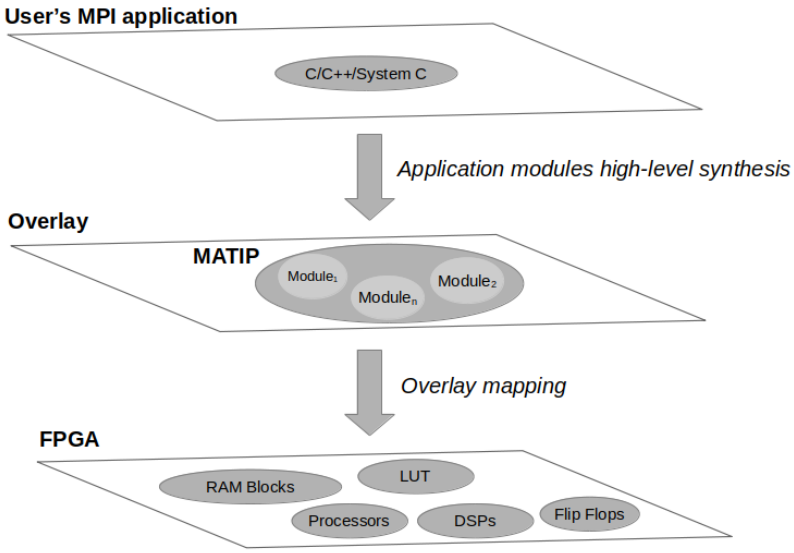

Figure 4.2: *Principe de l'approche d'Overlay MATIP*

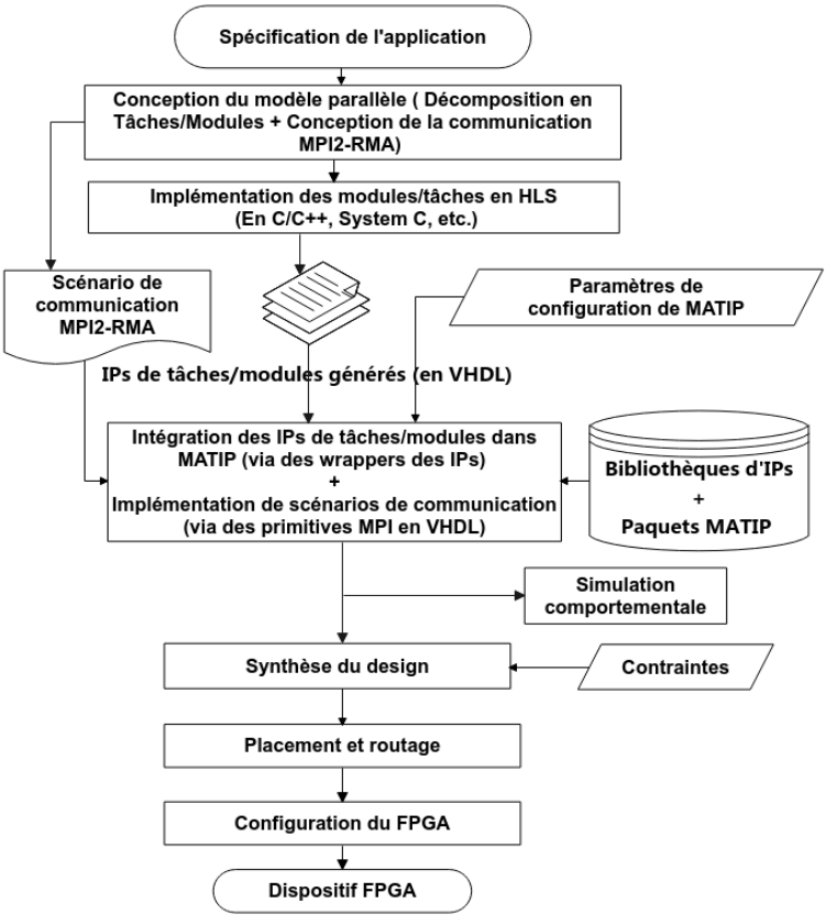

Figure 4.3: *Flot de conception de l'Overlay MATIP*

La communication

Dans cette étape, la communication nécessaire pour coordonner l'exécution des tâches est déterminée, et les structures et algorithmes de communication appropriés sont définis. La communication associée à un algorithme peut être spécifiée en deux phases. Premièrement, on définit une structure de canal qui relie, directement ou indirectement, les tâches qui ont besoin de données (consommateurs) aux tâches qui possèdent ces données (producteurs). Dans notre approche, ces canaux sont garantis par les couches d'interconnexion et de communication offertes par MATIP. Ensuite, on spécifie les messages qui doivent être envoyés et reçus sur ces canaux. Le développeur doit à cet effet identifier les caractéristiques des données transmises sur son message en termes de format (caractère, entier signé ou non signé, nombre réel virgule fixe ou virgule flottante, etc.), et de nombre d'éléments dans un message (lorsque l'on transmet un vecteur de données par exemple). Par ailleurs seul le protocole de communication MPI-2 RMA est pris en compte ; cela signifie donc que le développeur devra choisir une instruction de communication qui correspond à la situation parmi celles qui existent dans le standard MPI-2 RMA (MPI_Put, MPI_Get, etc.). À l'issue de cette étape, conformément à l'architecture MATIP, le développeur obtient un modèle de communication qui reflète la structuration des canaux de communications entre tâches et les traduits en des opérations de communication MPI-2 RMA.

L'agglomération

Dans l'étape d'agglomération, on réexamine les décisions prises dans les phases de partitionnement et de communication en vue d'obtenir un algorithme qui s'exécutera efficacement sur MATIP. En particulier, on se demande s'il est utile de combiner, ou d'agglomérer, les tâches identifiées par la phase de partitionnement, de manière à obtenir un plus petit nombre de tâches, chacune de plus grande taille. On détermine également s'il est utile de répliquer les données et/ou les calculs.

Pour le prototypage et l'optimisation rapide du modèle parallèle, la version logicielle de ce modèle peut être implémentée en langage de haut niveau (C/C++) et déployée sur environnement multiprocesseur. Cela permet notamment de valider la faisabilité du modèle de communication avec le standard MPI-2 RMA et l'efficience du modèle parallèle de l'application comparé à la version séquentielle. Les codes ainsi obtenus pourront être réutilisés pour les implémentations HLS des différentes tâches ou modules. L'optimisation (agglomération) et la validation de l'efficacité du modèle parallèle peut se faire dans cette partie de prototypage à l'aide des plateformes, outils et librairies d'instrumentation et mesure des programmes parallèles MPI. Le développeur peut alors prendre des décisions de partitionnement et d'agglomération sur la base de l'analyse des données des performances fournies par ces différents outils. Dans la littérature, il existe plusieurs outils de cette catégorie ; parmi les plus connues, on distingue : Score-P[114], AIMS (Automated Instrumentation and Monitoring System)[104], nupshot distribué avec MPE[105], Pablo[106], SvPablo[107], Paradyn[108], VAMPIR[109], etc.

Le mappage

Dans cette quatrième et dernière étape du processus de conception d'un algorithme parallèle, on spécifie où chaque tâche doit s'exécuter. Dans l'environnement MATIP, il s'agit de tâches matérielles. Dans cette étape il sera décidé si les tâches seront déployées comme tâches statiques ou comme tâches dynamiques. Cette étape est à la seule charge du développeur qui devra utiliser les contraintes, les exigences fonctionnelles et non fonctionnelles, et la connaissance de l'application pour mieux décider.

4.3.3 Implémentation HLS des tâches matérielles

Après conception du modèle parallèle, on dispose d'un ensemble de tâches correspondant aux différents sous problèmes susceptibles de s'exécuter de façon simultanée. La phase d'implémentation par synthèse de haut niveau consiste à isoler et implémenter la fonction de traitement inhérente à cette tâche sans la prise en compte des aspects de communication, synchronisation, lecture et écritures des données. En d'autres termes, l'attention est uniquement accordée à la fonction de traitement inhérente à la tâche.

Ces fonctions de traitement doivent être implémentées de manière à optimiser au maximum un ensemble de contraintes dont les plus connues sont : la latence, la consommation en ressources du FPGA, la fréquence de fonctionnement, etc. A cet effet il existe des techniques courantes pour optimiser les résultats liés à ces différentes contraintes. La technique la plus courante est l'usage des pragmas réservés par chaque outil et ce suivant les caractéristiques intrinsèques de l'algorithme de la fonction de traitement en cours d'implémentation. Comme cela a été vu dans le chapitre 3, on distingue : les pragmas de déroulage de boucle, de fusion de boucles, de pipelining automatique, de synthèse hiérarchique, le partitionnement de mémoire, etc.

Pour le contrôle, la prise en compte des entrées et la livraison des sorties, chaque fonction de traitement est synthétisée en IP disposant d'interfaces constituées de multiples signaux/bus. Ces signaux sont de différents types, des signaux d'entrée pour l'envoie des données/paramètres d'entrée à l'IP, des signaux de sortie pour la réception des résultats de traitement renvoyés par l'IP, et les signaux de contrôle pour la gestion du fonctionnement de l'IP et l'observation de son état interne. Les outils Xilinx[158], par exemple, prennent en charge les paradigmes d'interface de mémoire, de flux et de registre, chaque paradigme prenant en charge différents protocoles d'interface pour communiquer avec le monde extérieur, comme le montre le tableau 4.3. Selon les caractéristiques des entrées et sorties prises en compte par la fonction de traitement, certaines interfaces sont mieux adaptées que d'autres. Par exemple, dans le cas des outils Xilinx, les interfaces par défaut sont définies par le type d'argument C dans la fonction principale, et le paradigme par défaut, comme le montre le tableau 4.4.

Les modules implémentés sont exportés sous forme d'IPs matérielles synthétisées en VHDL. Il est aussi possible, à la place d'une implémentation HLS, d'utiliser une IP existante décrite en VHDL.

Tableau 4.3: *Types d'interfaces supportées par Vivado HLS*

Paradigme	Description	Types d'interface
Mémoire	L'IP accède aux données par le biais de mémoires telles que DDR, HBM, PL-RAM/BRAM/ URAM et protocoles d'interface pris en charge	ap_memory, BRAM, AXI4 Memory Mapped (m_axi)
Flux	L'interface des données prises en charge est diffusée en continu dans l'IP à partir d'une autre source de diffusion en continu, comme un processeur vidéo ou un autre noyau, et peut également être diffusée en continu à partir de l'IP.	ap_fifo, AXI4-Stream (axis)
Registre	L'IP accède aux données par le biais d'interfaces de registre réalisées par des lectures et des écritures de registre.	ap_none, ap_hs, ap_ack, ap_ovld, ap_vld, and AXI4-Lite adapter (s_axilite).

4.3.4 Intégration des tâches et implémentation du scénario de communication

Après implémentation HLS des différents modules ou fonctions de traitement inhérents au modèle parallèle, l'étape suivante est l'intégration de ces derniers dans l'architecture MATIP. Cela

Tableau 4.4: *Interfaces par défaut utilisées dans vivado HLS*

Type d'argument C/C++	Paradigmes pris en charge	Paradigme par défaut	Protocole d'interface par défaut		
			Entrée	Sortie	Entrée-sortie
Variable scalaire (passage par valeur)	Registre	Registre	ap_none	N/A	N/A
Tableau	Mémoire, Flux	Mémoire	ap_memory	ap_memory	ap_memory
Pointeur	Mémoire, Flux, Registre	Registre	ap_none	ap_vld	ap_ovld
Référence	Registre	Registre	ap_none	ap_vld	ap_vld
hls : :stream	Flux	Flux	ap_fifo	ap_fifo	N/A

se fait en suivant un flot de conception basé sur l'utilisation des IPs dans une architecture décrite en VHDL. Le principe de fonctionnement d'une telle approche est le suivant : les différentes IPs correspondant aux fonctions de traitement des nœuds du modèle parallèle sont importées dans la base de données locale du projet lié à l'architecture en cours de conception ; Par la suite on génère une interface d'instanciation de chaque IP ainsi importée appelée *IP Wrapper* ; une fois cela fait, il reste plus qu'à instancier ces modules au sein de l'architecture MATIP.

La phase d'intégration des modules des tâches consiste à suivre le flot de conception proposé par Gamom et al.[193]. Tout commence par la configuration de MATIP. Pour générer l'architecture et l'infrastructure de communication, on configure les paramètres de la plateforme MATIP en spécifiant le nombre de tâches statiques et dynamiques suivant la structuration du modèle parallèle conçu par le développeur. Cette configuration se fait dans le fichier *"HCL_Arch_conf.vhd"* (voir en annexe A.1) de la plateforme MATIP. Soit N le nombre total de tâches (nœuds de calcul) prévus dans le modèle parallèle, on configure la constante *NOC_SIZE* qui représente le nombre de tâches matérielles à instancier dans MATIP (voir en annexe A.1) en lui attribuant le nombre pair supérieur ou égal le plus proche de N. Soit S le nombre de tâches statiques prévus dans le modèle parallèle, on configure la constante *STATIC_HT* (en annexe A.1) en lui attribuant la valeur S. Si des tâches dynamiques sont prévues c'est-à-dire $S \neq N$ alors, la constante *DYN_ALLOWED* (qui renseigne sur le fait que les tâches dynamique sont autorisées ou non, voir en annexe A.1) prend le signal constant *'1'* à défaut elle prend la valeur *'0'*. Ainsi, si les tâches dynamiques sont prévues, le nombre de tâches dynamiques créées par l'architecture sera $D = N - S$.

Une fois l'architecture configurée, l'étape suivante est l'intégration des IPs dans la TIC de MATIP. Cela revient à instancier les prototypes des IPs générés dans les différents Wrappers. Ces instanciations se font dans les fichiers *"HT_stat.vhd"* (voir l'annexe A.2) et *"HT_dyn.vhd"* (voir l'annexe A.3) suivant que les tâches sont respectivement statiques ou dynamiques. Pour éviter de dupliquer les tâches de traitement dans les différents nœuds générés par l'architecture, on peut faire recours à l'instanciation conditionnelle de ces IPs en fonction des rangs des tâches.

Après intégration des IPs, il revient par la suite de décrire à l'aide d'une machine à état le scénario de communication de chaque tâche matérielle en termes de lecture, écriture, envoie, réception et traitement des données, ainsi que la synchronisation entre tâches. La logique de communication se fait selon le scénario de communication conçu par le développeur pendant la phase de conception du modèle parallèle. Pour l'implémentation de ce scénario de communication, il est prévu dans la plateforme MATIP une API MPI constituée de primitives MPI-2 RMA appelées MVP (MPI VHDL Primitives). L'implémentation du scénario de communication revient donc à l'usage de ces primitives dans une machine à état, et ce de façon conditionnelle en fonction du rang de la tâche comme cela se fait dans la programmation MPI sur environnement logiciel.

4.3.5 Tests et Validation (Simulation)

Cette phase se fait à l'aide d'un simulateur de circuit niveau RTL supportant le VHDL. Dans la littérature, il en existe plusieurs ; parmi les plus connus on distingue : ModelSim, Vivado Simulator, QuestaSim, etc. Peter Rössler et al.[194], font une revue assez exhaustive des simula-

teurs RTL VHDL qui sont utilisés. Il y a plusieurs niveaux de test et validation à réaliser par le développeur lors de la conception d'une application parallèle avec notre approche.

Le premier niveau de test est celui de l'IP de traitement liée à chaque tâche. En effet la fonction de traitement implémentée doit être testée et validée par simulation RTL non seulement pendant la phase d'implémentation HLS (cosimulation) mais aussi pendant son intégration dans MATIP ; car, deux aléas peuvent alors faire surface. Le premier aléa peut être dû à un mauvais paramétrage de l'IP pendant son intégration ; Et le second pourrait être issu d'une mauvaise manipulation des interfaces d'entrée/sortie et de contrôle offertes par la dite IP. Le développeur doit donc s'assurer que la configuration de l'IP et la manipulation des interfaces sont correctes.

Le second niveau de test consiste à s'assurer du bon fonctionnement de la logique de communication de chaque tâche matérielle en termes de séquences d'actions implémentées via une machine à état en vue de réaliser le modèle parallèle conçu par le développeur. Cela comprend : la lecture, l'écriture, l'envoie, la réception et le traitement des données de la tâche matérielle. Pour ce faire, il est question non seulement d'analyser les signaux des différentes variables présentes dans le système, mais aussi d'examiner pas à pas le contenu de la mémoire locale de chaque tâche, pour s'assurer de la cohérence des données tout au long de l'exécution.

Le troisième niveau de test revient à s'assurer du fonctionnement cohérent de toute l'architecture parallèle. Il est question de s'assurer du calcul correct de toute l'application parallèle, conformément au modèle conçu par le développeur. En outre, il peut s'agir ensuite de faire une évaluation des performances en temps de toute l'architecture, et ceci en simulation. Il faut noter que l'architecture peut être maintenue (corrigée) au cas où un dysfonctionnement est rencontré à n'importe quel niveau de test.

4.3.6 Synthèse et déploiement de l'architecture

Cette phase de notre approche est gérée par les outils de type EDA de façon automatique. Elle passe par les étapes : de synthèse, placement, routage et configuration du FPGA.

La synthèse

La synthèse (ou synthèse logique) est le processus par lequel une forme abstraite (une description HDL) du comportement du circuit est transformée en une implémentation de la conception en termes de portes logiques et d'interconnexions. Le résultat est généralement une liste de réseaux et divers rapports. Dans ce contexte, une "liste de réseaux" décrit la connectivité de la conception électronique, en utilisant des instances, des réseaux et, éventuellement, certains attributs. Il existe plusieurs formats de liste d'interconnexion propriétaires, mais la plupart des synthétiseurs peuvent générer le format EDIF (Electronic Design Interchange Format), qui est un format neutre pour le stockage des listes d'interconnexion électroniques. Les fournisseurs de FPGA ont leurs propres synthétiseurs (Xilinx XST, Altera Quartus II Integrated Synthesis) mais les principaux fournisseurs d'EDA ont des synthèses pour FPGA (Precision de Mentor Graphics et Synplify de Synplicity) qui peuvent être intégrées dans les outils EDA FPGA.

Le processus de synthèse effectue plusieurs optimisations indépendantes de la cible (simplification de la logique, affectation des états, etc.), mais les outils de synthèse prennent également en compte les technologies de la cible et effectuent des optimisations dépendantes de la cible.

Les optimisations disponibles dépendent du synthétiseur mais les optimisations les plus typiques sont présentes dans tous les synthétiseurs. Les optimisations typiques concernent la réduction de la surface, l'optimisation de la vitesse, la faible consommation d'énergie et la fréquence cible de l'ensemble de la conception.

Les optimisations de synthèse sont contrôlées globalement (pour la conception complète) ou localement pour chaque partie du code HDL. L'optimisation globale peut être introduite dans l'environnement intégré (puis traduite sous forme de commutateurs en ligne de commande ou dans un fichier supplémentaire), ou dans un fichier de contraintes spécifique. L'optimisation

locale peut être spécifiée dans un fichier de contraintes ou intégrée dans le code HDL. Le fichier de contraintes de synthèse est, typiquement, un fichier texte brut ayant une syntaxe différente, selon l'outil. Xilinx par exemple utilise le xcf (Xilinx Constraint File)[195]. Le principal avantage de l'utilisation de fichiers de contraintes (générés par une interface graphique ou saisis dans un éditeur de texte) est qu'il rend votre code source plus portable et sépare la fonctionnalité décrite dans le HDL du détail de synthèse.

Implémentation (Mapping, Placement et Routage)

L'étape de l'implémentation est faite par un outil spécifique au fournisseur qui place et route principalement le design dans le dispositif cible. Les entrées de l'implémentation sont les netlists générées par la synthèse et les contraintes d'implémentation du design. La sortie est une netlist propriétaire placée et routée (fichier ncd chez Xilinx, une représentation de base de données interne chez Altera) et plusieurs rapports résumant les résultats. En général, l'implémentation du design utilise des contraintes de temps et de surface. Les contraintes d'implémentation sont des instructions données aux outils d'implémentation FPGA pour diriger le mappage, le placement, le routage, le timing ou d'autres directives pour les outils d'implémentation. Les contraintes d'implémentation sont placées dans un ou plusieurs fichiers de contraintes, mais elles peuvent exister dans le code HDL ou dans un fichier de contraintes de synthèse et être propagées pour l'implémentation. Les fichiers de contraintes sont en texte brut mais il existe plusieurs outils graphiques qui facilitent l'édition de ces contraintes en évitant la nécessité de connaître la syntaxe exacte.

La contrainte de timing comprend la définition de l'horloge, les exigences de timing d'entrée et de sortie et les exigences de chemins combinatoires. La création de contraintes globales pour un design est le moyen le plus simple de fournir une couverture des connexions contraignables dans un design, et de guider les outils pour qu'ils répondent aux exigences de timing pour tous les chemins. Par exemple, étant donné une contrainte de fréquence de 100 MHz, nous contraignons chaque chemin combinatoire dans le design. Néanmoins, le concepteur doit parfois relâcher certaines contraintes globales et informer les outils d'implémentation que certains chemins peuvent prendre plus de temps. Les contraintes de placement indiquent aux outils d'implémentation où placer les éléments logiques dans le FPGA (pattes d'E/S, Flip-flops, ROMs, RAMs, LUTs, etc.). Comme chaque composant du design porte un nom unique, vous pouvez utiliser ce nom pour assigner à une région du FPGA où placer ce composant. Une contrainte de placement représentative, toujours utilisée dans un design, est la position des broches d'entrée/sortie.

Le tableau 4.5 résume notre proposition de flot de conception productive d'applications parallèles sur MATIP en termes d'étapes chacune caractérisée par des méthodes, outils, langages et niveau d'abstraction. Pour faire une preuve de concept de cette approche nous l'avons déroulé pour la mise en œuvre de l'application d'extraction des caractéristiques pour la reconnaissance des langues à tons dont nous avons fait une mise en œuvre parallèle MPI-2 RMA sur environnement logiciel (voir section 3.3). Dans la prochaine section, nous faisons un rapport détaillé de la mise en œuvre sur MATIP.

4.4 Mise en œuvre sur MATIP d'un modèle parallèle d'un algorithme d'extraction des caractéristiques pour la reconnaissance des langues à tons

Comme nous l'avons vu ci-dessus, notre méthodologie de mise en œuvre d'applications parallèles sur MATIP consiste en 6 principales étapes à savoir : la spécification fonctionnelle, la conception du modèle parallèle, l'implémentation HLS des tâches parallèles, l'intégration de

Tableau 4.5: *Tableau récapitulatif du flot de conception HLS+MATIP*

Etape	Méthodes	Outils	Langages	Niveau d'abstraction
1. Spécification fonctionnelle	Méthodes de spécification(ex :TAD, SREM-SREP, etc.)	Logiciels de spécification/modélisation	Langage de spécification, ex : Langage Z	–
2. Conception du modèle parallèle	Methode PCAM[103], Prototypage, Instrumentation MPI	Compilateur (Ex : Open MPI[113]), Outil d'instrumentation MPI (ex : Score-P[114] et ParaProf[115])	C, C++, Python, etc.	Haut niveau (algorithmique)
3. Implémentation HLS	Synthèse de haut niveau(HLS)	Outils HLS (ex : Vivado HLS)	C, C++, System C, etc.	Haut niveau (HLS)
4. Intégration des tâches et Implémentation du scénario de communication	Conception basée sur l'usage des IPs, Description HDL	MATIP[191], Outils RTL (Ex : Vivado, Modelsim, etc.)	VHDL	Niveau RTL
5. Tests et Validation	Simulation, Conception des Testbench	Simulateur RTL (Ex : Vivado simulator, Questasim, etc.)	VHDL	Niveau RTL
6. Synthèse et déploiement	CAO/EDA	Outils EDA (Ex : Xilinx XST)	–	Niveau RTL

celles-ci et l'implémentation du scénario de communication, les tests et validation, et enfin la synthèse et déploiement sur FPGA. Dans la section 3.3, nous avons déjà réalisé les deux premières étapes du flot de ce flot de conception. La figure 3.4 décrit le modèle parallèle que nous allons utiliser dans la mise en œuvre sur système embarqué et la figure 3.5 représente le modèle de communication à considérer. Dans la suite de cette section nous faisons un rapport de cette mise en œuvre.

4.4.1 Mise en œuvre HLS des modules des tâches parallèles

Pour réaliser l'implémentation HLS, nous avons exploité le code C++ obtenu de l'implémentation du modèle parallèle sur environnement logiciel. Dans chaque tâche, nous avons extrait une fonction de traitement qui doit être générée sous forme d'IP matérielle. Le code C++ réutilisé a subi de multiples transformations afin de rendre les fonctions de traitement des différentes tâches synthétisables. Cela a consisté d'une part à trouver d'autres alternatives aux constructions C/C++ non prises en charge par le synthétiseur HLS. Parmi les constructions non prises en charge on distingue : les appels systèmes, l'utilisation dynamique de la mémoire, les manipulations sur les pointeurs, le casting général des pointeurs, les tableaux de pointeurs, les pointeurs de fonction, les fonctions récursives, les comportements indéfinis (généralement induits par les structures de contrôle de type *if ... else*)[196]. Pour l'optimisation des IP matérielles en termes de temps de réponse, nous avons utilisé des pragma d'optimisation en l'occurrence les pragmas *HLS PIPELINE* et *HLS DATAFLOW*, car les traitements se faisant sous forme de boucles sur des vecteurs issus des échantillons audio. Les implémentations HLS ont été faites avec l'outil Vivado HLS de chez Xilinx et les IP matérielles ainsi générées ont été exportées en langage VHDL pour être intégrées dans MATIP. L'extrait de code A.4 présente les types et les pragmas utilisés dans la fonction de Hanning.

4.4.2 Intégration des tâches et implémentation du scénario de communication

Après l'implémentation HLS l'étape suivante est l'intégration des tâches matérielles dans MATIP, suivi par l'implémentation du scénario de communication adopté dans le modèle parallèle. Pour réaliser cette intégration, nous avons importé MATIP sous forme de projet Vivado. Puis, nous avons importé les différentes IP correspondant aux différents modules de traitement

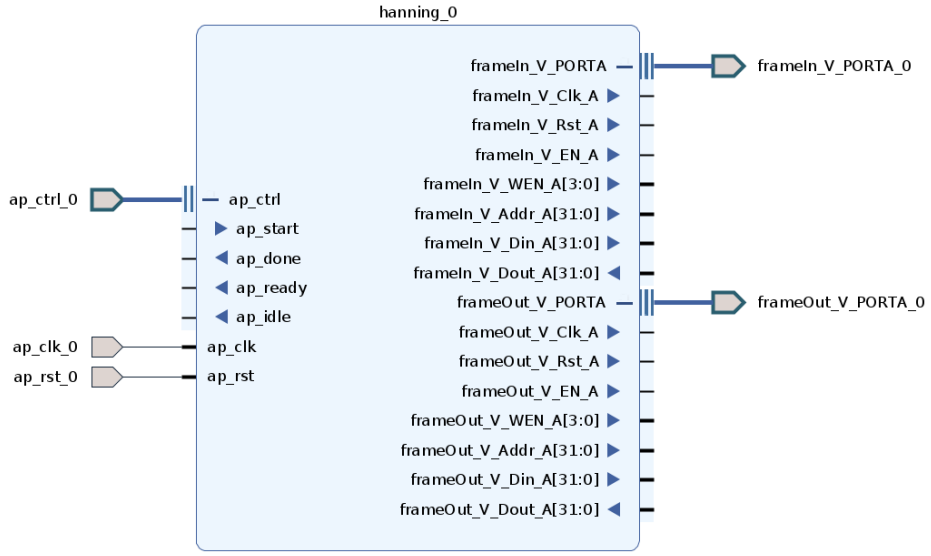

Figure 4.4: *IP matérielle de la fonction de Hanning exportée depuis Vivado HLS*

des tâches parallèles. Ensuite, nous avons intégré ces IPs dans MATIP grâce au *HDL Wrapper*[3]. Les différentes tâches ont été intégrées dans le TIC à travers les fichiers HT_stat.vhd, car toutes les tâches étant statiques. L'extrait de code A.5 représente les séquences de code VHDL qui permettent d'intégrer les IPs matérielles ; chaque IP est instanciée de façon conditionnelle en fonction du rang de la tâche dans laquelle elle doit être utilisée. Cela permet d'optimiser la quantité de ressources utilisées sur le FPGA. Pour l'envoi des données et la réception des résultats dans les modules de traitement, nous avons utilisé l'interface de type RAM_1P (Interface de type RAM à port unique, pour laquelle Vivado HLS détermine s'il faut l'implémenter dans le RTL avec un bloc de RAM ou comme RAM distribuée.) offerte par Vivado HLS ; cette interface nous a permis de mieux gérer la communication avec les IPs par rapport à d'autres plus complexes dans la manipulation (à l'instar des interfaces de type axi-stream). La figure 4.4 montre le prototype de l'IP matérielle du module de traitement de la fonction de Hanning avec les interfaces de données et de contrôle. Nous avons implémenté le scénario de communication en VHDL à travers une machine à états, et en invoquant à un moment/état précis les différentes MVP offertes par MATIP pour la communication MPI-2 RMA. La figure 4.5 présente la description des machines à états des tâches T0 qui est la fusion des tâches de lecture des données et la fonction de Hanning, T1 qui correspond à la fonction FFT et qui a la même structure que les tâches omises 2, 3, 4, et 5 (réception des données de la tâche $n-1$, traitement et transmission des résultats à la tâche $n+1$).

4.4.3 Test et validation

Les tests et validation se sont faits à plusieurs niveaux. Premièrement nous avons réalisé les tests unitaires, correspondant à la validation des implémentations des différents modules de traitement en utilisant la simulation et la co-simulation offertes par Vivado HLS. Ensuite ont suivi les tests d'intégrations à l'aide des testbench et la simulation des modules intégrés à MATIP. Cela nous a permis de tester la bonne intégration des modules et de s'assurer du bon fonctionnement du scénario de communication implémenté. Enfin, une simulation globale de la plateforme MATIP avec l'application déployée nous a permis de valider le bon fonctionnement

3. Fichier HDL généré à l'aide de l'EDA utilisé pour le développement RTL, qui permet l'intégration de l'IP importée dans un autre composant VHDL.

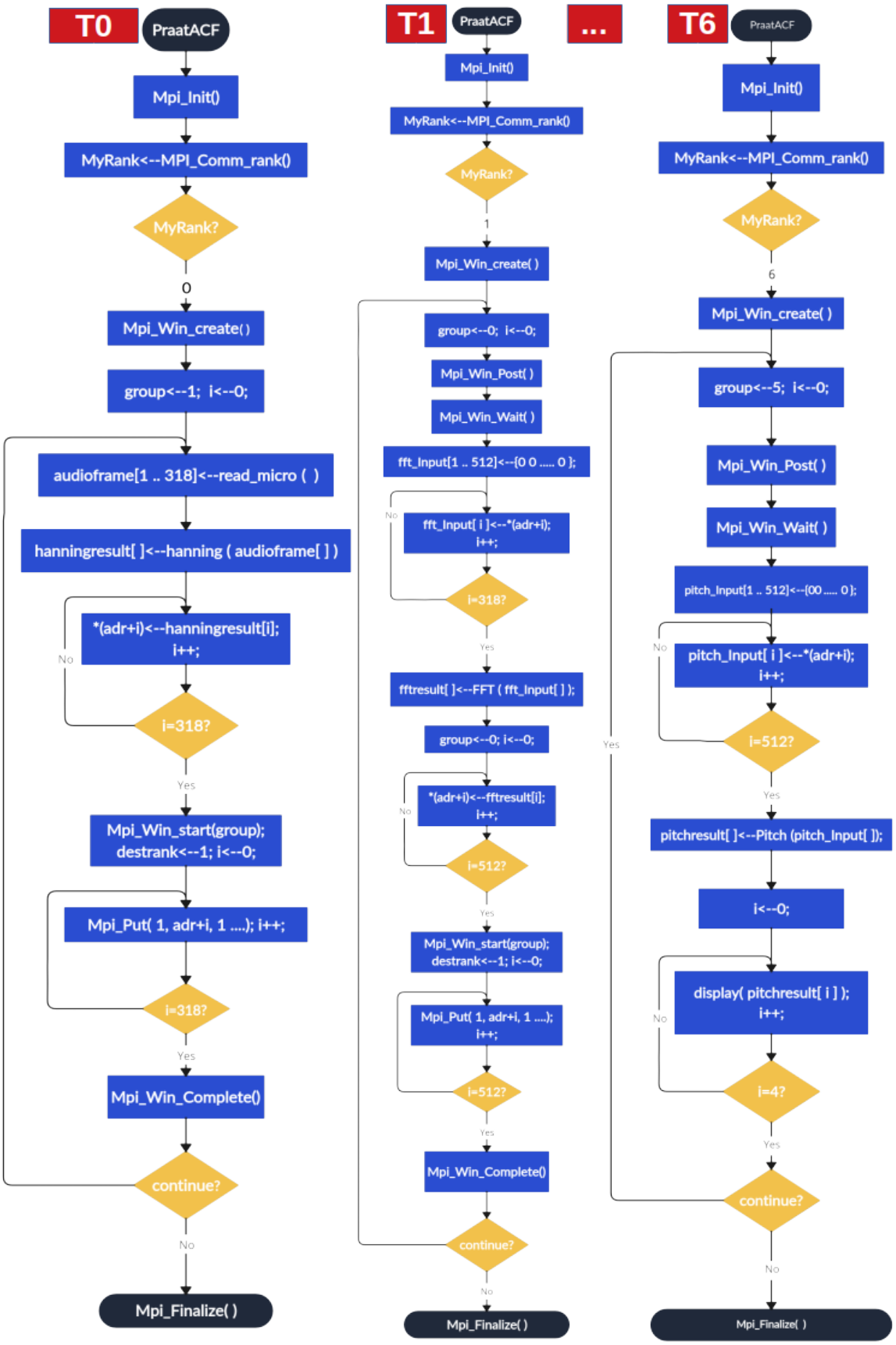

Figure 4.5: *Machines à état des tâches 0, 1 et 6. La tâche 1 a la même structure que les taches 2, 3, 4, et 5 omises sur cette figure.*

de toute l'application.

4.4.4 Synthèse et déploiement

Pour des besoins d'expérimentation, nous avons implémenté notre application en trois versions : une version parallèle MPI-2 RMA CPU, implémentée en C++ (voir section 3.3), une version parallèle MPI-2 RMA FPGA développée et intégré dans MATIP, et une version FPGA implémentée comme une IP matérielle. La version parallèle MPI-2 RMA CPU a été déboguée et compilée en utilisant l'implémentation Open MPI [113]. L'environnement de test matériel était un ordinateur de 2,2 GHz, *Intel(R) Core(TM) i7-8750H CPU*, comprenant 6 cœurs (avec 12 threads possibles parmi lesquels 7 ont été utilisés) avec 8 Go de RAM, et l'environnement de test logiciel est basé sur Ubuntu 20.04 comme système d'exploitation et OpenMPI 4.0.5 utilisé comme bibliothèque MPI (Message Passing Interface) exploitant C++ comme langage de programmation. L'exécution du programme a été faite avec l'instruction : ***mpirun - - oversubscribe***[4] ***-np 7 ./ACFMPI***.

Pour la version MATIP, nous avons déroulé notre méthodologie en utilisant les outils Vivado HLS et Vivado tel que décrit ci-avant.

La version FPGA sous forme d'IP matérielle est une implémentation naïve de l'algorithme Praat obtenue par agencement linéaire (par appel de sous fonction/composant) des différentes fonctions HLS correspondant aux différents modules. Les deux implémentations FPGA (MATIP et IP) sont cadencées à une fréquence de 100MHz. Le tableau 4.6 présente l'utilisation des ressources de la version IP matérielle, des modules d'application et celle de la version MATIP sur le FPGA *Xilinx xc7a100tcsg324-1*. En termes de ressources, la version MATIP consomme plus de ressource que l'IP matérielle obtenue par synthèse de haut niveau. Cette différence peut s'expliquer par le fait que MATIP en soi dispose d'un ensemble de composants qui ont une empreinte considérable en termes de consommation en ressources en dehors de celle induite par les tâches matérielles déployées.

Tableau 4.6: *Utilisation des ressources des modules de l'application et de l'architecture HLS+MATIP sur le FPGA Xilinx xc7a100tcsg324-1.*

	LUT	LUTRAM	FF	BRAM	DSP
Hanning	426 (0.67%)	0 (0.0%)	242 (0.19%)	2.5 (1.85%)	2 (0.83%)
FFT	13575 (21.41%)	247 (1.30%)	8071 (6.37%)	8.5 (6.30%)	101 (42.08%)
$\mid X \mid^2$	210 (0.33%)	0 (0.0%)	265 (0.21%)	0.5 (0.37%)	8 (3.33%)
iFFT	14047 (22.16%)	309 (1.63%)	9180 (7.24%)	8.5 (6.30%)	93 (38.75%)
Normalisation	1444 (2.28%)	0 (0.0%)	697 (0.55%)	0.5 (0.37%)	2 (0.83%)
Détection du pitch	17302 (27.29%)	110 (0.58%)	12008 (9.47%)	1.5 (1.11%)	174 (72.50%)
IP matérielle	38626 (60.92%)	96 (0.51%)	16229 (12.80 %)	15 (11.11%)	234 (97.50%)
Intégration à MATIP	51384 (81.05%)	376 (1.98%)	79381 (62.60%)	19.5 (14.44%)	0 (00.0.0%)

4.4.5 Expérimentations et résultats

Notre expérience a été réalisée sur un corpus de 8 paires minimales de mots de la langue *Kóló* enregistrés avec *Audacity*[5] à une fréquence de 8000 Hz.

En guise de rappel pour valider notre implémentation, nous avons évalué le pourcentage de F_0 bien estimés (F_0 pour lesquels le pourcentage d'erreur entre le résultat de notre implémentation et celui produit par le logiciel Praat est inférieur à 10%). La précision obtenue pour les différents enregistrements est donnée dans le tableau 3.2.

4. L'option - - *oversubscribe*, indique que le nombre de processus à attribuer à un nœud dans une allocation est supérieur au nombre de slots dont dispose ce nœud. Cela indique au système d'exploitation de manager l'exécution des processus entre les différents nœuds disponibles.

5. https ://www.audacityteam.org/

Pour évaluer les performances de notre modèle et valider la pertinence de l'Overlay MATIP, nous avons exécuté le calcul de 221 fenêtres sur la version CPU, sur l'IP matérielle et la version des tâches matérielles intégrées dans MATIP. Le protocole de mesure a consisté à prélever à chaque fois la date de sortie (en nombre de cycles) du résultat de chaque fenêtre ; les fenêtres étant envoyées l'une après l'autre sur les différentes implémentations de la chaîne de traitement. Ces mesures de latence en nombre de cycles révèlent que l'implémentation sur MATIP est au moins 7 fois plus efficace que celle réalisée sur le CPU, tandis que la version de l'IP matérielle est au moins 2,7 fois plus efficace. La figure 4.6 montre la courbe d'évolution de la latence en nombre de cycles des trois implémentations en fonction du nombre de fenêtres exécutées. Et la figure 4.7 montre les courbes d'évolution du taux d'accélération obtenu à partir du rapport entre la latence de l'implémentation de la version CPU et celle des versions de l'IP matérielle et des taches parallèles dans MATIP. Les deux courbes ont des asymptotes horizontales $y = 7$ pour la version sur MATIP et $y = 2,7$ pour l'IP matérielle. Ces asymptotes donnent des informations sur les valeurs d'accélération lorsque le nombre de fenêtres tend vers l'infini. La forte valeur d'accélération identifiée au début de l'exécution est due à une forte consommation en temps de la primitive *MPI_ Win_ create* dans la version logicielle pourtant exécutée une seule fois dans chaque tâche logicielle.

Pour comprendre ces résultats nous avons identifié quelles sont les parties du système parallèle sur lesquels ces gains sont obtenus. A cet effet, nous avons mesuré pour l'ensemble des échantillons traités la latence moyenne cumulée sur les différentes parties du traitement à savoir : *Wincreate* (Crée une fenêtre MPI pour une communication unilatérale.), *Winpost*(Démarre une époque d'exposition de fenêtre MPI RMA), *Winwait* (Termine une époque d'exposition MPI-2 RMA commencée avec Winpost), *Compute*(Fonction de traitement interne (Hanning, FFT, IFFT, etc.) pour chaque tâche appelée *Traitement* dans la figure 3.5), *Put data* (Envoie des données avec MPI_Put), et *Wincomplete* (Termine une opération MPI-2 RMA). La partie *Wincreate* (retirée du graphique[6]) est très gourmande en temps sur le CPU mais n'est exécutée qu'au lancement des tâches, d'où le taux d'accélération élevé sur la figure 4.7 qui chute rapidement. Comme nous pouvons le constater, dans ces différents segments de traitement (voir figure 4.8), nous avons de meilleures performances sur la version MATIP par rapport à la version CPU.

4.5 Discussion

Les résultats ci-dessus (voir section 4.4.5) montrent que notre approche HLS+MATIP est une approche d'Overlay FPGA efficace pour la conception d'applications MPI sur FPGA en termes d'accélération par rapport à une version CPU. Cet Overlay permet aux nouveaux utilisateurs d'utiliser le FPGA avec une philosophie d'un programme MPI purement logiciel. La compatibilité de MATIP avec l'approche HLS est un grand atout pour plus de productivité dans la conception d'applications. Comparé à SDMPSoC[168] (voir Tableau 4.7), MATIP met en œuvre la communication unilatérale connue pour être plus efficace que la communication bilatérale. De plus, réalisé en VHDL, MATIP est compatible avec tous les FPGAs et ASICs. Il admet des PEs reconfigurables, d'où sa flexibilité. Son évolutivité dépend du réseau de la couche d'interconnexion.

Tableau 4.7: *SDMPSoC Vs. MATIP Qualitative comparison*

	Com. MPI	**Type de FPGA**	**Types de PEs**	**Flot de conception**	**PEs reconfigurables**	**Scalabilité**
SDMPSoC	bilatérale	Xilinx	MB & Modules HW	Automatique	Non	Limité par les ressources
MATIP	unilatérale	Tous	Modules HW	Semi-Automatique	Oui	Limité par le NoC

6. Valeur, version CPU : 481122826 cycles ; version MATIP : 264858 cycles

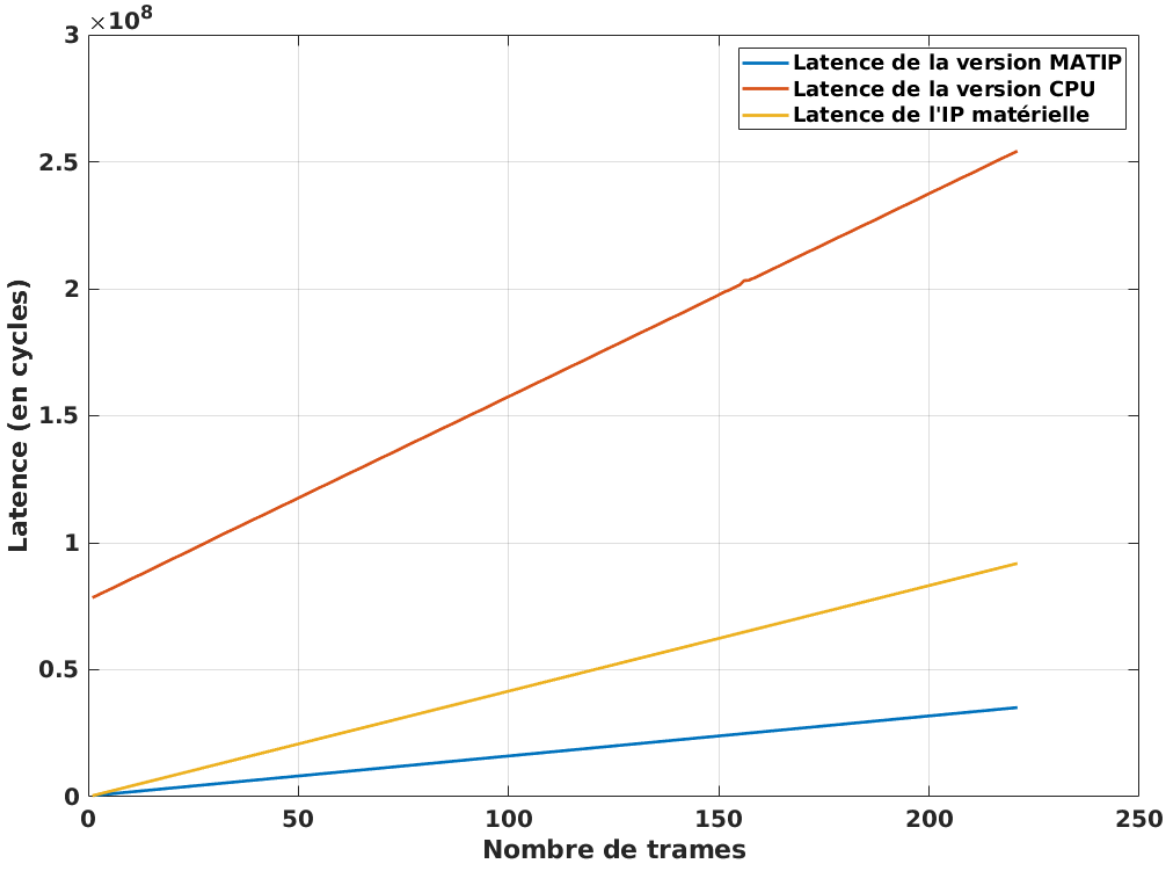

Figure 4.6: *Courbe d'évolution de la latence en nombre de cycles de l'implémentation CPU, la version d'IP matérielle et la version MATIP.*

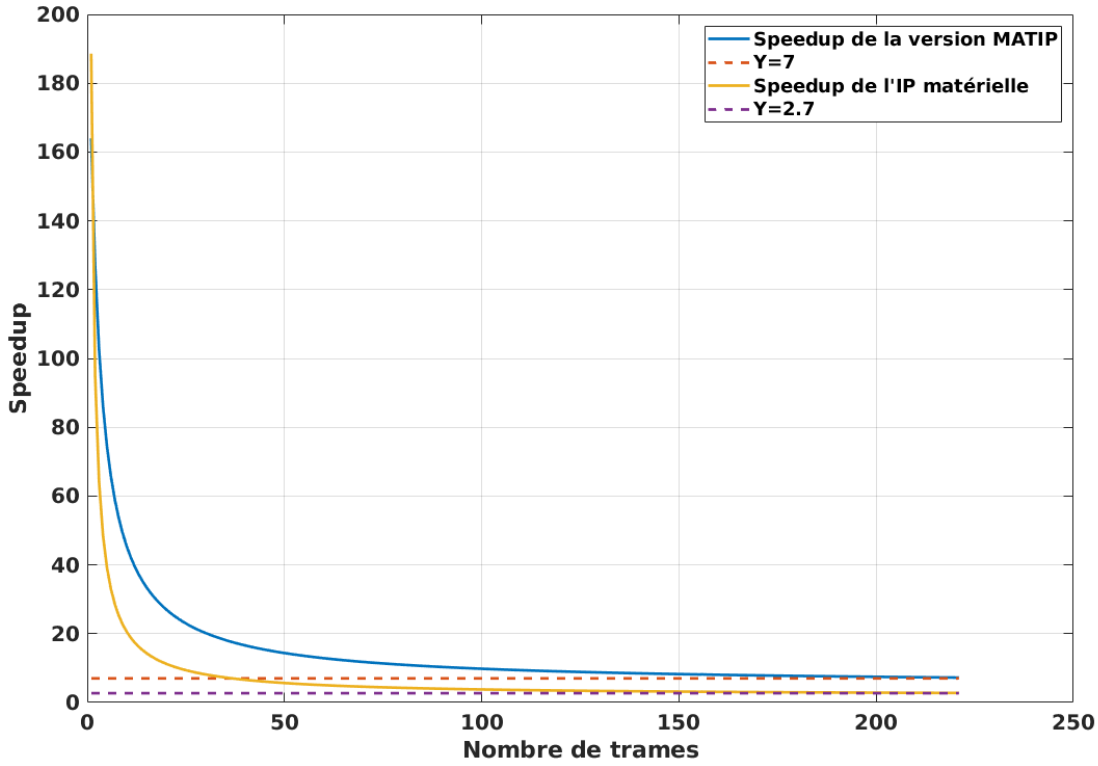

Figure 4.7: *Accélération de la version MATIP vs. Accélération de la version d'IP matérielle calculée par rapport à l'implémentation CPU.*

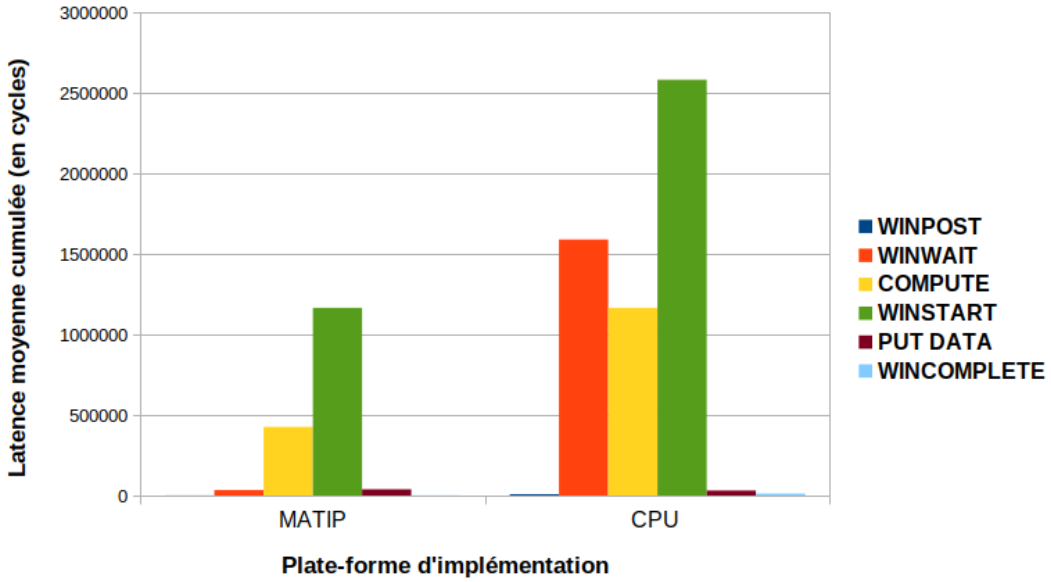

Figure 4.8: *Histogrammes d'analyse comparative des parties internes d'exécution des tâches sur les implémentations MATIP et CPU*

Par rapport à notre objectif de départ qui est de mettre en œuvre sur système embarqué une application de reconnaissance vocale temps réel, cette approche nous a permis d'accélérer le module d'extraction des caractéristiques. De façon plus générale, un composant logiciel accéléré par parallélisation avec notre approche, doit pouvoir communiquer avec d'autres modules ou d'autres systèmes. Il est prévu dans la version initiale de MATIP des interfaces GPIO pour communiquer avec l'extérieur, mais cela ne peut pas permettre une communication haut débit avec d'autres systèmes et constitue un goulot d'étranglement. MATIP nécessite à cet effet une infrastructure qui puisse permettre non seulement la communication avec d'autres systèmes, mais aussi la mise en œuvre d'applications parallèles par co-conception matérielle logicielle. Dans le prochain chapitre nous proposons un modèle d'extension de MATIP pour la permettre la communication avec d'autres systèmes parallèles sur puce.

4.6 Résumé du chapitre

La parallélisation d'applications sur MP-RSoC est de plus en plus un besoin important dans la communauté des concepteurs d'applications embarquées sur puce du fait des contraintes de temps réel auxquelles fait face ce type d'application. Compte tenu de la complexité de conception et de mise en œuvre d'application parallèles sur puce, il est nécessaire de proposer des méthodes et outils qui permettraient une mise en œuvre productive de ce type d'application. Dans ce chapitre, nous avons proposé une approche de conception d'applications parallèles sur MP-RSoC qui s'adosse sur la plateforme MATIP un outil de déploiement de tâches matérielles sur MP-RSoC. L'idée de cette approche est d'utiliser l'outil MATIP comme FPGA Overlay en la combinant à l'approche de synthèse de haut niveau (HLS). Ainsi pour mettre en œuvre la parallélisation d'une application sur puce, le développeur conçoit le modèle parallèle de ladite application en suivant l'approche PCAM de FOSTER et al. Ce modèle, constitué d'un ensemble de tâches susceptibles de s'exécuter de façon simultanée et d'un modèle de communication MPI associé est implémenté suivant un ensemble d'étapes. D'abord les tâches parallèles sont implémentées sous forme d'IPs par synthèse de haut niveau ; cela a un intérêt d'accroissement de productivité. Ensuite les IPs ainsi obtenues sont intégrés à l'Overlay MATIP qui offre une plateforme d'interconnexion et des services de communication sous le standard MPI-2 RMA. Après validation de la cohérence du système, l'application peut ensuite être déployée sur FPGA à travers une suite d'outils EDA.

Cette approche nous a permis de réaliser de façon productive l'implémentation du modèle parallèle d'un algorithme d'extraction des caractéristiques du signal vocal pour la reconnaissance des langues à tons. À cet effet, après des expérimentations sur un nombre suffisant de données vocales, une analyse sur les mesures de temps de réponse faite sur différentes implémentations de la même application nous permet de conclure que notre approche est efficiente, comparée à une implémentation logicielle sur processeur ainsi qu'une implémentation naïve de la même application sous forme d'IP matérielle par synthèse de haut niveau.

Afin de permettre la connectivité de MATIP à d'autres systèmes parallèles, et de rendre notre approche compatible à la co-conception matérielle logicielle ; il est nécessaire de concevoir une infrastructure de communication qui le permettrait. Le chapitre suivant, présente notre proposition d'une telle infrastructure.

Chapitre

5 Conception de XMATIP : une plateforme pour la co-conception des applications parallèles MPI-2 RMA

Sommaire

5.1 Introduction

Dans le chapitre précédent nous avons proposé une approche de conception d'applications parallèles sur système embarqué via une architecture reconfigurable de type FPGA. Cette approche nous a permis de mettre en œuvre sur MP-RSoC, la parallélisation du module d'extraction des caractéristiques pour la reconnaissance des langues à tons. Pour l'instant, notre approche n'intègre que des tâches matérielles, mais certaines applications nécessitent parfois l'usage d'un CPU pour plus de flexibilité. Nous devons, pour ce faire, concevoir un système MPI parallèle sur puce hétérogène. Le développeur dans notre proposition va spécifier a priori le partitionnement matériel logiciel des différentes tâches à travers un langage pragma voir Algorithme 2 [1] . Le système final à générer est un ensemble de deux environnements d'exécution de tâches parallèles. Les

1. où *mpi_comm_primitive* est une primitive de communication ; ex : MPI_Send, MPI_Put, etc.

deux environnements sont structurés de manière à ce que, les différentes tâches communiquent de manière transparente comme si elles étaient déployées sur un environnement d'exécution de tâches MPI homogène. Dans la littérature, tant du côté matériel que logiciel, il existe des architectures et des API permettant l'exécution de tâches parallèles MPI. On distingue par exemple MPICH et OpenMPI pour les APIs logicielles et SOC-MPI[189] et MATIP[191] pour la partie matérielle. L'objectif de ce chapitre est de proposer un modèle qui permet une communication MPI homogène matériel/logiciel sur des systèmes sur puce hétérogènes. L'intérêt d'un tel modèle est de faciliter le codesign matériel/logiciel et la co-synthèse d'applications MPI. Les sections suivantes présentent le modèle que nous proposons dans ce but.

5.2 L'idée de base

L'application est décrite par un modèle parallèle MPI dans un unique langage de haut niveau. Chaque tâche est mappée sur un environnement d'exécution qui est soit matériel soit logiciel ; on parle alors de tâches matérielles et logicielles. Après une synthèse de haut niveau, les tâches matérielles sont synthétisées en éléments processeurs/accélérateurs matériels et les tâches logicielles sont compilées et exécutées sur un processeur. Dans chaque environnement, on dispose d'un ensemble de tâches qui peuvent communiquer de manière homogène. Le système hétérogène final ainsi obtenu est donc un ensemble de deux clusters hétérogènes qui doivent communiquer de manière homogène pour faciliter la co-conception ou la co-synthèse matériel-logiciel. Ce problème est donc réductible à un problème de communication de tâches MPI entre clusters hétérogènes.

Dans l'état de l'art, dans l'environnement logiciel, des approches et des modèles ont été proposés pour résoudre ce type de problème. On distingue :

— PACX-MPI (PArallel Computer eXtension)[197] qui a été conçu principalement pour permettre aux applications MPI de fonctionner sur plusieurs superordinateurs (de type systèmes de processeurs massivement parallèles) sans avoir à modifier le code source de ces applications et en exploitant pleinement le sous-système de communication de chaque machine ;

— MPICH/Madeleine III est une implémentation basée sur MPICH spécifiquement conçue pour supporter nativement plusieurs configurations de réseaux hétérogènes. Ceci est réalisé en interfaçant le niveau de canal le plus bas de MPICH avec une bibliothèque de communication multiprotocole déjà disponible appelée Madeleine[198]. Cette approche permet la réutilisation de composants logiciels facilement disponibles et empêche les changements de fonctionnalités dans le code MPICH qui pourraient provoquer des incompatibilités ;

— MPICH-G2[199] est une implémentation MPI compatible avec la grille, qui est également un portage de MPICH, construit sur des services. Le Globus Toolkit est un ensemble de composants logiciels de la collection Globus Toolkit, conçu pour soutenir le développement d'applications pour les environnements de calcul distribué à haute performance, ou "Grilles". Les services fournis par cette boîte à outils aident MPICH-G2 à prendre en charge une exécution transparente et efficace dans ces environnements hétérogènes tout en garantissant que l'hétérogénéité est gérée au niveau de l'application.

Daniel Balkanski et al. dans [200] ont réalisé une étude comparative entre ces différents modèles, et expérimentalement entre PACX-MPI et MPICH/Madeleine III. Ces auteurs concluent que les deux approches de conception PACX-MPI et MPICH/Madeleine III ont leurs forces et leurs faiblesses :

— L'approche de conception PACX-MPI peut fournir une partie significative de la performance des communications MPI locales au sein de chaque sous-cluster de manière portable. Ces API MPI locales sont typiquement des implémentations MPI très stables et hautement réglées qui fournissent généralement la meilleure performance possible pour un type donné de réseau intra-clusters. L'inconvénient est que plus les API MPI locales sous-jacentes sont rapides, plus l'overhead introduit par PACX-MPI dans chaque sous-groupe

est élevé.

— L'approche de conception MPICH/Madeleine III permet d'obtenir de meilleures performances de communication avec une grande homogénéité de communication entre les nœuds d'un sous-cluster et entre les nœuds de différents sous-clusters. Elle présente toutefois l'inconvénient de nécessiter beaucoup plus de travail pour assurer la stabilité et les performances des API MPI spécialisées sur chaque sous cluster en fonction des technologies d'interconnexion.

Dans ce travail, nous proposons une approche pour construire un modèle de communication basé sur PACX-MPI intégrant le standard MPI-2 RMA qui peut être exploité par une approche de co-conception matériel-logiciel ainsi que par une approche de synthèse de haut niveau (co-synthèse matériel-logiciel).

Le choix de PACX-MPI se justifie par le fait qu'il ne nécessite pas une grande modification des API MPI des clusters existants pour être implémenté. Le fait que ce modèle exploite les APIs MPI internes de chaque cluster sans modification majeure de son architecture ou pas du tout, rend sa mise en œuvre accessible. De plus, sa structure, qui intègre deux tâches de communication interclusters, peut être un avantage supplémentaire parce que cela permet d'être indépendant du moyen de communication entre les clusters, il est possible dès lors d'utiliser différents réseaux, notamment des réseaux sur puce tel que le bus AXI.

5.3 Modèle de communication MPI-2 RMA basé sur l'approche PACX-MPI

Dans cette section, nous proposons des modèles et des traitements informatiques qui doivent être appliqués à une application parallèle MPI-2 RMA décrite suivant le template présenté dans l'algorithme 2, pour la génération d'une architecture parallèle de co-conception matérielle logicielle communicant suivant le paradigme MPI-2 RMA. Pour bien expliciter notre approche, dans la prochaine sous sous-section nous présentons le fonctionnement du modèle PACX-MPI sur lequel nous nous inspirons pour construire notre modèle de communication.

5.3.1 Présentation du modèle PACX-MPI

PACX-MPI (PArallel Computer eXtension)[197] a été principalement conçu pour permettre l'exécution d'applications MPI sur plusieurs supercalculateurs appelés MPP (Massively Parallel Processor systems) sans avoir à introduire des changements dans le code source de ces applications et en exploitant pleinement le sous-système de communication de chaque machine. Pour atteindre cet objectif, PACX-MPI est conçu comme une bibliothèque résidant entre l'application utilisateur et l'implémentation MPI locale intra-machine.

Lorsque l'application appelle une fonction MPI, l'appel est intercepté par PACX-MPI et une décision est prise pour savoir si un contact avec une autre machine est nécessaire pendant l'exécution de l'appel. Dans le cas contraire, la bibliothèque envoie le message en utilisant l'appel MPI correspondant de la bibliothèque MPI locale sous-jacente. Ainsi, les implémentations MPI du fournisseur, très bien adaptées, sont utilisées pour toutes les communications intra-machine. Lorsque l'appel MPI concerne une autre machine, la communication est transmise via le réseau en utilisant des sockets TCP/IP, mais dans ce cas, les processus MPI n'échangent pas directement les messages. Au lieu de cela, deux nœuds spéciaux sont réservés, sur chaque système parallèle, un pour chaque direction de communication (entrante et sortante). Sur chacun de ces nœuds, un processus MPI démon est en cours d'exécution et s'occupe de la communication avec les nœuds locaux, de la compression et de la décompression des données pour la communication à distance et de la communication avec les pairs de démons des autres machines parallèles. L'utilisation de deux nœuds de communication supplémentaires s'est avérée faciliter la gestion du trafic. PACX fournit à l'utilisateur un COMM_WORLD MPI global, comme le montre la figure 5.1 ou chaque

Algorithme 2 : Exemple de pseudo-code MPI pour le codesign par synthèse de haut niveau

```
// Initialisation de MPI
#pragma mpi MPI_Init(...) // Primitive MPI utilisée pour initialiser l'environnement
   d'exécution MPI (à redéfinir dans ce contexte)
#pragma mpi MPI_Comm_size(...) // Détermine la taille du groupe associé à un
   communicateur (à redéfinir )
#pragma mpi MPI_Comm_rank(...)// Détermine le rang du processus appelant dans le
   communicateur (à redéfinir )
.....
// Déclaration des objets
type A[] ; type B[] ;
.....
// Description des tâches
if rank == r_0 then
    .....
end
else if rank == r_i then
    #pragma mpi map_plateform impl=software cluster=c1 name=t_{r_i}
    .....
    // description du contenu de la tâche
    .....
    #pragma mpi mpi_comm_primitive( param1, param2, ...)
    .....
end
.....
else if rank == r_j then
    #pragma mpi map_plateform impl=hardware cluster=c2 name=t_{r_j}
    .....
    // description du contenu de la tâche
    .....
    #pragma mpi mpi_comm_primitive( param1, param2, ...)
    .....
end
.....
else if rank == r_n then
    .....
end
#pragma mpi MPI_Finalize(...)
```

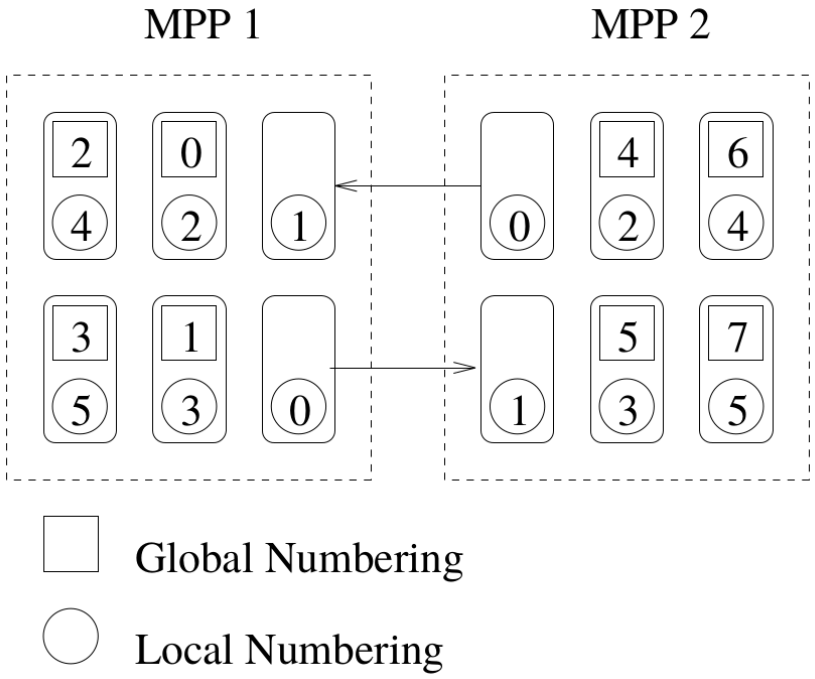

Figure 5.1: *Schéma de base de PACX sur deux MPP[197]*

tâche dispose d'un rang global qui correspond à un rang local en fonction de la machine dans laquelle celle-ci est déployée.

Les prochaines sous-sections présentent modèles et les traitements à appliquer sur notre proposition de template de description d'application, pour construire un modèle de communication PACX-MPI à implémenter sur MATIP en vue permettre la génération/synthèse d'architectures de co-conception matérielle logicielle.

5.3.2 La génération du communication world (COMM_WORLD)

A partir du modèle parallèle décrit par le développeur dans un langage de haut niveau (voir l'algorithme 2), nous extrayons la configuration de partitionnement matériel-logiciel définie par le langage pragma. Afin de constituer deux clusters : l'un constitué de tâches matérielles et l'autre de tâches logicielles. A ces tâches sont attribués des rangs locaux dans chaque cluster (chaque rang local étant supérieur à 1 car les rangs 0 et 1 sont réservés aux tâches de communication interclusters). Pour la génération de l'architecture hétérogène, nous construisons une table de correspondance des rangs pour chaque cluster. L'algorithme que nous proposons pour la construction de cette table de correspondance prend en entrée, pour un sous-cluster C_i l'ensemble R_G des rangs globaux, et l'ensemble $R_{G_{C_i}} \subset R_G$ des rangs globaux des tâches mappées sur le cluster C_i. Cet algorithme attribue des rangs locaux croissants r_l de R_{LC_i} aux différentes tâches mappées sur le cluster C_i puis calcule pour chaque rang global $r_g \in R_G$ les rangs $r_{l_{in}} \in R_{LC_i}$ et $r_{l_{out}} \in R_{LC_i}$ respectivement les rangs locaux de destination pour les communications entrantes et sortantes. Nous proposons dans l'algorithme 3 la procédure d'attribution des rangs locaux pour un cluster C_i. Dans l'algorithme 4, nous spécifions le processus de construction de la table de correspondance des communications. La table de correspondance ainsi obtenue est utilisée par le synthétiseur de haut niveau pour la génération des codes de communication internes de chaque cluster. Selon le modèle PACX-MPI, les communications dites distantes qui ont lieu entre des tâches situées de part et d'autre des deux clusters doivent passer par les tâches de communication interclusters (de rang 0 et 1). Les rangs $r_{l_{in}}$ et $r_{l_{out}}$ sont utilisés pour l'instanciation des primitives de communication entrante et sortante respectivement. Le tableau 5.1 présente la liste

de certaines primitives de communication et leurs catégories selon qu'elles manipulent des rangs pour les communications entrantes ou sortantes.

Algorithme 3 : Allocation de rangs locaux dans le cluster C_i

Entrées : $R_{G_{C_i}} \subset R_G$: Rangs globaux des tâches mappées dans C_i
Sorties : $A_{C_i} : \{(r_g, r_l)\} \subset R_G \times R_{LC_i}$, La table des allocations
soit $r_l \in R_{LC_i}$
$r_l \leftarrow 2$
pour chaque r_g ***dans*** $R_{G_{C_i}}$ **faire**
  $A_{C_i} \leftarrow A_{C_i} \cup \{(r_g, r_l)\}$
  $r_l \leftarrow r_l + 1$
retourner A_{C_i}

Algorithme 4 : Construction de la table de communication de C_i.

Entrées : R_G : L'ensemble des rangs globaux,
$R_{G_{C_i}} \subset R_G$: Rangs globaux des tâches mappées dans C_i,
$A_{C_i} \subset R_G \times R_{LC_i}$: La table d'allocation
Sorties : $T_{C_i} : \{(r_g, r_{l_{in}}, r_{l_{out}})\} \subset R_G \times R_{LC_i} \times R_{LC_i}$, La table de communication
pour chaque r_g ***dans*** R_G **faire**
  si $r_g \in R_{G_{C_i}}$ **alors**
    soit $a_g \in A_{C_i}, \setminus\ a_g = (r_{g_k}, r_{l_k})$, **et** $r_{g_k} = r_g$
    // Tâche dans C_i: $r_{l_{in}} \leftarrow r_{l_k}$ et $r_{l_{out}} \leftarrow r_{l_k}$
    $T_{C_i} \leftarrow T_{C_i} \cup \{(r_g, r_{l_k}, r_{l_k})\}$
  sinon
    // Tâche à l'extérieur de C_i: $r_{l_{in}} \leftarrow 1$ et $r_{l_{out}} \leftarrow 0$
    $T_{C_i} \leftarrow T_{C_i} \cup \{(r_g, 1, 0)\}$
retourner T_{C_i}

Tableau 5.1: *Liste de certaines primitives de communication qui manipulent les rangs pour les communications entrantes ou sortantes.*

Primitive	Entrante	Sortante
MPI_Send		x
MPI_Recv	x	
MPI_Group_incl	x	x
MPI_Put		x
MPI_Get	x	

Pour illustrer le fonctionnement de ces algorithmes, nous considérons une application MPI composée de 8 tâches réparties dans deux clusters C_1 et C_2 comme spécifié dans le tableau 5.2. Le tableau 5.3 montre le résultat de l'algorithme d'allocation locale de rang. La figure 5.2 représente la structure synoptique de l'interconnexion entre les deux grappes hétérogènes. Le tableau 5.4 présente la table de correspondance $ComTab_{C_i}$ obtenue en exécutant l'algorithme 4 sur les résultats obtenus dans la phase d'allocation des rangs.

5.3.3 Mécanismes de communication MPI-2 RMA basés sur l'approche PACX-MPI

L'approche PACX-MPI implémente la communication de tâches déployées dans deux clusters hétérogènes en prenant en compte deux cas : lorsque les tâches sont sur le même cluster et

Tableau 5.2: *Partitionnement d'une application mpi de 8 tâches dans deux clusters*

	Cluster C_1				Cluster C_2			
r_g	0	1	2	3	4	5	6	7

Tableau 5.3: *Le résultat de l'algorithme d'attribution de rangs locaux*

	Cluster C_1				Cluster C_2			
r_g	0	1	2	3	4	5	6	7
r_l	2	3	4	5	2	3	4	5

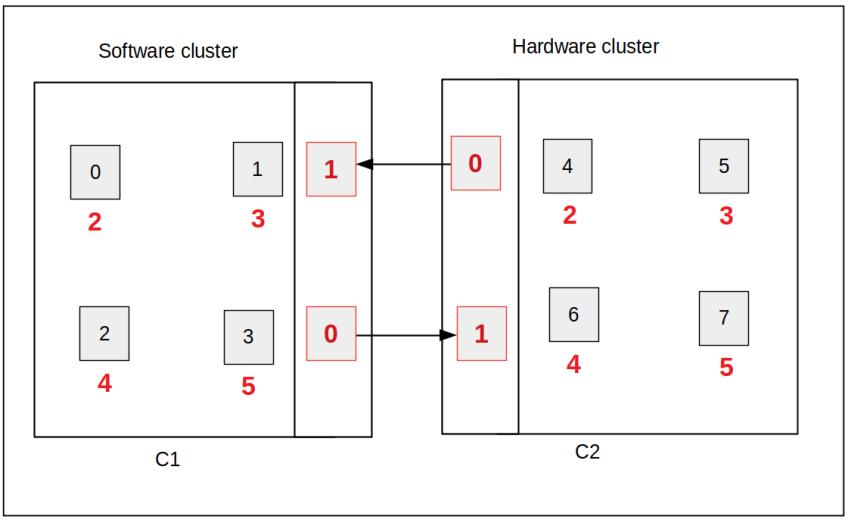

Figure 5.2: *Structure synoptique de l'interconnexion entre les deux clusters hétérogènes*

Tableau 5.4: *Construction des tables de communication de C_1 et C_2.*

	Cluster C_1							
r_g	0	1	2	3	4	5	6	7
$r_{l_{in}}$	2	3	4	5	1	1	1	1
$r_{l_{out}}$	2	3	4	5	0	0	0	0
	Cluster C_2							
r_g	0	1	2	3	4	5	6	7
$r_{l_{in}}$	1	1	1	1	2	3	4	5
$r_{l_{out}}$	0	0	0	0	2	3	4	5

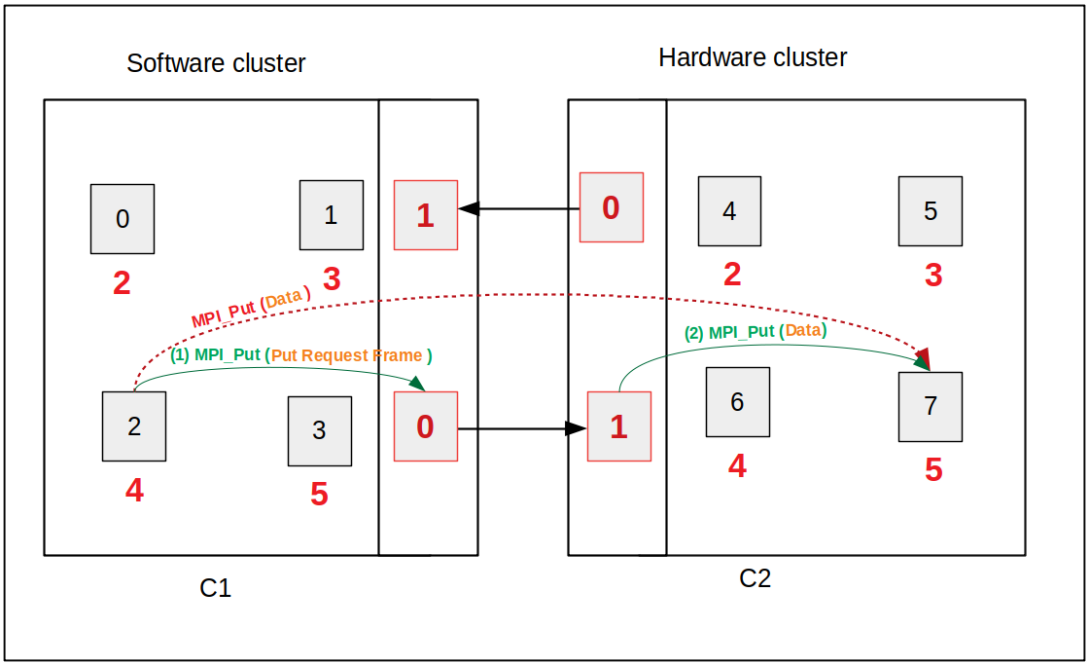

FIGURE 5.3: *Communication par MPI_Put entre deux tâches de clusters différents*

lorsqu'elles sont distribuées entre les deux clusters. Les travaux présentés dans [197] prennent en compte les mécanismes de communication en exploitant le standard MPI-1, en particulier *MPI_Send*, *MPI_Recv*, etc. Dans ce travail, nous proposons des mécanismes de communication utilisant la norme MPI-2 RMA. Nous proposons les fonctions *rankin* et *rankout* qui prennent en entrée le cluster C_i et le rang global r_g et retournent respectivement r_{lin} et r_{lout} les rangs locaux correspondants (voir le tableau 5.4). Dans la norme MPI-2 RMA, nous implémentons deux primitives principales de communication pair à pair, à savoir *MPI_Put* et *MPI_Get*.

Soit A et B deux tâches MPI de rangs globaux respectifs r_A et r_B déployées dans les grappes respectives C_A et C_B ; telles que la tâche A envoie des données à la tâche B via la primitive *MPI_Put* selon le scénario de communication de l'application. La réalisation de cette communication dépendra du fait que les tâches A et B sont dans le même cluster ou non :

- Si A et B sont dans le même cluster, c'est-à-dire que $rankout(ComTab_{C_A}, r_B) \neq 0$, alors nous instancions la primitive *MPI_Put* avec le paramètre de destinataire $rankout(ComTab_{C_A}, r_B)$, la communication est alors appelée *communication intra-cluster* ;
- Si A et B ne sont pas dans le même cluster, c'est-à-dire que $rankout(ComTab_{C_A}, r_B) = 0$, alors la tâche A construit une trame de communication contenant les paramètres de la primitive *MPI_Put*, les données et toutes les autres informations utiles (voir Tableau 5.5), puis les envoie à la tâche (de rang local 0) chargée de transmettre les informations de communication externe. La trame est transmise à la tâche de rang 1 de l'autre cluster qui assure la transmission des informations à la tâche finale B via la primitive *MPI_Put*. La figure 5.3 illustre la communication entre deux tâches de clusters différents. Nous proposons l'algorithme 5 qui décrit *pacx_MPI_Put*, la réécriture de la primitive *MPI_Put* en se basant sur une API MPI-2 RMA existante pour une utilisation dans une tâche locale PACX-MPI déployée dans l'un des clusters : cluster logiciel ou cluster MATIP. *pacx_MPI_Put* implémente le principe de fonctionnement décrit ci-avant pour la mise en œuvre de la primitive MPI_Put.

De même, nous considérons que la tâche A souhaite communiquer avec la tâche B en recevant des données via la primitive *MPI_Get* fournie dans la norme MPI-2 RMA. La réalisation de cette opération dépend également du fait que les tâches A et B se trouvent dans le même cluster ou

Tableau 5.5: *Structure de la trame de communication MPI_ Put(InstCode est le code instruction de la primitive et les autres éléments sont les paramètres de la primitive prévus dans la documentation MPI-2 RMA)*

InstCode	origin_count	origin_datatype	trget_rank	trget_disp	trget_count	trget_datatype	**data**

non. Ainsi :

— Si A et B font partie du même cluster, c'est-à-dire que $rankin(ComTab_{C_A}, r_B) \neq 1$, alors la communication est réalisée en instanciant la primitive *MPI_ Get* avec $rankin(ComTab_{C_A}, r_B)$ comme rang de destination ;

— Si A et B ne font pas partie du même cluster, c'est-à-dire que $rankin(ComTab_{C_A}, r_B) = 1$, alors la tâche A crée une trame de requête *MPI_ Get* (voir Tableau 5.6), la transmet à la tâche de rang 0 du cluster C_A qui la transmet à son tour à la tâche de rang 1 du cluster C_B. Cette dernière effectue l'opération Get avec la tâche B, puis produit une trame de réponse Get contenant les données récupérées (voir Tableau 5.7) et la transmet à la tâche de rang 0 du cluster C_B. La trame est ensuite transmise à la tâche de rang 1 du cluster C_A ; les données parviennent ainsi à la tâche A soit par une requête *MPI_ Get* initiée par la tâche A vers la tâche de rang 1, soit par une requête *MPI_ Put* initiée par la tâche de rang 1 vers la tâche A. La figure 5.4 illustre le processus de communication entre deux tâches de clusters différents via la primitive *MPI_ Put*. Nous proposons l'algorithme 6 qui décrit *pacx_ MPI_ Get*, la réécriture de la primitive *MPI_ Get* en se basant sur une API MPI-2 RMA existante pour une utilisation dans une tâche locale PACX-MPI. *pacx_ MPI_ Get* implémente également le principe de fonctionnement déroulé ci-dessus.

Tableau 5.6: *Structure de la trame de la requête MPI_ Get (InstCode est le code instruction de la primitive et les autres éléments sont les paramètres de la primitive prévus dans la documentation MPI-2 RMA)*

InstCode	origin_rank	origin_count	origin_datatype	trget_rank	trget_disp	trget_count	trget_datatype

Tableau 5.7: *Structure de la trame de réponse MPI_ Get (InstCode est le code instruction de la primitive et les autres éléments sont les paramètres de la primitive de la requête MPI_ Get de départ)*

InstCode	origin_rank	origin_count	origin_datatype	trget_disp	trget_count	trget_datatype	**data**

5.3.4 Principe de fonctionnement des tâches 0 et 1 de communication externe

Pour assurer la communication intercluster, les tâches 0 et 1 sont connectées respectivement aux interfaces de communications sortante et entrante. Ainsi, la tâche 0 connectée à l'interface de sortie du cluster, a la charge d'acheminer toutes les trames de communication extra-cluster ; la tâche 1 quant à elle lit et analyse les trames venant de l'interface des communications entrantes. À cet effet, en fonction de la nature [2] de la trame entrante (nature déduite à partir du code instruction situé en tête dans la trame), la tâche 1 effectue un ensemble d'actions permettant de réaliser la requête encodée dans ladite trame. L'algorithme 7 décrit en pseudo-code le fonctionnement de la tâche 1 ; et l'algorithme 8 décrit le fonctionnement de la tâche 0.

S'exécutant tout au long de la durée de vie de l'application, la tâche 1 décrite dans l'algorithme 7 écoute le canal des communications entrantes et lit les trames. Elle analyse et traite ces dernières en fonction de leur nature déterminée par la première donnée qui représente le code instruction. Ce code instruction peut prendre plusieurs valeurs notamment : *MPI_ Put_ Code* pour les trames d'exécution distante de la primitive *MPI_ Put*, *MPI_ Get_ Request_ Code* pour la requête d'exécution distante de la primitive *MPI_ Get*, *MPI_ Get_ Response_ Code* pour la

2. trame MPI_Put, MPI_Get, Réponse MPI_Get d'une tâche externe au cluster, etc.

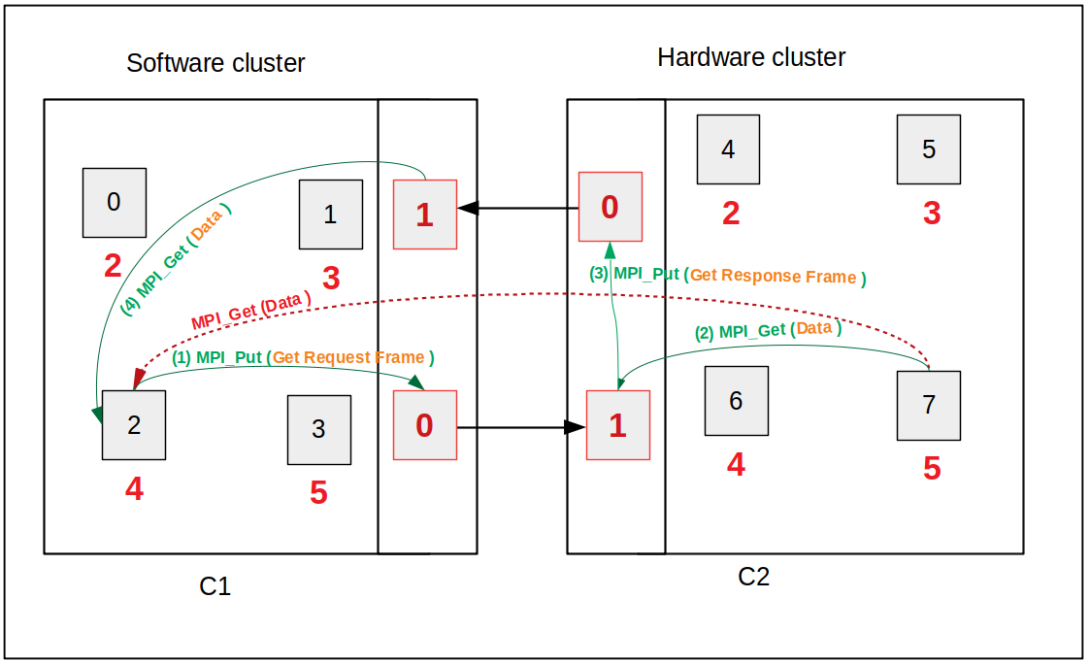

FIGURE 5.4: *Processus de communication entre deux tâches de clusters différents via la primitive MPI_Put.*

Algorithme 5 : pacx_MPI_Put (**const void** *origin_addr, **int** origin_count, **MPI_Datatype** origin_datatype, **int** target_rank, **MPI_Aint** target_disp, **int** target_count, **MPI_Datatype** target_datatype, **MPI_Win** win)

```
const InstCodePut ← p ; // p choisi de façon conventionnelle
r_lout : int ;
r_lout ← rankout(ComTab_Ci, target_rank) ;
si r_lout ≠ 0 alors
    // la tâche cible est dans le cluster local
    appel local_MPI_Put (origin_addr, origin_count, origin_datatype, r_lout, target_disp,
      target_count, target_datatype, win) ;
sinon
    // la tâche cible est hors du cluster local
    HeadPutRequest [ ] ← {InstCodePut, origin_count, origin_datatype, target_rank,
      target_disp, target_count, target_datatype } ; // construction l'en-tête de la trame
      de la requête Put
    new_addr←concat(addressof(HeadPutRequest), length(HeadPutRequest), origin_addr,
      origin_count) ; // cancatenation de la trame HeadPutRequest avec la trame de
      données
    appel local_MPI_Put (new_addr, origin_count+length(HeadPutRequest),
      any_datatype, r_lout, 0, origin_count+length(HeadPutRequest), any_datatype, win) ;
      // envoyer la trame com à la tâche de sortie r_lout =0 pour ce cas
```

Algorithme 6 : pacx_MPI_Get (**void** *origin_addr, **int** origin_count, **MPI_Datatype** origin_datatype, **int** target_rank, **int** target_disp, **int** target_count, **MPI_Datatype** target_datatype, **MPI_Win** win)

```
const InstCodeGet ← g ; // g choisi de manière coventionnelle
r_lin, r_l, origin_rank : int ;
r_lin ← rankin(ComTab_Ci, target_rank) ;
si r_lin ≠ 1 alors
    // la tâche cible est dans le cluster local
    appel local_MPI_Get (origin_addr, origin_count, origin_datatype, r_lin, target_disp,
      target_count, target_datatype, win) ;
sinon
    // la tâche cible est hors du cluster local
    local_MPI_Comm_rank(comm, &r_l) ; // obtention du rang local de la tâche en
      cours
    origin_rank ← getglobalrank(ComTab_Ci, r_l) ; // getglobalrank renvoie le rang global
      r_g qui correspond au rang local r_l dans la table de correspondance ComTab_Ci
    GetRequest[ ] ← {InstCodeGet, origin_rank, origin_count, origin_datatype, target_rank,
      target_disp, target_count, target_datatype } ; // construction de la trame GetRequest
    appel local_MPI_Put (addressof(GetRequest), length(GetRequest), MPI_INT, 0, 0,
      length(GetRequest), MPI_INT, win) ; // envoi de la trame GetRequest à la tâche du
      cluster de sortie 0
    appel local_MPI_Get (origin_addr, origin_count, origin_datatype, r_lin, target_disp,
      target_count, target_datatype, win) ; // Obtenir la réponse auprès de la tâche 1
```

réponse (envoie des données) liée à la requête *MPI_ Get* préalablement initiée par une trame de type (*MPI_ Get_ Request_ Code*). Ainsi, en fonction du type de trame, après lecture des différents paramètres, la tâche de rang 1 réalise l'ensemble des actions y afférentes. Pour le cas d'une requête de type MPI_Put par exemple, il est question d'identifier la tâche destinataire sur la base de son rang global *trget_ rank* transmis dans la trame (voir tableau 5.5) en l'appliquant à la fonction *rankout* qui exploite la table de correspondance de ce cluster(voir le tableau 5.4) et retourne le rang local de la tâche distante déployée dans ce cluster.

La tâche de rang 0, chargée des communications sortantes quant à elle (voir algorithme 8) permet aux autres tâches locales d'acheminer les requêtes des communications distantes (hors cluster). À tour de rôle, cette tâche reçoit les trames de communication des autres tâches locales, et les transmet sur le canal des communications sortantes.

5.4 XMATIP : une mise en œuvre du modèle MPI-2 RMA & PACX-MPI pour la co-conception et la co-synthèse HW/SW

Nous proposons une nouvelle plate-forme XMATIP (eXtended MATIP), une extension de la plate-forme MATIP qui permet la prise en compte de la co-conception matérielle logicielle d'applications parallèles sur FPGA. Cette extension correspond à l'implémentation dans MATIP, des deux tâches de communication permettant la communication du cluster induit par MATIP avec l'extérieur. Nous proposons dans XMATIP une API qui exploite l'API MATIP existante. Les tâches responsables de la gestion des communications sont équipées d'interfaces de communication standard pour l'envoi (la tâche de rang 0) et la réception (la tâche de rang 1) des communications.

Le processus de mise en œuvre de l'architecture XMATIP consiste en un ensemble de lots importants pour l'atteinte des objectifs prévus par le modèle conceptuel (Modèle de communication PACX-MPI conçu et adapté pour le standard MPI-2 RMA). Il s'agit notamment de : la mise en œuvre des interfaces/canaux de communication entrantes et sortantes avec l'extérieur

Algorithme 7 : Fonctionnement de la tâche des communications entrantes (tâche 1)

```
const MPI_Put_Code ← k1 ; // k1 choisi de manière coventionnelle
const MPI_Get_Request_Code ← k2 ; // k2 choisi de manière coventionnelle
const MPI_Get_Response_Code ← k3 ; // k3 choisi de manière coventionnelle
InstCode, origin_count,origin_rank, target_rank, target_rank,target_count, : int ;
origin_datatype, target_datatype : MPI_Datatype ;
target_disp : MPI_Aint ;
tant que vrai faire
    lire(InstCode) ;
    // Le code instruction corespond à MPI_Put
    si InstCode = MPI_Put_Code alors
        // Lecture des autres paramètres
        lire(origin_count) ; lire(origin_datatype) ; lire(target_rank) ;
        lire(target_disp) ; lire(target_count) ; lire(target_datatype) ;
        tabdata ← tableau[origin_count] ;
        pour i allant de 1 à origin_count faire
            lire(tabdata[i]) ; // Lecture des données
        rank ← rankout(target_rank) ;
        // MPI_Put: envoie des données à la tâche destinataire trouvée dans le cluster
        si rank ≠ 0 alors
            appel local_MPI_Put (tabdata, origin_count, origin_datatype, rank,
              target_disp, target_count, target_datatype, win) ;
    // Le code instruction corespond à MPI_Get
    si InstCode = MPI_Get_Request_Code alors
        // Lecture des autres paramètres
        lire(origin_rank) ; lire(origin_count) ; lire(origin_datatype) ;
        lire(target_rank) ; lire(target_disp) ; lire(target_count) ;
        lire(target_datatype) ;
        tabdata ← tableau[origin_count] ;
        rank ← rankout(target_rank) ;
        // MPI_Get: On récupère les données depuis la tâche destinataire trouvée dans le
           cluster. Puis on renvoie la réponse en passant par la tâche 0
        si rank ≠ 0 alors
            appel local_MPI_Get (tabdata, origin_count, origin_datatype, rank,
              target_disp, target_count, target_datatype, win) ;
            Get_Response[ ]={MPI_Get_Response_Code, origin_rank, origin_count,
              origin_datatype, target_disp, target_count, target_datatype,tabdata,}
            appel local_MPI_Put (Get_Response, sizeof(Get_Response),
              CUSTUM_DATA_TYPE, 0, 0, sizeof(Get_Response),
              CUSTUM_DATA_TYPE, win) ;
    // Le code instruction corespond à une réponse MPI_Get extra-cluster
    si InstCode = MPI_Get_Response_Code alors
        // Lecture des autres paramètres
        lire(origin_rank) ; lire(origin_count) ; lire(origin_datatype) ;
        lire(target_disp) ; lire(target_count) ; lire(target_datatype) ;
        tabdata ← tableau[origin_count] ;
        pour i allant de 1 à origin_count faire
            lire(tabdata[i]) ; // Lecture des données
        rank ← rankout(origin_rank) ;
        // MPI_Get_Response: Exposition de la fenêtre des données venues du cluster
           distant au bénéfice de la tâche à l'origine de la requête.
        si rank ≠ 0 alors
            appel MPI_Win_create(tabdata, sizeof(tabdata), sizeof(int),
              MPI_INFO_NULL, MPI_COMM_WORLD, &win) ;
            appel MPI_Group_incl(comm_group, 1, &rank, &group) ;
            appel MPI_Win_post(group, 0, win) ;
            appel MPI_Win_wait(win) ;
```

Algorithme 8 : Fonctionnement de la tâche des communications sortantes (tâche 0)

```
const NB_Process ← K ; // K Nombre de tâches dans le cluster
const Taille_Frame_Max ← M ; // M Taille maximale d'une frame de communication
i ← 1 ;
tant que vrai faire
    si prêt(i) alors
        tabframe ← tableau[Taille_Frame_Max] ;
        rank ← i ;
        appel MPI_Win_create(tabframe, sizeof(tabframe), sizeof(int),
          MPI_INFO_NULL, MPI_COMM_WORLD, &win) ;
        appel MPI_Group_incl(comm_group, 1, &rank, &group) ;
        appel MPI_Win_post(group, 0, win) ;
        appel MPI_Win_wait(win) ;
        ecrire(tabframe) ;
    si i = NB_Process alors
        i ← 1 ;
    sinon
        i ← i+1 ;
```

du cluster (interface de communication de type AXI-Stream), l'implémentation d'un mécanisme de synchronisation entre les tâches de traitement et les tâches responsables des communications (Synchronisation basée sur des registres drapeaux), et enfin l'implémentation proprement dite des tâches de communication 0 et 1.

5.4.1 Mise en œuvre des interfaces/canaux de communication avec l'extérieur

Pour la communication des tâches du cluster induit par MATIP avec l'extérieur, nous avons opté pour le bus *AXI-Stream* comme canal pour l'envoi et la réception de données à haut débit. L'interface pour les communications entrantes est donc une interface *AXI-Slave* et l'interface pour les communications sortantes est *AXI-Master*. Pour que les tâches matérielles 0 et 1 puissent y accéder, nous avons connecté ces interfaces à l'architecture de la TIC. Cela a consisté à ajouter à la TIC les ports (interfaces) d'entrée et de sortie prévues par le protocole AXI-Stream. Pour l'implémentation rapide du protocole AXI nous avons utilisé l'IP AXI4-Stream FIFO[201] présente dans la bibliothèque des IP de Xilinx. Ainsi, nous avons instancié deux fois cette IP en connectant à la TIC l'interface *Master* sur l'une des instances et l'interface *Slave* sur l'autre. L'extrait de code A.6 représente le code VHDL de connexion des canaux de communication aux tâches 0 et 1. La figure 5.5 présente la structure de l'architecture finale de XMATIP ainsi obtenue. Il s'agit d'un cluster de tâches matérielles induit par MATIP capable de communiquer avec l'extérieur par MPI-2 RMA grâce à notre proposition d'API inspiré de PACX-MPI et à l'aide d'une infrastructure de communication externe haut débit de type AXI-stream. Cette architecture peut être utilisée pour développer des accélérateurs matériels autant qu'elle peut servir pour la conception des MPSoC hétérogènes.

5.4.2 Mise en œuvre du mécanisme de synchronisation entre tâches

Pour permettre une gestion efficace des communications et étant donné le risque élevé de sollicitations concurrentes des autres tâches vis-à-vis des tâches de communication avec l'extérieur (rang 0 et 1), il a été nécessaire de mettre en œuvre des mécanismes de synchronisation. A cet effet, nous avons défini des protocoles de synchronisation pour les principales primitives d'envoi/réception de données à savoir MPI_Put et MPI_Get. Le protocole de synchronisation pour la primitive MPI_Put est représenté dans le diagramme de séquence de la figure 5.6. Il se

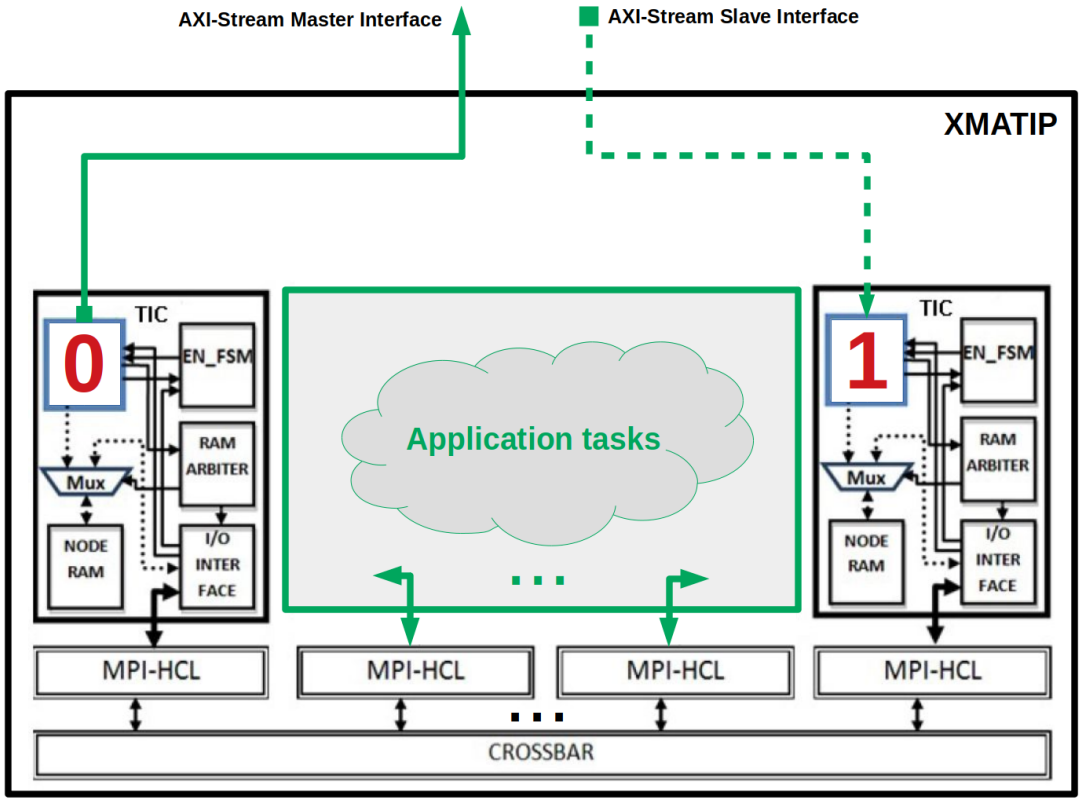

Figure 5.5: *Architecture XMATIP*

déroule comme suit : lorsqu'une tâche T_i veut envoyer des données à la tâche T_j via MPI_Put, la tâche T_i initie une demande d'envoi via un message *Need_ To_ Put()*. A la suite de ce message, la tâche T_j expose sa mémoire de communication, puis répond par un message *Ready_ Put()* pour signaler qu'elle est prête à recevoir les données. Par cette réponse, la tâche T_i réalise donc l'opération de communication MPI_Put puis envoie un message de confirmation *Done_ Put()*. De façon analogue, le protocole de synchronisation pour la primitive MPI_Get est géré par les messages *Need_ To_ Get()*, *Ready_ Get()* et *Done_ Get()* échangés entre les tâches T_i et T_j tel que décrit dans le diagramme de séquence de la figure 5.7.

Ainsi l'implémentation réelle de ces mécanismes de synchronisation s'est faite par la création des registres drapeaux pour les différentes catégories des messages. Pour un message/signal binaire de synchronisation donné (*Need_ To_ Get* par exemple) chaque tâche dispose d'autant de registres drapeaux qu'il y a de tâches dans le cluster induit par MATIP. Ainsi, pour le protocole de synchronisation MPI_Put, il faut 3 vecteurs de registres drapeaux (pour les différents messages *Need_ To_ Put, Ready_ Put, Done_ Put*) de K valeur chacun, K étant le nombre de tâches qu'il y a dans le cluster. Pour simplifier l'implémentation, nous avons fixé $K = 16$, valeur qui correspond au nombre maximal de tâches supportées par MATIP pour le moment.

L'envoi d'un message de T_i à T_j consiste pour T_i à assigner la valeur '1' à l'indice i du vecteur de registres drapeau correspondant au message. Après lecture du message, le registre drapeau est réinitialisé à la valeur '0' par T_j ; cela permet d'éviter une considération ultérieure du même message. Pour les opérations de modification et de lecture des différents registres drapeau, nous avons ajouté à la TIC (par conséquent à chaque tâche) des bus de données, d'adresse et de contrôle manipulables au sein de la tâche par le développeur. Pour le message *Need_ To_ Put* par exemple on a *need_ toput_ read_ data*(pour les valeurs du registre), *need_ toput_ read_ addr* (pour le numéro de la tâche concernée), *need_ toput_ read_ en* (comme signal de controle) pour la lecture, et *need_ toput_ write_ data*, *need_ toput_ write_ addr*, *need_ toput_ write_ en* pour l'écri-

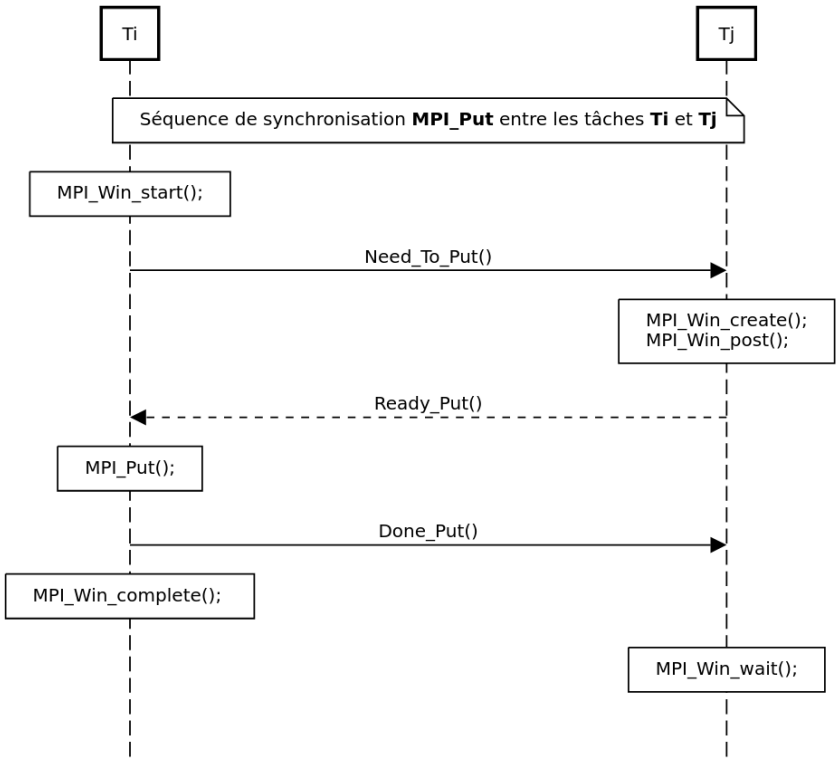

Figure 5.6: *Diagramme de séquence du protocole de synchronisation de la primitive MPI_ Put*

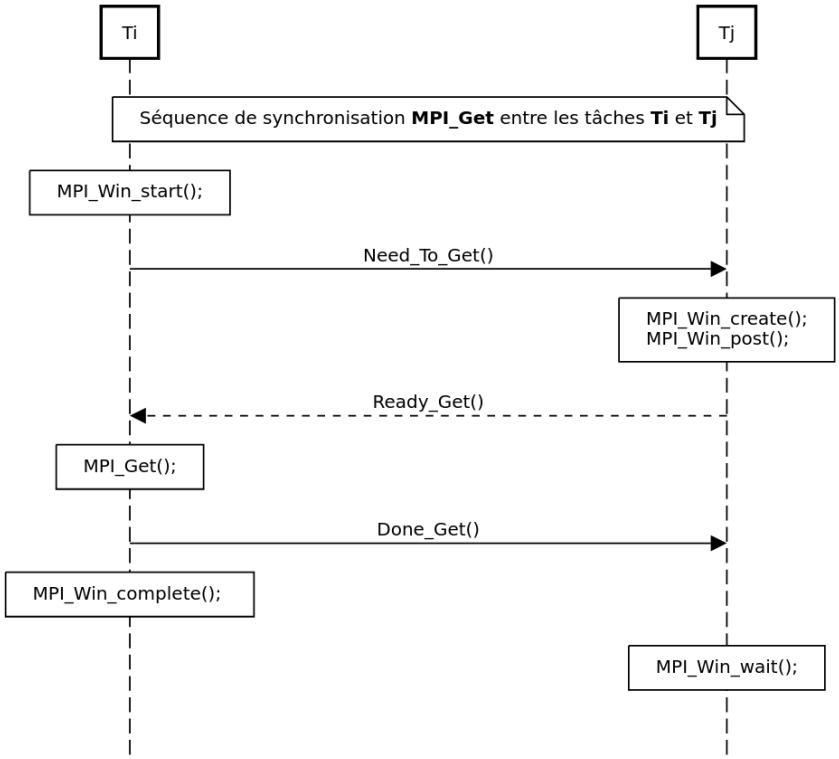

Figure 5.7: *Diagramme de séquence du protocole de synchronisation de la primitive MPI_ Get*

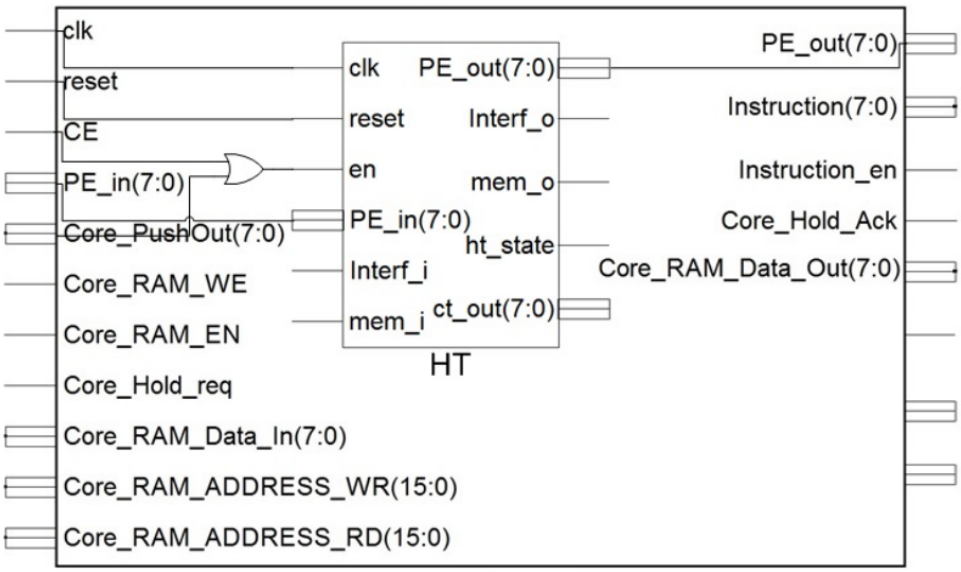

Figure 5.8: *Ancienne structure de l'interface TIC encapsulant la tâche matérielle (HT)[191]*

ture. La figure 5.8 présente l'ancienne structure de l'interface TIC encapsulant la tâche matérielle (HT)[191]. La figure 5.9 présente la nouvelle structure de l'interface TIC encapsulant la tâche matérielle trois groupes de signaux sont ajoutés représentant respectivement l'interface de communication avec l'extérieur (AXI-Stream), la synchronisation MPI_Put et MPI_Get.

5.4.3 Implémentation des tâches de communication

La tâche de gestion des communications entrantes assimilable à un processeur de communication avec une file d'attente a été implémentée via une machine à état. Sa fonction est de lire et de décoder le code d'instruction situé en tête de la trame de communication (voir Algorithme 7). Ainsi en fonction de l'instruction, elle lit les paramètres nécessaires et exécute ensuite l'ensemble des opérations associées. L'extrait de code A.7 présente un extrait de la tâche de gestion des communications entrantes implémentée en VHDL. Dans cet extrait, on peut voir que les branches de départ de la machine à états sont sélectionnées en fonction du code instruction contenue dans la variable *myinstruction*.

La tâche de gestion de la communication sortante est une tâche chargée de recevoir les requêtes/frames de communication des autres tâches et de les transmettre sur le canal de communication sortant (voir Algorithme 8). L'unicité de cette tâche pose un problème d'accès concurrent. En d'autres termes, cette tâche pose un problème de section critique car plusieurs tâches peuvent vouloir envoyer des données à l'extérieur du cluster au même moment. Dans l'état de l'art, plusieurs solutions sont proposées, notamment les solutions dites équitables telles que : le tie-breaker algorithm/algorithme de Peterson, le ticket algorithm, le bakery algorithm[202]. Ces différentes solutions sont efficaces pour la gestion d'une section critique en environnement logiciel, mais en contexte d'implémentation matérielle elles présentent des limites en termes consommation en ressources car la plupart nécessitent la mise sur pied des structures de données qui sont coûteuses en ressources mémoire. En outre le parcours de ces structures pour l'élection de la tache qui entre en section critique est coûteux en temps proportionnellement au nombre de tâches candidates.

Pour la simplicité d'implémentation, nous avons opté pour une solution de token ring. Le token étant géré par la variable i (voir algorithme 8). Ainsi la tâche 0 transmet, chacun à son tour, le jeton aux autres tâches du cluster ; si la tâche qui détient le jeton est prête à envoyer un message, son message est reçu et transmis sur le canal ; sinon, le jeton passe simplement à la tâche suivante et ainsi de suite. Pour signaler le besoin d'envoi de message, une tâche T_i envoie

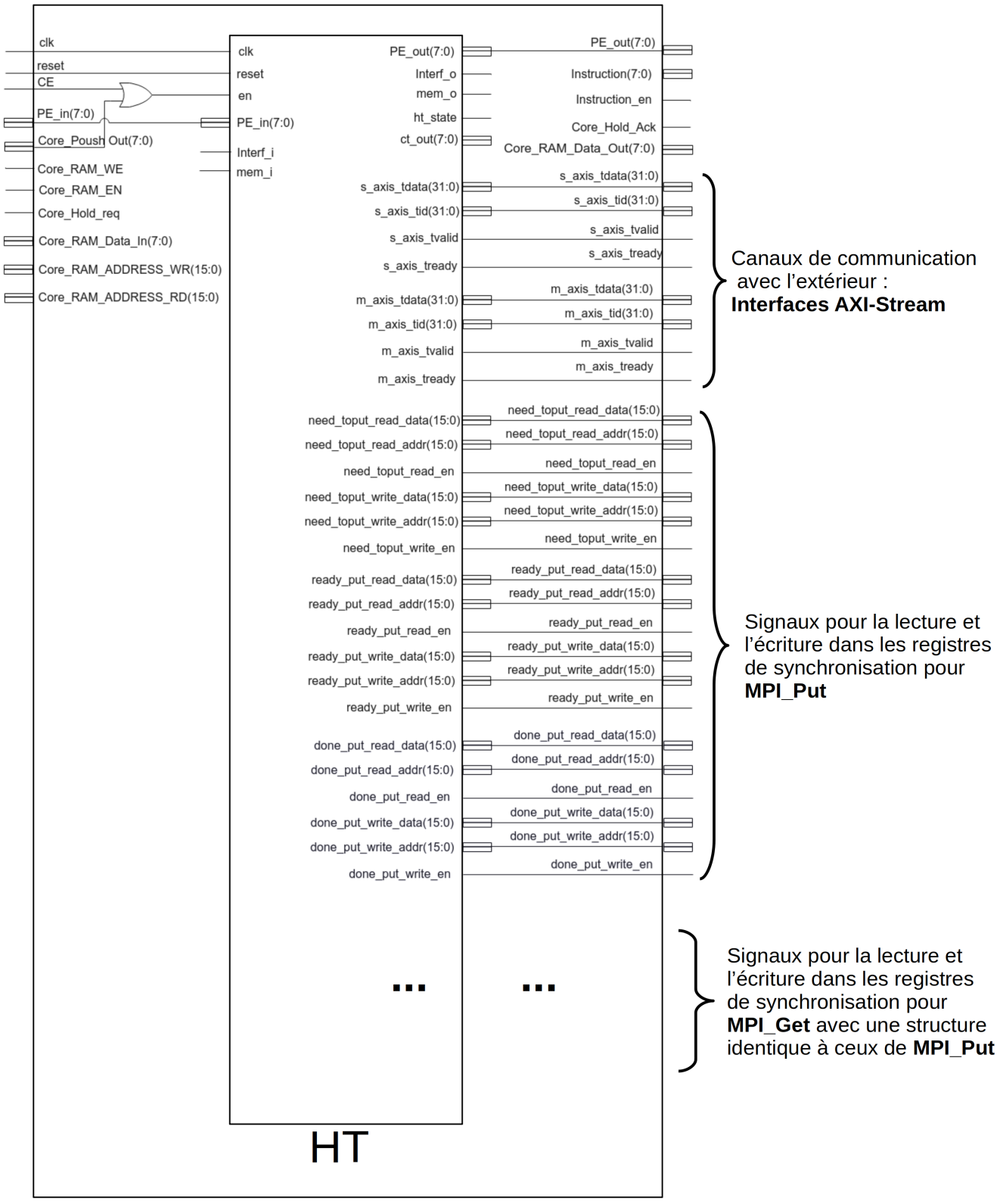

Figure 5.9: *Nouvelle structure de l'interface TIC encapsulant la tâche matérielle (HT) : trois groupes de signaux sont ajoutés pour la communication avec l'extérieur, la synchronisation MPI_ Put et MPI_ Get*

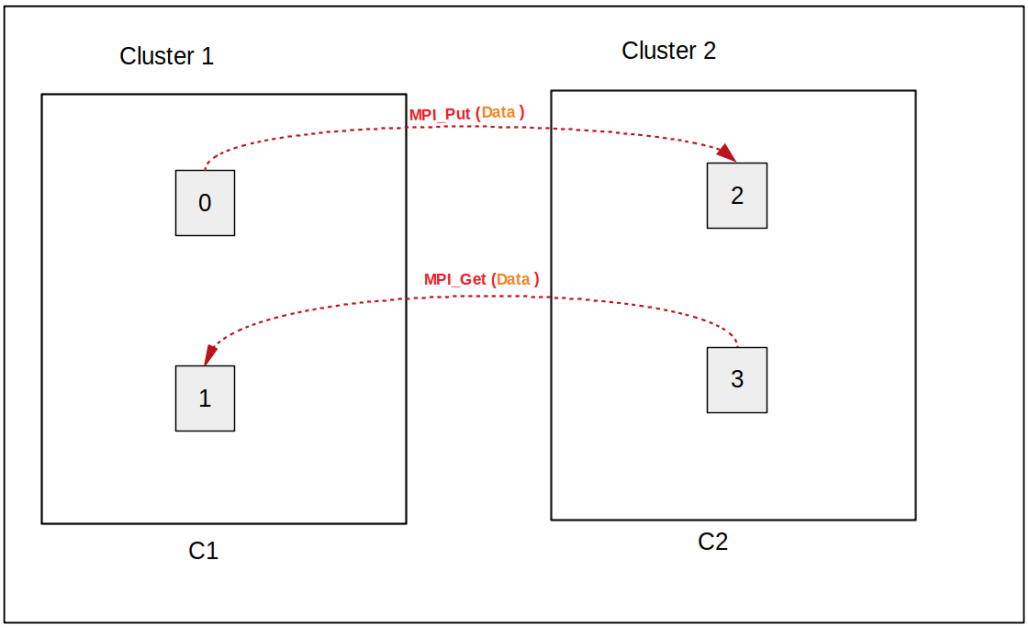

Figure 5.10: *Structure de l'application de validation de XMATIP*

le message *Need_ To_ Put()* à destination de la tâche 0, au moment où la tâche T_i acquière le jeton, la tâche 0 initie le processus de réception puis transmet sur le canal, la trame du message à acheminer à l'extérieur du cluster. L'extrait de code A.8 met en œuvre la tâche de gestion des communications sortantes implémentée en VHDL, la variable *listenposition* assure la gestion du jeton.

L'extrait de code A.9 représente le package VHDL qui spécifie les différentes machines à état qui gèrent les deux tâches. La machine à état spécifiée par *typ_ mae_ xmatip_ output* par exemple permet de gérer la tâche chargée des communications extérieures ; ainsi, lorsqu'elle se lance elle se met à *Start* ensuite passe à l'état *Listen* pour écouter les tâches qui sont prêtes à envoyer les données. Dès lors que l'une de ces tâches est prête, la machine passe à l'état *GetReady* qui consiste à recevoir la trame de communication auprès de la tâche qui détient le jeton et on passe à l'état *Proceed* l'acheminer via le canal de communication sortant ; ensuite la machine à l'état *Done*. Dans ce dernier état, les registres drapeau sont mis à jour et la machine revient à l'état *Listen*.

5.5 Test et validation de l'architecture XMATIP

Pour valider notre modèle, nous avons implémenté une application constituée de 4 tâches. Les 4 tâches numérotées allant du rang 0 au rang 3 sont réparties sur 2 clusters. Ainsi les tâches 0 et 1 sont dans le premier cluster que nous appelons C_1 et les tâches 2 et 3 dans le second cluster appelé C_2. Le modèle de communication est tel que : la tâche de rang 0 envoie des données à la tâche de rang 2 par la primitive *MPI_ Put* et la tâche de rang 1 récupère des données auprès de la tâche 3 par la primitive *MPI_ Get* comme l'illustre la figure 5.10. La figure 5.11 illustre la structuration de cette application suivant le modèle PACX-MPI.

L'implémentation du premier cluster a été faite avec XMATIP. Pour des contraintes de temps, nous avons simulé le second cluster en injectant les trames de requêtes et réponses PACX-MPI sur le canal des communications entrantes (Interface AXI-Slave de l'architecture XMATIP). Ce système peut également être utilisé si on veut simplement exploiter le cluster XMATIP comme accélérateur matériel où les tâches du dit cluster peuvent être considérées comme des tâches serveur de traitement de donnée ; l'architecture externe étant un client des serveurs en question.

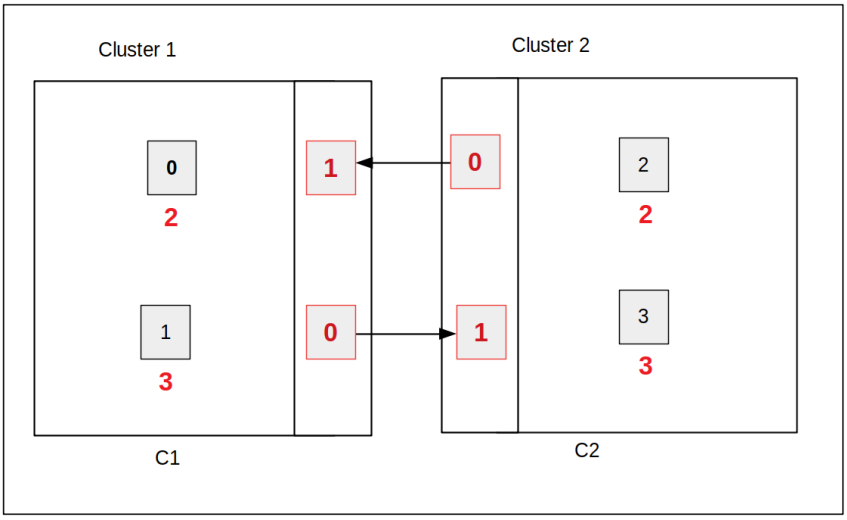

Figure 5.11: *Structuration de l'application de validation suivant le modèle PACX-MPI*

La figure 5.12 est un extrait de la simulation de notre application sur XMATIP. Il s'agit d'une séquence d'opérations que réalise la tâche 1 chargée des communications entrantes. En effet cette dernière a reçu une trame de réponse à une requête *MPI_ Get* formulée par la tâche de rang global 1 à destination d'une tâche externe au cluster. La tâche 1 procède donc à la lecture des paramètres avant de lancer l'exposition des données reçues à la tâche locale destinataire qui est éventuellement en attente de la réponse.

5.6 Discussion

XMATIP, extension de l'architecture MATIP par l'intégration du modèle de communication PACX-MPI offre la possibilité que des tâches matérielles constituées en cluster puissent communiquer avec d'autres systèmes. L'ajout de cette fonctionnalité à l'architecture d'overlay MATIP apporte de nombreux atouts. Le premier atout est lié au fait que, XMATIP permet désormais la possibilité de mise en œuvre d'une architecture parallèle intégrant la co-conception matérielle logicielle. Grâce à l'ajout de cette plateforme de communication avec l'extérieur, il est désormais possible de concevoir avec l'overlay MATIP un système parallèle qui intègre des tâches matérielles et des tâches logicielles pouvant être réparties sur deux clusters : un cluster de tâches matérielles induit par MATIP et un cluster de tâches logicielles qui s'exécutent sur un cluster de processeurs. La figure 5.13 décrit en quoi pourrait consister notre approche d'overlay intégrant la co-conception matérielle logicielle.

Dans cette mise à jour de notre approche, après conception du modèle parallèle, le concepteur réalise un partitionnement matériel logiciel où certaines tâches sont implémentées sur processeur et d'autres déployées sur l'architecture XMATIP. Après implémentation des deux sous-systèmes, on réalise leur intégration formant ainsi un cluster de deux clusters (Co2C).

La faisabilité de cette approche nécessite de mettre en œuvre dans le sous-système logiciel, un modèle de communication PACX MPI logiciel qui collabore avec celui mis en œuvre dans XMATIP. Un modèle de communication qui tournera sur un réseau de processeurs sur puce (réutilisé ou conçu par le développeur) qu'on interconnecte avec XMATIP via des bus AXI-Stream tel que prévu dans son architecture. De façon synthétique, la lise en œuvre d'une application via cette approche passe par :

— La conception d'un modèle parallèle MPI-2 RMA tel que vu dans le chapitre 4 ;
— Le partitionnement matériel logiciel pour décider de la réparation des tâches dans les deux sous-systèmes ;

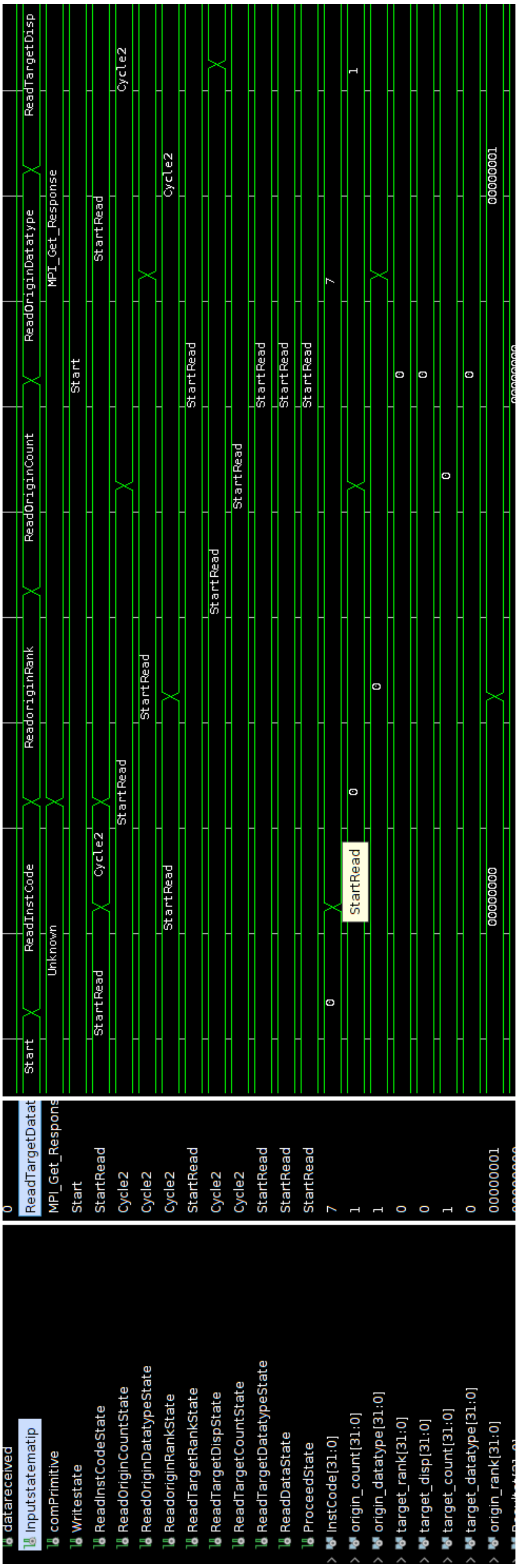

Figure 5.12: *Extrait de la visualisation de la simulation de l'application sur XMATIP*

— L'implémentation MPI-2 RMA des tâches logicielles sur la base des technologies prises en compte par les processeurs de ce cluster ;
— Une implémentation matérielle des autres tâches via une approche HLS + XMATIP qui sera juste l'exploitation de la méthodologie proposée dans le chapitre 4 sur l'architecture XMATIP ;
— L'intégration des deux sous-systèmes par interconnexion de XMATIP avec le cluster de CPU ; étape qui aboutit au déploiement après test et validation du système obtenu.

Pour pousser plus loins les possibilités, nous aimerions aller jusqu'à la construction d'une approche de co-synthèse matérielle logicielle. La figure 5.14 illustre le fonctionnement d'une telle approche. Dans ce flot le concepteur décrit l'application en langage de haut niveau à travers des spécifications du système sur différents aspects : description des tâches en langage de haut niveau, spécification du scénario MPI, spécification des interfaces matérielles des modules de traitement, etc. Ensuite, un outil de type synthèse de haut niveau vient synthétiser une architecture matérielle et logicielle.

L'idée de cette approche est de construire une méthodologie structurée en deux phases : une phase de spécification en langage de haut niveau et une phase de synthèse de haut niveau.

Dans la phase de spécification, il est question de décrire l'application parallèle sous différents aspects à travers 04 composantes :

— Une composante de description des tâches de traitement dans un langage de haut niveau (exemple le C/C++). Cette composante permettra de décrire les traitements à exécuter dans les différents nœuds soit sous forme de processus (après compilation pour les tâches logicielles), soit sous forme d'accélérateur matériel (après synthèse pour les tâches matérielles) ;
— Une composante de spécification du scénario de communication MPI-2 RMA qui nécessite la création d'un langage de pragma avec des algorithmes d'analyse de celui-ci pour la génération de la plateforme qui l'implémente ;
— Une composante de spécification des interfaces des modules matériels (tâches de traitement matérielles) en fonction des besoins et des contraintes de l'application ; car l'un des enjeux de cette approche est l'automatisation de la manipulation des modules matériels à travers leurs interfaces d'entrée et de sortie des données, de manière à faire une intégration automatisée et cohérente avec le scénario de communication ;
— Une composante de spécification de l'allocation des tâches entre les clusters matériel et logiciel ; cette composante peut aussi être implémentée sous forme d'un langage de pragma.

Dans la phase de synthèse/co-synthèse, il est question de générer de façon automatique le système déduit de la description de l'application faite par le développeur. Cette phase aussi fait intervenir plusieurs composantes :

— Une composante de traduction et de compilation qui prend en compte la description des tâches mappées sur le cluster logiciel, les compile et les déploie sur les différents processeurs ;
— Une composante de synthèse des tâches matérielles et de déploiement automatique sur XMATIP. Cette composante est assimilable à une automatisation complète de l'approche que nous avons proposée dans le chapitre 4, sauf qu'elle doit prendre en compte les spécificités propres à XMATIP ;
— Une autre composante est la configuration automatisée de l'interconnexion entre les deux clusters ;
— L'appel de toutes ces composantes est structuré et géré dans une composante centrale de génération automatique de plateforme.

À noter que, à la fin de chaque phase il est prévu une étape de validation par simulation ou co-simulation.

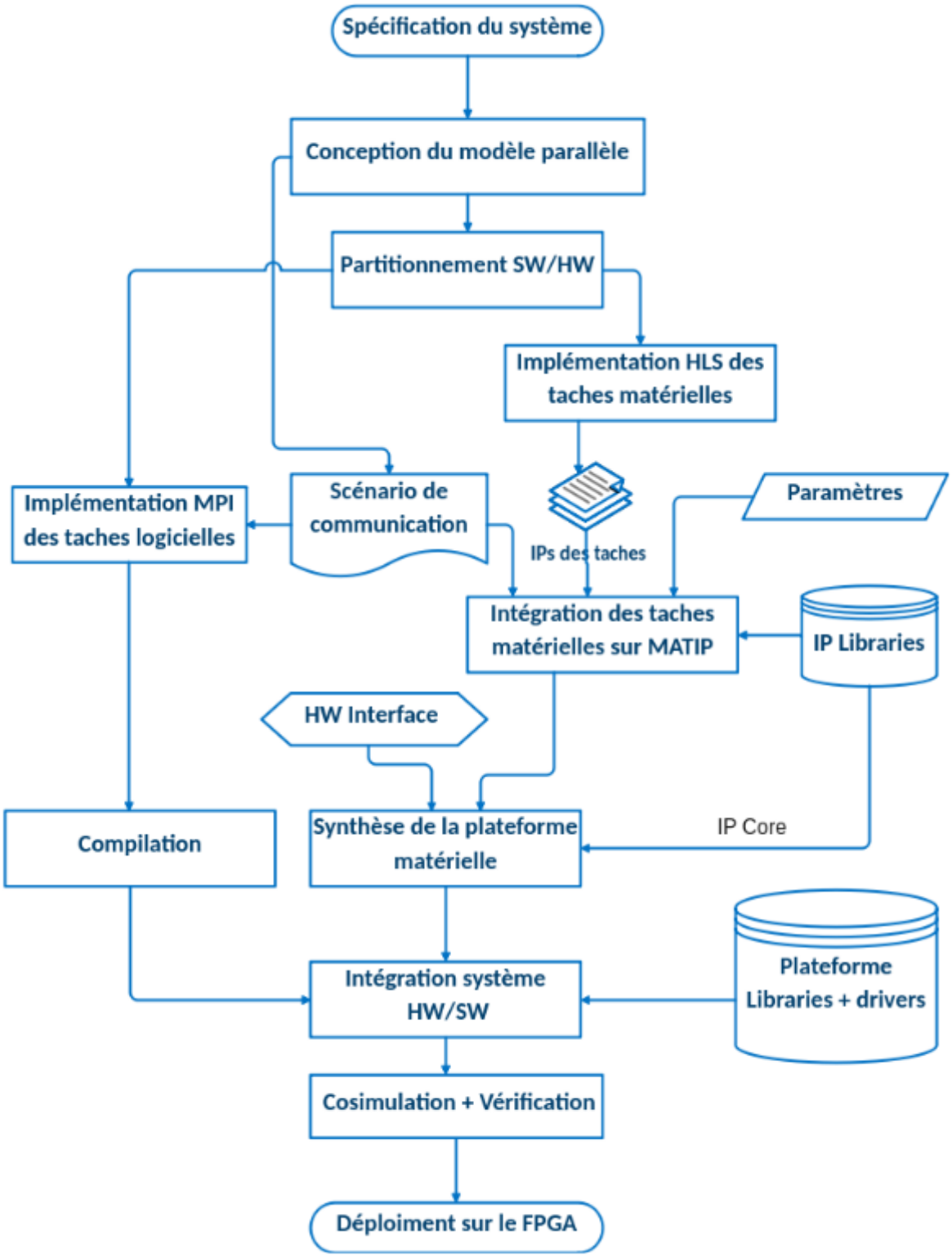

Figure 5.13: *Approche de codesign avec l'overlay MATIP*

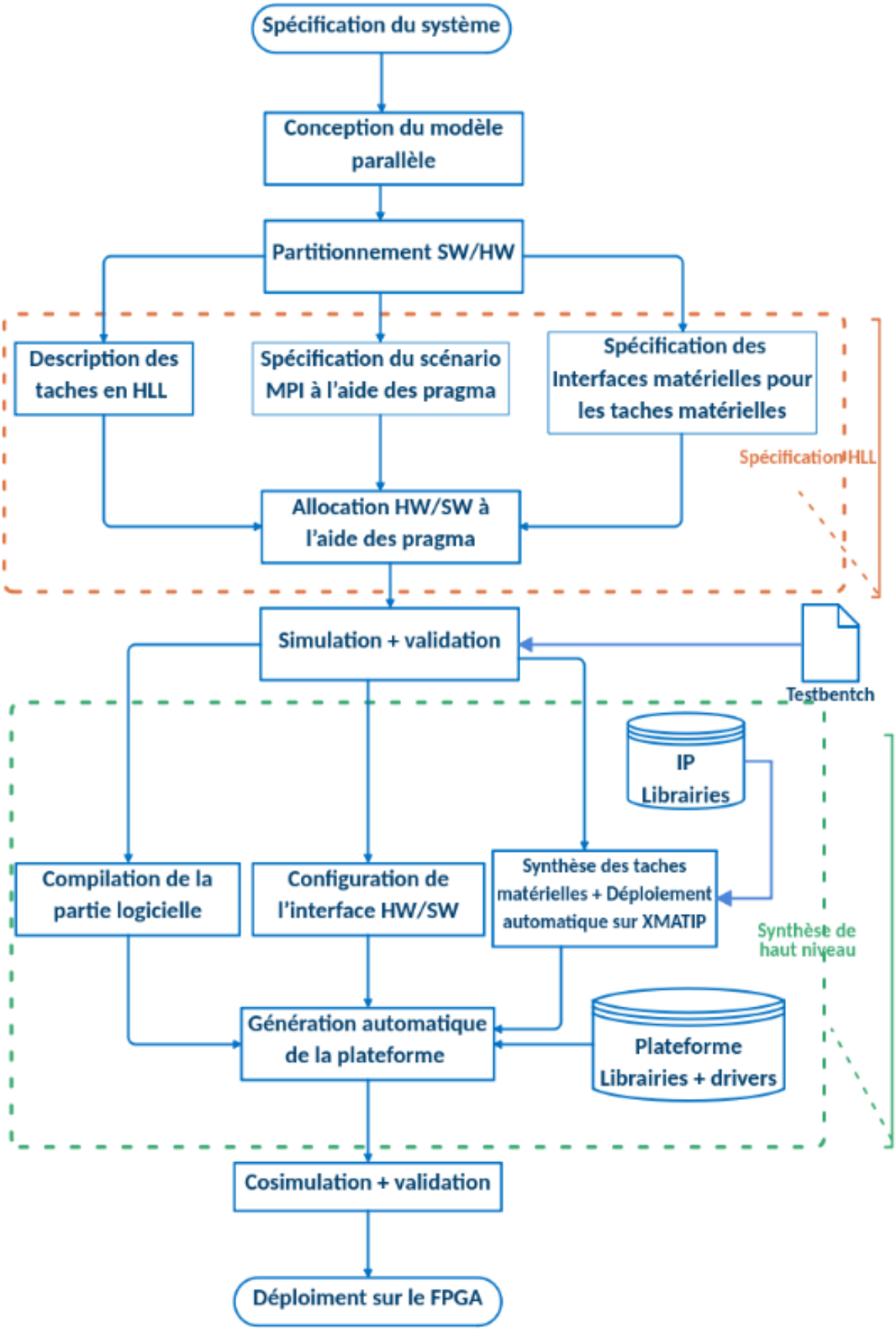

Figure 5.14: *Approche de co-synthèse matérielle logicielle avec l'overlay MATIP*

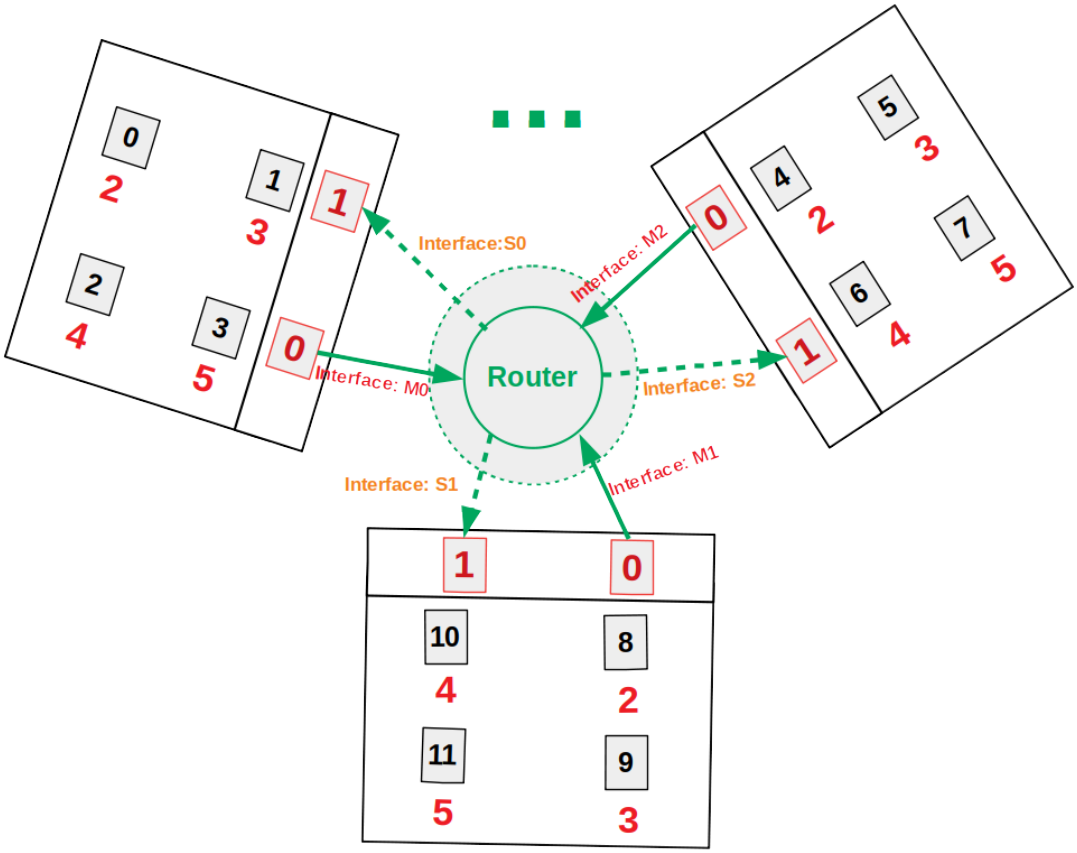

Figure 5.15: *Inter-connexion de plusieurs clusters XMATIP*

Le second atout de cette extension c'est que XMATIP permet d'accroître le nombre de tâches de MATIP et d'assurer ainsi le passage à l'échelle. En fonction des caractéristiques du réseau sur puce utilisé au niveau de la couche d'interconnexion, MATIP peut être limité en nombre de tâches matérielles à déployer. Par exemple, dans la version actuelle, MATIP ne peut intégrer que 16 tâches au maximum. Cela constitue donc un point bloquant, si on veut construire une architecture qui va au-delà de la limite prévue par le NoC. XMATIP permet d'instancier plusieurs clusters et de les interconnecter de façon que les tâches puissent communiquer de façon transparente. La figure 5.15 représente ce à quoi peut ressembler une telle architecture. Les interfaces des communications entrantes et sortantes de XMATIP étant de type AXI-Stream, on peut interconnecter plusieurs clusters basés sur XMATIP et contourner ainsi la limite en nombre de tâches. L'outil d'interconnexion peut être un réseau sur puce ou un routeur de paquets compatible aux bus AXI-Stream. AXI Interconnect est un exemple de composant qui peut permettre ce type d'interconnexion. On obtient une architecture capable de réaliser un calcul parallèle plus ou moins massif. Une telle architecture peut même être déployée sur environnement cloud permettant ainsi aux ingénieurs de réaliser des calculs massivement parallèles sur FPGA.

5.7 Résumé du chapitre

La parallélisation constitue un moyen viable pour la réduction du temps de traitement des applications. Notre approche permet de la mettre en œuvre de façon productive sur MP-RSoC. Toutefois, une application étant un ensemble de modules qui collaborent, il est nécessaire que MATIP, socle de notre approche offre une infrastructure efficace pour la communication avec des systèmes et ou des composants externes. Dans ce chapitre, nous avons présenté le contenu de la conception et la mise en œuvre de cette infrastructure en construisant XMATIP, une

extension de MATIP qui intègre cette exigence. Cette version étendue a été réalisée à travers la conception et la mise en œuvre d'un modèle de communication MPI entre clusters hétérogènes, inspiré du modèle PACX-MPI. Nous avons adapté le modèle PACX-MPI conçu pour MPI-1 au paradigme MPI-2 RMA. Puis nous avons mis en œuvre et testé ce modèle dans l'architecture MATIP. L'ajout de cette infrastructure de communication crée de nouvelles possibilités pour notre approche. De façon immédiate, cette infrastructure permet de connecter de façon efficace l'accélérateur matériel du module d'extraction des caractéristiques que nous avons mis en œuvre au reste des modules du système de reconnaissance vocale. Vu sur un spectre plus large, Il rend notre approche compatible à des approches de codesign et co-synthèse matérielle logicielle. De plus, grâce à cette infrastructure, il est possible de construire un cluster de clusters induits par MATIP afin de réaliser du calcul massivement parallèle sur FPGA.

Chapitre

6 Conclusion générale

Sommaire

6.1 Bilan des travaux

La langue propre à la communauté d'origine d'un individu occupe une place très importante pour son déploiement dans les différentes activités socio-économiques. Le besoin de prise en compte des langues locales dans les sociétés africaines dont l'utilisation a été réduite au profit des langues officielles est de plus en plus prononcé. Leur intégration dans les services socio-économiques qui sont aujourd'hui fortement digitalisés pourrait apporter plus de pertinence et de qualité en termes d'utilisabilité. Pour le cas du Cameroun, l'intégration des langues locales à ces applications/services présente des intérêts sur diverses dimensions. Ces intérêts sont notamment d'ordres : sociolinguistiques, éducatifs, économiques, juridiques et même scientifiques. L'interaction homme machine via la reconnaissance vocale est le mode le plus pratique dans cet écosystème rempli d'objets et de services intelligents. Ces objets intelligents sont souvent le produit de la mise en œuvre des traitements sur système embarqué. Des systèmes qui sont généralement sous une contrainte de temps réel. Les langues camerounaises sont pour la majorité des langues à tons et peu dotées. Ces caractéristiques rendent complexe la mise en œuvre des systèmes de reconnaissance vocale.

Dans cette thèse, nous avons adressé la mise en œuvre d'un système de reconnaissance des langues camerounaises sur système embarqué. Pour la satisfaction de la contrainte de temps réel, nous avons pu identifier deux principales approches de réduction du temps de traitement que sont : la parallélisation et la mise en œuvre sur architecture matérielle efficace en temps de traitement (FPGA, DSP, GPU, etc.). Dans ce travail, nous avons opté pour une approche hybride. L'idée de notre approche est de mettre en œuvre de la parallélisation sur une architecture reconfigurable (FPGA).

Compte tenu de la complexité de l'architecture d'un système de reconnaissance vocale, nous avons fait une restriction de l'application en nous focalisant pour un début uniquement sur la composante d'extraction des caractéristiques du signal vocal. La fonction d'auto-corrélation de Praat est l'algorithme que nous avons retenu, au vu de sa précision comparée à celle des autres algorithmes et de son taux d'utilisation dans la communauté des scientifiques et développeurs.

Après implémentation d'une version sur environnement logiciel, nous avons adressé la mise en œuvre du modèle parallèle de cet algorithme sur système embarqué, et plus précisément sur

architecture reconfigurable. En analysant les contours de cette architecture, nous nous sommes rendu compte que la mise en œuvre d'une application parallèle sur architecture reconfigurable invoque plusieurs problématiques qui peuvent s'avérer très lourdes pour le développeur. Il s'agit notamment : de la mise en œuvre de l'infrastructure d'interconnexion et de communication entre tâches, de la synchronisation et de la mise en œuvre des tâches de traitement parallèles proprement dites. Par ailleurs la mise en œuvre d'applications sur architecture reconfigurable tel que les FPGAs se fait généralement à l'aide des langages de type HDL connus pour être de très bas niveaux. Tout ceci pose donc un problème de productivité de développement, où l'ingénieur a à sa charge toutes les problématiques citées ci-avant.

A la recherche d'une solution, nous avons parcouru l'état de l'art des approches de mise en œuvre productive des applications parallèles sur architecture reconfigurable. Parcours au cours duquel nous avons identifié plusieurs approches, notamment : les approches basées sur l'usage des outils de conception assistée par ordinateur (CAO), les approches basées sur la réutilisation des IPs, les approches basées sur l'utilisation des générateurs de circuits, les approches basées sur la synthèse de haut niveau et les approches basées sur les FPGA Overlay.

Malgré cet état de l'art riche en approches de solutions, nous n'avons pas pu identifier une approche qui puisse nous permettre de mettre en œuvre un modèle parallèle MPI-2 RMA de façon productive, en exploitant un langage de haut niveau proche de celui que nous avons utilisé pour la mise en œuvre de la version sur environnement logiciel. Nous avons jugé qu'il est intéressant pour un ingénieur, de disposer d'une solution qui lui permette de se focaliser uniquement sur l'application sans se soucier des problématiques liées à la mise en œuvre des systèmes parallèles. Après réflexion, nous avons eu une idée de solution, qui se base sur une plateforme existante, l'idée étant d'utiliser un outil de conception orienté plateforme appelé MATIP comme FPGA Overlay.

Cet outil abstrait/encapsule les problématiques liées à la mise en œuvre des systèmes parallèles. En effet, il permet le déploiement de tâches matérielles sur architecture reconfigurable avec une infrastructure de communication qui implémente le paradigme MPI-2 RMA. Pour plus de productivité, nous avons combiné cet outil avec l'approche de synthèse de haut niveau pour en faire un FPGA Overlay basé sur la synthèse de haut niveau. L'approche HLS permet à l'ingénieur de mettre en œuvre les taches parallèles dans un langage de haut niveau comme le C/C++. Nous avons développé cette idée d'approche en la déclinant en un ensemble d'étapes, méthodes et outils à exploiter pour la mise en œuvre productive d'un modèle parallèle d'une application sur architecture reconfigurable. Nous nous sommes par la suite servis de cette approche pour mettre en œuvre le modèle parallèle de l'algorithme d'extraction des caractéristiques sur FPGA. Après évaluation de cette mise en œuvre, en comparaison avec la version faite sur environnement logiciel et une mise en œuvre HLS de l'algorithme séquentiel sous forme d'IP, la version issue de notre approche s'est avérée être plus efficace en temps de traitement (évalué en nombre de cycle). Soit 7 fois plus rapide que le modèle parallèle sur environnement logiciel et près de 3 fois plus efficace que l'IP matérielle.

Afin de rendre notre approche compatible à la co-conception et co-synthèse matérielle logicielle, et par ailleurs permettre une communication des taches avec d'autres systèmes, nous avons conçu et mis en œuvre XMATIP, la version étendue de l'Overlay MATIP qui implémente la communication MPI-2 RMA entre clusters hétérogènes. Le modèle de communication implémenté a été inspiré du modèle PACX-MPI conçu pour MPI-1 et nous l'avons adapté au paradigme MPI-2 RMA. La mise en œuvre de XMATIP ouvre de nombreuses perspectives à notre approche. Elle rend cette dernière compatible à la co-conception et la co-synthèse matérielle logicielle. En outre, elle permet la mise en œuvre d'applications parallèles sur un cluster de clusters sur puce (On Chip Cluster of Clusters). Chaque cluster ici est constitué d'un ensemble de tâches matérielles déployées sur une instance de XMATIP.

6.2 Perspectives

Le travail proposé dans cette thèse ouvre plusieurs horizons dans le domaine de la mise en œuvre d'applications parallèles sur architectures reconfigurables. Nous envisageons plusieurs perspectives qui portent principalement sur :

- L'élaboration d'une version notre approche totalement automatisée
- L'élaboration d'une approche de co-synthèse matérielle logicielle d'application parallèles inspirée de l'approche proposée dans cette thèse
- L'évaluation des performances de XMATIP pour la mise en œuvre d'un cluster de clusters.

6.2.1 Automatisation totale de notre approche

Nous envisageons de rendre notre approche totalement automatique. En effet, cette approche est semi-automatique car l'étape d'intégration des taches matérielles et d'implémentation du scénario de communication est encore manuelle. Son automatisation peut être rendue possible par la mise en œuvre d'un générateur / synthétiseur du scénario de communication VHDL à partir de sa spécification faite en langage de haut niveau comme le C/C++. Par ailleurs, la synthèse de haut niveau des tâches matérielles et l'intégration de ces tâches à MATIP peuvent être automatisées à l'aide des scripts TCL. Disposer d'une solution totalement automatisée pourra donc éloigner le concepteur de toute la complexité de MATIP.

6.2.2 L'élaboration d'une approche de co-synthèse matérielle logicielle d'application parallèles

Sur la base d'une version automatisée de notre approche, on peut par la suite envisager l'élaboration d'une approche de co-synthèse matérielle logicielle où le développeur spécifie par lui-même le partitionnement et le mappage des tâches et l'outil de co-synthèse se charge de la compilation, la synthèse et le déploiement des tâches parallèles qui communiquent de façon transparente grâce au modèle de communication de clusters hétérogènes implémenté dans XMATIP. Cela donnerait donc plus de flexibilité au développeur.

6.2.3 Mise en œuvre et évaluation d'un cluster de clusters avec XMATIP

L'autre perspective est de tester et évaluer la mise en œuvre d'une application parallèle sur un cluster de clusters induits par XMATIP. Si une telle architecture s'avère être efficiente, XMATIP peut constituer une solution très intéressante pour le calcul haute performance sur FPGA, la disponibilité des ressources FPGA étant la seule limite majeure.

Bibliographie

Références du Chapitre 2: Intégration d'une application de reconnaissance des langues à tons dans un système embarqué

[1] Victor Cherbuliez. *L'aventure de Ladislas Bolski*. Hachette, 1883 *cf. p. 6.*

[2] Crystal David. *Language death*. 2000 *cf. p. 6.*

[3] Claude Hagège. *Halte à la mort des langues*. Odile Jacob, 2000 *cf. p. 6.*

[4] Zachée Denis Bitjaa Kody. "La dynamique des langues camerounaises en contact avec le français : Approche macrosociolinguistique". Thèse de doct. Univ. Yaounde, 2004 *cf. p. 6-8, 20.*

[5] C Hagege. "Words And Worlds : World Languages Review". In : *Multilingual Matters* (2005) *cf. p. 6.*

[6] Shing-Tai Pan, Chih-Chin Lai et Bo-Yu Tsai. "The implementation of speech recognition systems on FPGA-based embedded systems with SOC architecture". In : *International Journal of Innovative Computing, Information and Control* 7.11 (2011), p. 6161-6175 *cf. p. 6.*

[7] Yongqiang He et Xiguang Dong. "Real time speech recognition algorithm on embedded system based on continuous Markov model". In : *Microprocessors and Microsystems* 75 (2020), p. 103058 *cf. p. 6.*

[8] Joseph H Greenberg et al. "African linguistic classification". In : *General history of Africa* 1 (1981), p. 113-121 *cf. p. 6.*

[9] Steven Bird. "Orthography and identity in Cameroon". In : *Written Language & Literacy* 4.2 (2001), p. 131-162 *cf. p. 7, 8.*

[10] Germain Télesphore NDI. "Esquisse d'une analyse syntaxique de la phrase búlú : approche générative". Thèse de doct. Université de Yaoundé, 2001 *cf. p. 7.*

[11] Prosper Abéga. *Tonologie de la langue ewondo : l'ewondo sans les tons est une langue morte*. Presses de l'UCAC, 1998 *cf. p. 7.*

[12] Kenneth L Pike. "Tone Languages : A Technique for Determining the Number and Type of Pitch Contrasts in a Language, with Studies in Tonemic Substitution and Fusion." In : (1964) *cf. p. 7.*

[13] George N Clements. "The description of terraced-level tone languages". In : *Language* (1979), p. 536-558 *cf. p. 7.*

[14] Vincent Berment. "Méthodes pour informatiser les langues et les groupes de langues «peu dotées»". Thèse de doct. Université Joseph-Fourier-Grenoble I, 2004 *cf. p. 8, 9.*

[15] Viet Bac Le. "Reconnaissance automatique de la parole pour des langues peu dotées". Thèse de doct. Université Joseph-Fourier-Grenoble I, 2006 *cf. p. 8.*

[16] Sanjivani S Bhabad et Gajanan K Kharate. "An overview of technical progress in speech recognition". In : *International Journal of advanced research in computer science and software Engineering* 3.3 (2013), p. 488-497 *cf. p. 9, 10.*

[17] Ratnadeep Deshmukh et Abdulmalik Alasadi. "Automatic speech recognition techniques : A review". In : (2018) *cf. p. 9, 13.*

[18] Wiqas Ghai et Navdeep Singh. "Literature review on automatic speech recognition". In : *International Journal of Computer Applications* 41.8 (2012) *cf. p. 10.*

[19] Jean Louis K E Fendji et al. "Automatic Speech Recognition using limited vocabulary : A survey". In : *Applied Artificial Intelligence* 36.1 (2022), p. 2095039 *cf. p. 10.*

[20] Jean Louis KE Fendji et al. "Automatic Speech Recognition And Limited Vocabulary : A Survey". In : *arXiv preprint arXiv :2108.10254* (2021) *cf. p. 10.*

[21] MG Sarwar Murshed et al. "Machine learning at the network edge : A survey". In : *ACM Computing Surveys (CSUR)* 54.8 (2021), p. 1-37 *cf. p. 10.*

[22] Øystein Birkenes et al. "Penalized logistic regression with HMM log-likelihood regressors for speech recognition". In : *IEEE Transactions on Audio, Speech, and Language Processing* 18.6 (2009), p. 1440-1454 *cf. p. 11.*

[23] Tsung-Hui Chang et al. "A convex optimization method for joint mean and variance parameter estimation of large-margin CDHMM". In : *2008 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2008, p. 4053-4056 *cf. p. 11.*

[24] Haşim Sak et al. "Fast and accurate recurrent neural network acoustic models for speech recognition". In : *arXiv preprint arXiv :1507.06947* (2015) *cf. p. 11.*

[25] Mishaim Malik et al. "Automatic speech recognition : a survey". In : *Multimedia Tools and Applications* 80.6 (2021), p. 9411-9457 *cf. p. 11.*

[26] Hui Jiang, Xinwei Li et Chaojun Liu. "Large margin hidden Markov models for speech recognition". In : *IEEE transactions on audio, speech, and language processing* 14.5 (2006), p. 1584-1595 *cf. p. 11.*

[27] Isaac Westby et al. "FPGA acceleration on a multi-layer perceptron neural network for digit recognition". In : *The Journal of Supercomputing* 77.12 (2021), p. 14356-14373 *cf. p. 11.*

[28] Teuvo Kohonen. "Self-organized formation of topologically correct feature maps". In : *Biological cybernetics* 43.1 (1982), p. 59-69 *cf. p. 11.*

[29] RLK Venkateswarlu et R Vasantha Kumari. "Novel approach for speech recognition by using self—Organized maps". In : *2011 international conference on emerging trends in networks and computer communications (ETNCC)*. IEEE. 2011, p. 215-222 *cf. p. 11.*

[30] Thamir Rashed Saeed, Jabar Salman et A Hussein Ali. "Classification improvement of spoken arabic language based on radial basis function". In : *International Journal of Electrical and Computer Engineering IJECE* 9.1 (2019), p. 402-408 *cf. p. 12.*

[31] RLK Venkateswarlu, R Vasantha Kumari et G Vani Jayasri. "Speech recognition using radial basis function neural network". In : *2011 3rd international conference on electronics computer technology*. T. 3. IEEE. 2011, p. 441-445 *cf. p. 12.*

[32] Alex Graves et al. "Connectionist temporal classification : labelling unsegmented sequence data with recurrent neural networks". In : *Proceedings of the 23rd international conference on Machine learning*. 2006, p. 369-376 *cf. p. 12, 15.*

[33] Jane Oruh, Serestina Viriri et Adekanmi Adegun. "Long Short-Term Memory Recurrent Neural Network for Automatic Speech Recognition". In : *IEEE Access* 10 (2022), p. 30069-30079 *cf. p. 12.*

[34] Manish Dhakal et al. "Automatic speech recognition for the Nepali language using CNN, bidirectional LSTM and ResNet". In : *2022 International Conference on Inventive Computation Technologies (ICICT)*. IEEE. 2022, p. 515-521 *cf. p. 12.*

[35] Lilia Lazli et Mokhtar Sellami. "Connectionist probability estimators in HMM arabic speech recognition using fuzzy logic". In : *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer. 2003, p. 379-388 *cf. p. 12.*

[36] Nona Helmi et B Hoda Helmi. "Speech recognition with fuzzy neural network for discrete words". In : *2008 fourth international conference on natural computation*. T. 7. IEEE. 2008, p. 265-269 *cf. p. 12.*

[37] Yusra Faisal Al-Irhayim et Maher Khalaf Hussein. "Speech recognition of isolated Arabic words via using wavelet transformation and fuzzy neural network". In : *Computer engineering and intelligent systems* 7.3 (2016) *cf. p. 12.*

[38] Rubén Solera-Ureña et al. "Svms for automatic speech recognition : a survey". In : *Progress in nonlinear speech processing*. Springer, 2007, p. 190-216 *cf. p. 12.*

[39] Osman Eray, Sezai Tokat et Serdar Iplikci. "An application of speech recognition with support vector machines". In : *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE. 2018, p. 1-6 *cf. p. 12.*

[40] Qifeng Zhu et Abeer Alwan. "On the use of variable frame rate analysis in speech recognition". In : *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100)*. T. 3. IEEE. 2000, p. 1783-1786 *cf. p. 12.*

[41] Douglas O'Shaughnessy. "Linear predictive coding". In : *IEEE potentials* 7.1 (1988), p. 29-32 *cf. p. 12.*

[42] Hynek Hermansky et Nelson Morgan. "RASTA processing of speech". In : *IEEE transactions on speech and audio processing* 2.4 (1994), p. 578-589 *cf. p. 12.*

[43] Sergey Ioffe. "Probabilistic linear discriminant analysis". In : *European Conference on Computer Vision*. Springer. 2006, p. 531-542 *cf. p. 12.*

[44] Steven Davis et Paul Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences". In : *IEEE transactions on acoustics, speech, and signal processing* 28.4 (1980), p. 357-366 *cf. p. 13.*

[45] Hynek Hermansky. "Perceptual linear predictive (PLP) analysis of speech". In : *the Journal of the Acoustical Society of America* 87.4 (1990), p. 1738-1752 *cf. p. 13.*

[46] Saliha Benkerzaz, Youssef Elmir et Abdeslam Dennai. "A study on automatic speech recognition". In : *Journal of Information Technology Review* 10.3 (2019), p. 80-83 *cf. p. 13.*

[47] Jaspreet Kaur, Amitoj Singh et Virender Kadyan. "Automatic speech recognition system for tonal languages : state-of-the-art survey". In : *Archives of Computational Methods in Engineering* 28.3 (2021), p. 1039-1068 *cf. p. 13, 23.*

[48] Claytone Sikasote et Antonios Anastasopoulos. "BembaSpeech : A Speech Recognition Corpus for the Bemba Language". In : *arXiv preprint arXiv :2102.04889* (2021) *cf. p. 14.*

[49] Moussa Doumbouya, Lisa Einstein et Chris Piech. "Using radio archives for low-resource speech recognition : towards an intelligent virtual assistant for illiterate users". In : *Proceedings of the AAAI Conference on Artificial Intelligence*. T. 35. 17. 2021, p. 14757-14765 *cf. p. 14.*

[50] Hadrien Gelas, Laurent Besacier et François Pellegrino. "Developments of Swahili resources for an automatic speech recognition system". In : *Spoken Language Technologies for Under-Resourced Languages*. 2012 *cf. p. 14.*

[51] Bonaventure F. P. Dossou et Chris C. Emezue. "OkwuGbé : End-to-End Speech Recognition for Fon and Igbo". In : *CoRR* abs/2103.07762 (2021). arXiv : 2103.07762 *cf. p. 14.*

[52] Odetunji Àjadi. ODeloBi. "Recognition of Tones in Yorúbá Speech : Experiments With Artificial Neural Networks." In : *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks* 83 (2008), p. 23-48 *cf. p. 15.*

[53] Shahrul Azmi Mohd Yusof, Abdulwahab Funsho Atanda et M Hariharan. "A review of Yorúbá Automatic Speech Recognition". In : *2013 IEEE 3rd International Conference on System Engineering and Technology*. IEEE. 2013, p. 242-247 *cf. p. 15.*

[54] OA Adetunmbi, OO Obe et JN Iyanda. "Development of Standard Yorúbá speech-to-text system using HTK". In : *International Journal of Speech Technology* 19.4 (2016), p. 929-944 *cf. p. 15.*

[55] Matthew K Luka, F Ibikunle et O Gregory. "Neural network based Hausa language speech recognition". In : *(IJARAI) International Journal of Advanced Research in Artificial Intelligence* 1.2 (2012), p. 39-44 *cf. p. 15.*

[56] Tim Schlippe et al. "Hausa large vocabulary continuous speech recognition". In : *Spoken Language Technologies for Under-Resourced Languages*. 2012 *cf. p. 15.*

[57] Patrice Yemmene et Laurent Besacier. "Motivations, challenges, and perspectives for the development of an Automatic Speech Recognition System for the under-resourced Ngiemboon Language". In : *Proceedings of The First International Workshop on NLP Solutions for Under Resourced Languages (NSURL 2019) co-located with ICNLSP 2019-Short Papers*. 2019, p. 59-67 *cf. p. 15, 17.*

[58] Fatima Hamlaoui et al. "BULBasaa : A bilingual Basaá-French speech corpus for the evaluation of language documentation tools". In : *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. 2018 *cf. p. 15.*

[59] Elvis Mboning Tchiaze et Jules Assoumou. "NTeALan-AI/NLP/NLU Platforms For Sharing and Leveraging African Language Resources For Education In Africa". In : *International Conference Language Technologies for All (LT4All) : Enabling Linguistic Diversity and Multilingualism Worldwide*. 2019 *cf. p. 15.*

[60] Elvis Mboning Tchiaze. "Collaborative construction of lexicographic and parallel datasets for African languages : first assessment". In : *arXiv preprint arXiv :2103.16712* (2021) *cf. p. 15.*

[61] Qing Li et Caroline Yao. *Real-time concepts for embedded systems*. CRC press, 2003 *cf. p. 16.*

[62] Peter Marwedel. *Embedded System Design*. 1re éd. Kluwer Academic Publishers, 2003. isbn : 9781402076909 ; 1402076908 *cf. p. 16.*

[63] Ronald Waxman et al. *High-Level System Modeling*. Springer, 1996 *cf. p. 16.*

[64] Hermann Kopetz et Real-Time Sytems. "Design principles for distributed embedded applications". In : *Real-Time Systems. Springer* (1997) *cf. p. 16.*

[65] King L Tai. "System-In-Package (SIP) : challenges and opportunities". In : *Proceedings 2000. Design Automation Conference.(IEEE Cat. No. 00CH37106)*. IEEE. 2000, p. 191-196 *cf. p. 17.*

[66] Aloysius Ngefac. "Linguistic choices in postcolonial multilingual Cameroon". In : *Nordic Journal of African Studies* 19.3 (2010), p. 16-16 *cf. p. 18.*

[67] Jayashri Vajpai et Avnish Bora. "Industrial applications of automatic speech recognition systems". In : *International Journal of Engineering Research and Applications* 6.3 (2016), p. 88-95 *cf. p. 18.*

[68] Agence de régulation des télécommunications Cameroun. *OBSERVATOIRE ANNUEL 2019 DU MARCHE DES COMMUNICATIONS ELECTRONIQUES*. http://art.cm. [Online ; accessed 22-September-2022]. 2019 *cf. p. 18.*

[69] Moeid M Elsokah, Hana H Saleh et Amer R Ze. "Next Generation Home Automation System Based on Voice Recognition". In : *Proceedings of the 6th International Conference on Engineering & MIS 2020*. 2020, p. 1-7 *cf. p. 19.*

[70] Chinju Paul, Amal Ganesh et C Sunitha. "An overview of IoT based smart homes". In : *2018 2nd International Conference on Inventive Systems and Control (ICISC)*. IEEE. 2018, p. 43-46 *cf. p. 19.*

[71] Chee-Pun Ooi et al. "FPGA-based embedded architecture for IoT home automation application". In : *Indonesian Journal of Electrical Engineering and Computer Science* 14.2 (2019), p. 646-652 *cf. p. 19.*

[72] Mohammed Hasan Ali et Qasim Mohamed Al Azze. "Design and implementation a security system for bank using voice recognition". In : *International Journal of Power Electronics and Drive Systems* 10.4 (2019), p. 2126 *cf. p. 19.*

[73] Stephen Jonathan Melnikoff. "Speech recognition in programmable logic". In : (2003) *cf. p. 19.*

[74] Erasmus Muh, Sofiane Amara et Fouzi Tabet. "Sustainable energy policies in Cameroon : A holistic overview". In : *Renewable and Sustainable Energy Reviews* 82 (2018), p. 3420-3429 *cf. p. 20.*

[75] Peiyan Dong et al. "Rtmobile : Beyond real-time mobile acceleration of rnns for speech recognition". In : *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, p. 1-6 *cf. p. 20.*

[76] Lam D Pham et al. "An ASIC-Based Artificial Neural Network Applied Real-time Speech Recognition SOPC". In : *Journal of Science and Technology : Issue on Information and Communications Technology* 2.1 (2016), p. 38-48 *cf. p. 20.*

[77] Eshaan Mudgal et al. "Template Based Real-Time Speech Recognition Using Digital Filters on DSP-TMS320F28335". In : *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. IEEE. 2018, p. 1-6 *cf. p. 20.*

[78] Iacopo Casalini, Marco Marini et Luca Fanucci. "FPGA Implementation of a Configurable Vocal Feature Extraction Embedded System for Dysarthric Speech Recognition". In : *International Conference on Applications in Electronics Pervading Industry, Environment and Society*. Springer. 2022, p. 221-228 *cf. p. 20.*

[79] Kaiwei Zou et al. "Learn-to-scale : Parallelizing deep learning inference on chip multiprocessor architecture". In : *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, p. 1172-1177 *cf. p. 20.*

[80] Antoine Owona. "L'orthographe harmonisée de l'ewondo". Thèse de doct. Université de Yaoundé, 2004 *cf. p. 21.*

[81] Ashima Arora, Virender Kadyan et Amitoj Singh. "Effect of tonal features on various dialectal variations of Punjabi language". In : *Advances in Signal Processing and Communication*. Springer, 2019, p. 467-475 *cf. p. 21.*

[82] Jike Chong et al. "Opportunities and challenges of parallelizing speech recognition". In : *Proceedings of the 2nd USENIX conference on Hot topics in parallelism. HotPar*. T. 10. 2010, p. 2-2 *cf. p. 21.*

[83] Wolfgang Hess. *Pitch determination of speech signals : algorithms and devices*. T. 3. Springer Science & Business Media, 2012 *cf. p. 22.*

[84] Wolfgang J Hess. "Pitch Determination of Speech Signals—A Survey". In : *Spoken Language Generation and Understanding*. Springer, 1980, p. 263-278 *cf. p. 22.*

[85] Paul Boersma et al. "Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound". In : *Proceedings of the institute of phonetic sciences*. T. 17. 1193. Citeseer. 1993, p. 97-110 *cf. p. 22, 23, 30.*

[86] David Talkin et W Bastiaan Kleijn. "A robust algorithm for pitch tracking (RAPT)". In : *Speech coding and synthesis* 495 (1995), p. 518 *cf. p. 22.*

[87] Alain De Cheveigné et Hideki Kawahara. "YIN, a fundamental frequency estimator for speech and music". In : *The Journal of the Acoustical Society of America* 111.4 (2002), p. 1917-1930 *cf. p. 22.*

[88] Arturo Camacho et John G Harris. "A sawtooth waveform inspired pitch estimator for speech and music". In : *The Journal of the Acoustical Society of America* 124.3 (2008), p. 1638-1652 *cf. p. 22.*

[89] Dik J Hermes. "Measurement of pitch by subharmonic summation". In : *The journal of the acoustical society of America* 83.1 (1988), p. 257-264 *cf. p. 22.*

[90] Alexander Sorin et al. "The ETSI extended distributed speech recognition (DSR) standards : client side processing and tonal language recognition evaluation". In : *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*. T. 1. IEEE. 2004, p. I-129 *cf. p. 22.*

[91] Hideki Kawahara et al. "Nearly defect-free F0 trajectory extraction for expressive speech modifications based on STRAIGHT". In : *Ninth European Conference on Speech Communication and Technology*. 2005 *cf. p. 22, 23.*

[92] Denis Jouvet et Yves Laprie. "Performance analysis of several pitch detection algorithms on simulated and real noisy speech data". In : *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE. 2017, p. 1614-1618 *cf. p. 22.*

[93] Sofia Strömbergsson. "Today's Most Frequently Used F0 Estimation Methods, and Their Accuracy in Estimating Male and Female Pitch in Clean Speech." In : *Interspeech*. Dresden. 2016, p. 525-529 *cf. p. 22, 23.*

Références du Chapitre 3: Conception d'un système embarqué parallèle sur architecture reconfigurable

[85] Paul Boersma et al. "Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound". In : *Proceedings of the institute of phonetic sciences*. T. 17. 1193. Citeseer. 1993, p. 97-110 *cf. p. 22, 23, 30.*

[94] Michael J Flynn. "Some computer organizations and their effectiveness". In : *IEEE transactions on computers* 100.9 (1972), p. 948-960 *cf. p. 25.*

[95] William Gropp et al. *Using MPI : portable parallel programming with the message-passing interface*. T. 1. MIT press, 1999 *cf. p. 26.*

[96] Bradford Nichols, Dick Buttlar et Jacqueline Proulx Farrell. *Pthreads Programming*. USA : O'Reilly & Associates, Inc., 1996. isbn : 1565921151 *cf. p. 27.*

[97] Mitsuhisa Sato. "OpenMP : parallel programming API for shared memory multiprocessors and on-chip multiprocessors". In : *Proceedings of the 15th international symposium on System Synthesis*. 2002, p. 109-111 *cf. p. 27.*

[98] Robert D Blumofe et al. "Cilk : An efficient multithreaded runtime system". In : *ACM SigPlan Notices* 30.8 (1995), p. 207-216 *cf. p. 27.*

[99] James Reinders. *Intel threading building blocks : outfitting C++ for multi-core processor parallelism*. " O Reilly Media, Inc.", 2007 *cf. p. 27.*

[100] Marc Snir et al. *MPI–the Complete Reference : the MPI core*. T. 1. MIT press, 1998 *cf. p. 28.*

[101] Marco Aldinucci et Marco Danelutto. "Skeleton-based parallel programming : Functional and parallel semantics in a single shot". In : *Computer Languages, Systems & Structures* 33.3-4 (2007), p. 179-192 *cf. p. 28.*

[102] Christoph Kessler et Jörg Keller. "Models for parallel computing : Review and perspectives". In : *Mitteilungen-Gesellschaft für Informatik eV, Parallel-Algorithmen und Rechnerstrukturen* 24 (2007), p. 13-29 *cf. p. 28.*

[103] I. Foster et J. Foster. *Designing and Building Parallel Programs : Concepts and Tools for Parallel Software Engineering*. Literature and Philosophy. Addison-Wesley, 1995. isbn : 9780201575941 *cf. p. 28, 32, 67, 74.*

[104] Jerry C Yan et al. *The Automated Instrumentation and Monitoring System (AIMS) : Design and Architecture*. Rapp. tech. 1997 *cf. p. 30, 69.*

[105] Anthony Chan, William Gropp et Ewing Lusk. *User's guide for mpe extensions for mpi programs*. Rapp. tech. Technical Report ANL-98/xx, Argonne National Laboratory, 1998. The updated . . ., 1998 *cf. p. 30, 69.*

[106] Daniel A Reed et al. "Scalable performance analysis : The Pablo performance analysis environment". In : *Proceedings of Scalable Parallel Libraries Conference*. IEEE. 1993, p. 104-113 *cf. p. 30, 69.*

[107] Luiz De Rose, Ying Zhang et Daniel A Reed. "SvPablo : A multi-language performance analysis system". In : *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer. 1998, p. 352-355 *cf. p. 30, 69.*

[108] B Miller et Jeff Hollingsworth. *Paradyn parallel performance tools*. 2000 *cf. p. 30, 69.*

[109] Jim Galarowicz et Bernd Mohr. *Analyzing message passing programs on the Cray T3E with PAT and VAMPIR*. Citeseer, 1998 *cf. p. 30, 69.*

[110] Jean Claude Simon. *Spoken Language Generation and Understanding : Proceedings of the NATO Advanced Study Institute Held at Bonas, France, June 26–July 7, 1979*. T. 59. Springer Science & Business Media, 2012 *cf. p. 30.*

[111] David Weenink. "Speech signal processing with Praat". In : *Haettu* 16 (2014), p. 2014 *cf. p. 31.*

[112] William Gropp, Steven Huss-Lederman et Marc Snir. *MPI : the complete reference. The MPI-2 extensions*. T. 2. Mit Press, 1998 *cf. p. 33.*

[113] Edgar Gabriel et al. "Open MPI : Goals, concept, and design of a next generation MPI implementation". In : *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer. 2004, p. 97-104 *cf. p. 33, 74, 77.*

[114] Andreas Knüpfer et al. "Score-p : A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir". In : *Tools for High Performance Computing 2011*. Springer, 2012, p. 79-91 *cf. p. 34, 69, 74.*

[115] Robert Bell, Allen D Malony et Sameer Shende. "Paraprof : A portable, extensible, and scalable tool for parallel performance profile analysis". In : *European Conference on Parallel Processing*. Springer. 2003, p. 17-26 *cf. p. 34, 74.*

[116] Praveenkumar Babu et Eswaran Parthasarathy. "Reconfigurable FPGA architectures : A survey and applications". In : *Journal of The Institution of Engineers (India) : Series B* 102.1 (2021), p. 143-156 *cf. p. 38.*

[117] Neelesh Ranjan Srivastava et Vikas Mittal. "A Review on Hardware Implementations of Signal Processing Algorithms". In : *Latest Trends in Renewable Energy Technologies* (2021), p. 295-302 *cf. p. 39.*

[118] Victor Lomas-Barrie et al. "Fuzzy artmap-based fast object recognition for robots using FPGA". In : *Electronics* 10.3 (2021), p. 361 *cf. p. 39.*

[119] Ahmed Jawad A AlBdairi et al. "Face Recognition Based on Deep Learning and FPGA for Ethnicity Identification". In : *Applied Sciences* 12.5 (2022), p. 2605 *cf. p. 39.*

[120] Eduardo Alcaién et al. "Hardware Architectures for Real-Time Medical Imaging". In : *Electronics* 10.24 (2021), p. 3118 *cf. p. 39.*

[121] Abdulmajeed Adil Yazdeen et al. "FPGA implementations for data encryption and decryption via concurrent and parallel computation : A review". In : *Qubahan Academic Journal* 1.2 (2021), p. 8-16 *cf. p. 40.*

[122] Alvaro Cintas Canto, Mehran Mozaffari Kermani et Reza Azarderakhsh. "Reliable Constructions for the Key Generator of Code-Based Post-Quantum Cryptosystems on FPGA". In : *ACM Journal on Emerging Technologies in Computing Systems (JETC)* (2022) *cf. p. 40.*

[123] Stefano Corda et al. "Near memory acceleration on high resolution radio astronomy imaging". In : *2020 9th Mediterranean Conference on Embedded Computing (MECO)*. IEEE. 2020, p. 1-6 *cf. p. 40.*

[124] Huaixiang Hu et al. "Design and Implementation of Intelligent Speech Recognition System Based on FPGA". In : *Journal of Physics : Conference Series*. T. 2171. 1. IOP Publishing. 2022, p. 012010 *cf. p. 40.*

[125] Weili Liu. "Voice control system based on Zynq FPGA". In : *Journal of Physics : Conference Series*. T. 1631. 1. IOP Publishing. 2020, p. 012177 *cf. p. 40.*

[126] Sa Wang. "Online English translation information retrieval based on FPGA and Internet of Things". In : *Microprocessors and Microsystems* (2020), p. 103387 *cf. p. 40.*

[127] Muhammad Abdullah Rafique et al. "A cost and resource efficient telemetry host station design using FPGA". In : *2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. IEEE. 2018, p. 799-804 *cf. p. 40.*

[128] Christophe Bobda et al. "The future of FPGA acceleration in datacenters and the cloud". In : *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 15.3 (2022), p. 1-42 *cf. p. 40.*

[129] John L. Gustafson. "Moore's Law". In : *Encyclopedia of Parallel Computing*. Sous la dir. de David Padua. Boston, MA : Springer US, 2011, p. 1177-1184. isbn : 978-0-387-09766-4. doi : 10.1007/978-0-387-09766-4_81 *cf. p. 42.*

[130] Semiconductor Industry Association. *2011 International Technology Roadmap for Semiconductors (ITRS) - design*. https://www.semiconductors.org/wp-content/uploads/2018/08/2011Design.pdf. [Online ; accessed 01-September-2022]. 2011 *cf. p. 42, 43.*

[131] Maxime Pelcat et al. "Design productivity of a high level synthesis compiler versus HDL". In : *2016 International Conference on Embedded Computer Systems : Architectures, Modeling and Simulation (SAMOS)*. IEEE. 2016, p. 140-147 *cf. p. 42.*

[132] Maxime Pelcat. "Models, Methods and Tools for Bridging the Design Productivity Gap of Embedded Signal Processing Systems". Thèse de doct. Université Clermont Auvergne, 2017 *cf. p. 43, 44.*

[133] Taho Dorta et al. "Reconfigurable multiprocessor systems : a review". In : *International Journal of Reconfigurable Computing* 2010 (2010) *cf. p. 43.*

[134] Jean-Pierre Deschamps, Gustavo D Sutter et Enrique Cantó. "Eda tools". In : *Guide to FPGA Implementation of Arithmetic Functions*. Springer, 2012, p. 127-151 *cf. p. 45, 46.*

[135] Intel. *Intel® Quartus® Prime Software*. https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html. [Online ; accessed 01-September-2022]. 2022 *cf. p. 45, 52.*

[136] Tom Feist. "Vivado design suite". In : *White Paper* 5 (2012), p. 30 *cf. p. 45.*

[137] "IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows". In : *IEEE Std 1685-2014 (Revision of IEEE Std 1685-2009)* (2014), p. 1-510. doi : 10.1109/IEEESTD.2014.6898803 *cf. p. 46.*

[138] Microsemi. *Libero SoC Design Suite*. https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/fpga/libero-software-later-versions. [Online ; accessed 01-September-2022]. 2022 *cf. p. 46.*

[139] Microsemi. *Fusion*. https://www.microsemi.com/product-directory/fpgas/1691-fusion. [Online ; accessed 01-September-2022]. 2022 *cf. p. 46.*

[140] Lattice Semiconductor. *Lattice Diamond*. https://www.latticesemi.com/Products/DesignSoftwareAndIP/FPGAandLDS/LatticeDiamond. [Online ; accessed 01-September-2022]. 2022 *cf. p. 46.*

[141] *ispLEVER Classic Software*. https://www.latticesemi.com/ispleverclassic. [Online ; accessed 01-September-2022]. 2022 *cf. p. 46.*

[142] Adam T Arnesen. "Increasing Design Productivity for FPGAs Through IP Reuse and Meta-Data Encapsulation". Thèse de doct. Brigham Young University, 2011 *cf. p. 47.*

[143] Emil Girczyc et Steve Carlson. "Increasing design quality and engineering productivity through design reuse". In : *Proceedings of the 30th international Design Automation Conference*. 1993, p. 48-53 *cf. p. 47.*

[144] M Själander. *HMS Multiplier Generator*. 2008 *cf. p. 48.*

[145] Florent De Dinechin et Bogdan Pasca. "Designing custom arithmetic data paths with FloPoCo". In : *IEEE Design & Test of Computers* 28.4 (2011), p. 18-27 *cf. p. 48.*

[146] Panagiotis D Vouzis, Sylvain Collange et Mark G Arnold. "A novel cotransformation for LNS subtraction". In : *Journal of Signal Processing Systems* 58.1 (2010), p. 29-40 *cf. p. 48.*

[147] David B Thomas. "Parallel generation of gaussian random numbers using the table-hadamard transform". In : *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE. 2013, p. 161-168 *cf. p. 48.*

[148] David B Thomas. "A general-purpose method for faithfully rounded floating-point function approximation in FPGAs". In : *2015 IEEE 22Nd symposium on computer arithmetic*. IEEE. 2015, p. 42-49 *cf. p. 48.*

[149] H Fatih Ugurdag et al. "Hardware division by small integer constants". In : *IEEE Transactions on Computers* 66.12 (2017), p. 2097-2110 *cf. p. 48.*

[150] Matei Istoan et Florent De Dinechin. "Automating the pipeline of arithmetic datapaths". In : *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE. 2017, p. 704-709 *cf. p. 48.*

[151] Anastasia Volkova et al. "Towards hardware IIR filters computing just right : Direct form I case study". In : *IEEE Transactions on Computers* 68.4 (2018), p. 597-608 *cf. p. 48.*

[152] Matei Istoan et Florent de Dinechin. "Pipeline automatique d'opérateurs dans FloPoCo 5.0". In : *COMPAS'2016 : Conférence d'informatique en Parallélisme, Architecture et Système*. 2016 *cf. p. 49.*

[153] Leandro Mateus Giacomini Rocha et al. "Framework-based Arithmetic Datapath Generation to Explore Parallel Binary Multipliers". In : *Journal of Integrated Circuits and Systems* 15.3 (2020), p. 1-10 *cf. p. 48.*

[154] Philippe Coussy et al. "An introduction to high-level synthesis". In : *IEEE Design & Test of Computers* 26.4 (2009), p. 8-17 *cf. p. 50.*

[155] Mostafa W Numan et al. "Towards automatic high-level code deployment on reconfigurable platforms : A survey of high-level synthesis tools and toolchains". In : *IEEE Access* 8 (2020), p. 174692-174722 *cf. p. 51.*

[156] Lan Huang et al. "A survey on performance optimization of high-level synthesis tools". In : *Journal Of Computer Science and Technology* 35.3 (2020), p. 697-720 *cf. p. 51.*

[157] Radosław Cieszewski et al. "Review of parallel computing methods and tools for FPGA technology". In : *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2013*. T. 8903. SPIE. 2013, p. 596-608 *cf. p. 51.*

[158] Xilinx. "Vivado High-Level Synthesis." In : (2022) *cf. p. 51, 70.*

[159] Zhiru Zhang et al. "AutoPilot : A platform-based ESL synthesis system". In : *High-Level Synthesis*. Springer, 2008, p. 99-112 *cf. p. 51.*

[160] *The LLVM Compiler Infrastructure*. https://www.llvm.org/. [Online ; accessed 01-September-2022]. 2022 *cf. p. 51.*

[161] Khronos Group Inc. *OpenCL Overview*. https://www.khronos.org/opencl/. [Online ; accessed 01-September-2022]. 2022 *cf. p. 52.*

[162] Intel. *Intel FPGA SDK for OpenCL*. https://www.intel.com.au/content/www/au/en/software/programmable/sdk-for-opencl/overview.html. [Online ; accessed 01-September-2022]. 2022 *cf. p. 52.*

[163] *Clang : a C language family frontend for LLVM*. https://clang.llvm.org/. [Online ; accessed 01-September-2022]. 2022 *cf. p. 52.*

[164] Andrew Canis et al. "LegUp : high-level synthesis for FPGA-based processor/accelerator systems". In : *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. 2011, p. 33-36 *cf. p. 52.*

[165] Intel. *Intel HLS Compiler*. https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.htm. [Online ; accessed 01-September-2022]. 2022 *cf. p. 52.*

[166] Siemens Digital Industries Software. *Catapult High-Level Synthesis and Verification*. https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/. [Online ; accessed 01-September-2022]. 2022 *cf. p. 53.*

[167] Thomas Bollaert. "Catapult synthesis : a practical introduction to interactive C synthesis". In : *High-Level Synthesis*. Springer, 2008, p. 29-52 *cf. p. 53.*

[168] Jens Rettkowski et Diana Göhringer. "Sdmpsoc : Software-defined mpsoc for fpgas". In : *Journal of Signal Processing Systems* 92.10 (2020), p. 1187-1196 *cf. p. 53, 78.*

[169] Hayden Kwok-Hay So et Cheng Liu. "FPGA overlays". In : *FPGAs for Software Programmers*. Springer, 2016, p. 285-305 *cf. p. 54, 55.*

[170] Masudul Hassan Quraishi, Erfan Bank Tavakoli et Fengbo Ren. "A survey of system architectures and techniques for FPGA virtualization". In : *IEEE Transactions on Parallel and Distributed Systems* 32.9 (2021), p. 2216-2230 *cf. p. 54.*

[171] Xiangwei Li et Douglas L Maskell. "Time-multiplexed FPGA overlay architectures : A survey". In : *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 24.5 (2019), p. 1-19 *cf. p. 54.*

[172] Riadh Ben Abdelhamid, Yoshiki Yamaguchi et Taisuke Boku. "A highly-efficient and tightly-connected many-core overlay architecture". In : *IEEE Access* 9 (2021), p. 65277-65292 *cf. p. 54.*

[173] Riadh Ben Abdelhamid, Yoshiki Yamguchi et Taisuke Boku. "Condensing an overload of parallel computing ingredients into a single architecture recipe". In : *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2020, p. 25-28 *cf. p. 54, 60, 67.*

[174] Kentaro Sano et al. "FPGA-array with bandwidth-reduction mechanism for scalable and power-efficient numerical simulations based on finite difference methods". In : *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 3.4 (2010), p. 1-35 *cf. p. 55, 67.*

[175] Kurtis T. Johnson, Ali R Hurson et Behrooz Shirazi. "General-purpose systolic arrays". In : *Computer* 26.11 (1993), p. 20-31 *cf. p. 55.*

[176] Duncan G Elliott et al. "Computational RAM : Implementing processors in memory". In : *IEEE Design & Test of Computers* 16.1 (1999), p. 32-41 *cf. p. 55.*

[177] Kolin Paul, Chinmaya Dash et Mansureh Shahraki Moghaddam. "reMORPH : a run-time reconfigurable architecture". In : *2012 15th Euromicro Conference on Digital System Design*. IEEE. 2012, p. 26-33 *cf. p. 56, 67.*

[178] Kentaro Sano, Yoshiaki Hatsuda et Satoru Yamamoto. "Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth". In : *IEEE Transactions on Parallel and Distributed Systems* 25.3 (2013), p. 695-705 *cf. p. 57, 67.*

[179] Jan Gray. "Grvi phalanx : A massively parallel risc-v fpga accelerator accelerator". In : *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2016, p. 17-20 *cf. p. 58, 59, 67.*

[180] Nachiket Kapre et Jan Gray. "Hoplite : Building austere overlay nocs for fpgas". In : *2015 25th international conference on field programmable logic and applications (FPL)*. IEEE. 2015, p. 1-8 *cf. p. 58.*

[181] Aaron Severance et Guy GF Lemieux. "Embedded supercomputing in FPGAs with the VectorBlox MXP matrix processor". In : *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE. 2013, p. 1-10 *cf. p. 58.*

[182] Charles Eric LaForest et John Gregory Steffan. "Octavo : an FPGA-centric processor family". In : *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. 2012, p. 219-228 *cf. p. 59, 67.*

[183] Charles Eric LaForest, Jason Anderson et J Gregory Steffan. "Approaching overhead-free execution on FPGA soft-processors". In : *2014 International Conference on Field-Programmable Technology (FPT)*. IEEE. 2014, p. 99-106 *cf. p. 59.*

[184] Charles Eric Laforest et Jason H Anderson. "Microarchitectural comparison of the MXP and Octavo soft-processor FPGA overlays". In : *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 10.3 (2017), p. 1-25 *cf. p. 59.*

[185] Jan Gray. "2GRVI Phalanx : A 1332-core RISC-V RV64I processor cluster array with an HBM2 high bandwidth memory system and an OpenCL-like programming model in a Xilinx VU37P FPGA [WIP report]". In : *Proc. Int. Workshop (H2RC)*. 2019 *cf. p. 59.*

[186] Mike Wissolik et al. "Virtex UltraScale+ HBM FPGA : a revolutionary increase in memory performance". In : *Xilinx Whitepaper* (2017) *cf. p. 60.*

[187] Roseli Uhlendorf et al. "An MPI-based MPSoC Platform in FPGA". In : *IEEE Latin America Transactions* 19.4 (2021), p. 697-705 *cf. p. 61.*

[188] Wajid Hassan Minhass, Johnny Öberg et Ingo Sander. "Design and implementation of a plesiochronous multi-core 4x4 network-on-chip fpga platform with mpi hal support". In : *Proceedings of the 6th FPGAworld Conference*. 2009, p. 52-57 *cf. p. 62.*

[189] Philipp Mahr et al. "Soc-mpi : A flexible message passing library for multiprocessor systems-on-chips". In : *2008 International Conference on Reconfigurable Computing and FPGAs*. IEEE. 2008, p. 187-192 *cf. p. 62, 83.*

[190] Manuel Saldana et Paul Chow. "TMD-MPI : An MPI implementation for multiple processors across multiple FPGAs". In : *2006 International Conference on Field Programmable Logic and Applications*. IEEE. 2006, p. 1-6 *cf. p. 62.*

[191] Gamom Ngounou Ewo et Roland Christian. "Déploiement d'applications parallèles sur une architecture distribuée matériellement reconfigurable". Thèse de doct. Cergy-Pontoise, 2015 *cf. p. 62, 65, 67, 74, 83, 97.*

Références du Chapitre 4: Proposition d'une approche de facilitation de la co-conception d'applications parallèles MPI sur MPRSoC

[103] I. Foster et J. Foster. *Designing and Building Parallel Programs : Concepts and Tools for Parallel Software Engineering*. Literature and Philosophy. Addison-Wesley, 1995. isbn : 9780201575941 *cf. p. 28, 32, 67, 74.*

[104] Jerry C Yan et al. *The Automated Instrumentation and Monitoring System (AIMS) : Design and Architecture*. Rapp. tech. 1997 *cf. p. 30, 69.*

[105] Anthony Chan, William Gropp et Ewing Lusk. *User's guide for mpe extensions for mpi programs*. Rapp. tech. Technical Report ANL-98/xx, Argonne National Laboratory, 1998. The updated . . ., 1998 *cf. p. 30, 69.*

[106] Daniel A Reed et al. "Scalable performance analysis : The Pablo performance analysis environment". In : *Proceedings of Scalable Parallel Libraries Conference*. IEEE. 1993, p. 104-113 *cf. p. 30, 69.*

[107] Luiz De Rose, Ying Zhang et Daniel A Reed. "SvPablo : A multi-language performance analysis system". In : *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer. 1998, p. 352-355 *cf. p. 30, 69.*

[108] B Miller et Jeff Hollingsworth. *Paradyn parallel performance tools*. 2000 *cf. p. 30, 69.*

[109] Jim Galarowicz et Bernd Mohr. *Analyzing message passing programs on the Cray T3E with PAT and VAMPIR*. Citeseer, 1998 *cf. p. 30, 69.*

[113] Edgar Gabriel et al. "Open MPI : Goals, concept, and design of a next generation MPI implementation". In : *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer. 2004, p. 97-104 *cf. p. 33, 74, 77.*

[114] Andreas Knüpfer et al. "Score-p : A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir". In : *Tools for High Performance Computing 2011*. Springer, 2012, p. 79-91 *cf. p. 34, 69, 74.*

[115] Robert Bell, Allen D Malony et Sameer Shende. "Paraprof : A portable, extensible, and scalable tool for parallel performance profile analysis". In : *European Conference on Parallel Processing*. Springer. 2003, p. 17-26 *cf. p. 34, 74.*

[158] Xilinx. "Vivado High-Level Synthesis." In : (2022) *cf. p. 51, 70.*

[168] Jens Rettkowski et Diana Göhringer. "Sdmpsoc : Software-defined mpsoc for fpgas". In : *Journal of Signal Processing Systems* 92.10 (2020), p. 1187-1196 *cf. p. 53, 78.*

[173] Riadh Ben Abdelhamid, Yoshiki Yamguchi et Taisuke Boku. "Condensing an overload of parallel computing ingredients into a single architecture recipe". In : *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2020, p. 25-28 *cf. p. 54, 60, 67.*

[174] Kentaro Sano et al. "FPGA-array with bandwidth-reduction mechanism for scalable and power-efficient numerical simulations based on finite difference methods". In : *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 3.4 (2010), p. 1-35 *cf. p. 55, 67.*

[177] Kolin Paul, Chinmaya Dash et Mansureh Shahraki Moghaddam. "reMORPH : a run-time reconfigurable architecture". In : *2012 15th Euromicro Conference on Digital System Design*. IEEE. 2012, p. 26-33 *cf. p. 56, 67.*

[178] Kentaro Sano, Yoshiaki Hatsuda et Satoru Yamamoto. "Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth". In : *IEEE Transactions on Parallel and Distributed Systems* 25.3 (2013), p. 695-705 *cf. p. 57, 67.*

[179] Jan Gray. "Grvi phalanx : A massively parallel risc-v fpga accelerator accelerator". In : *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2016, p. 17-20 *cf. p. 58, 59, 67.*

[182] Charles Eric LaForest et John Gregory Steffan. "Octavo : an FPGA-centric processor family". In : *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. 2012, p. 219-228 *cf. p. 59, 67.*

[191] Gamom Ngounou Ewo et Roland Christian. "Déploiement d'applications parallèles sur une architecture distribuée matériellement reconfigurable". Thèse de doct. Cergy-Pontoise, 2015 *cf. p. 62, 65, 67, 74, 83, 97.*

[192] Pavel Zaykov. "MIMD implementation with PicoBlaze microprocessor using MPI functions". In : *Proceedings of the 2007 international conference on Computer systems and technologies*. 2007, p. 1-7 *cf. p. 65.*

[193] Roland Christian Gamom Ngounou Ewo et al. "Hardware mpi-2 functions for multi-processing reconfigurable system on chip". In : *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE. 2013, p. 273-280 *cf. p. 71.*

[194] Peter Rössler et al. "Survey and comparison of digital logic simulators". In : *2019 Austrochip Workshop on Microelectronics (Austrochip)*. IEEE. 2019, p. 87-92 *cf. p. 71.*

[195] XST Xilinx. *User Guide. V. 11.3*. 2018 *cf. p. 73.*

[196] Xilinx. *Vitis High-Level Synthesis User Guide*. 2020 *cf. p. 74.*

Références du Chapitre 5: Conception de XMATIP : une plateforme pour la co-conception des applications parallèles MPI-2 RMA

[189] Philipp Mahr et al. "Soc-mpi : A flexible message passing library for multiprocessor systems-on-chips". In : *2008 International Conference on Reconfigurable Computing and FPGAs*. IEEE. 2008, p. 187-192 *cf. p. 62, 83.*

[191] Gamom Ngounou Ewo et Roland Christian. "Déploiement d'applications parallèles sur une architecture distribuée matériellement reconfigurable". Thèse de doct. Cergy-Pontoise, 2015 *cf. p. 62, 65, 67, 74, 83, 97.*

[197] Thomas Beisel, Edgar Gabriel et Michael Resch. "An extension to MPI for distributed computing on MPPs". In : *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer. 1997, p. 75-82 *cf. p. 83, 84, 86, 89.*

[198] Olivier Aumage. "Heterogeneous multi-cluster networking with the Madeleine III communication library". In : *Proceedings 16th International Parallel and Distributed Processing Symposium*. IEEE. 2002, 12-pp *cf. p. 83.*

[199] Nicholas T Karonis, Brian Toonen et Ian Foster. "MPICH-G2 : A grid-enabled implementation of the message passing interface". In : *Journal of Parallel and Distributed Computing* 63.5 (2003), p. 551-563 *cf. p. 83.*

[200] Daniel Balkanski, Mario Trams et Wolfgang Rehm. "Communication Middleware Systems for Heterogenous Clusters : A Comparative Study." In : *CLUSTER*. 2003, p. 504-507 *cf. p. 83.*

[201] FIFO AXI4-Stream. "v4. 1 LogiCORE IP Product Guide". In : *Xilinx, pG080* (2019) *cf. p. 94.*

[202] Gregory R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. 1re éd. Addison-Wesley, 1999. isbn : 0201357526 ; 9780201357523 *cf. p. 97.*

Annexe

A Annexes

A.1 Extrait de code VHDL du fichier *HCL_Arch_conf.vhd* pour la configuration des tâches statiques et dynamiques dans MATIP

```
1  --  Package File Template
2  --
3  --  Purpose: This package defines the size of NoC ports and the number
4  -- of hardware tasks to create with the Mpi HCL environment.
5  -- The NoC_size parameter may vary from 2 to 16
6  -- The STATIC_HT parameter must be less or equal to the NoC_Size parameter
7  -- The DYN_ALLOWED authorize the binding of HT template environment to all the
       Noc Ports
8
9  library IEEE;
10 use IEEE.STD_LOGIC_1164.all;
11
12 package Hcl_Arch_conf is
13
14
15
16 -- Declare constants
17
18   constant NOC_SIZE   : integer :=6; --(2 to 16)indicates the number of NoC port
       to instantiate
19   constant STATIC_HT    : integer :=6;--(2 to NOC_SIZE)gives the the number of
      HT to Hardwire
20   constant DYN_ALLOWED  : std_logic:='0'; --allow the creation of dynamic
      hardware tasks
21 -- Declare functions and procedure
22
23 end Hcl_Arch_conf;
24
25
26 package body Hcl_Arch_conf is
27
28
29
30 end Hcl_Arch_conf;
```

Listing A.1: *Extrait de code VHDL du fichier HCL_Arch_conf.vhd pour la configuration des tâches statiques et dynamiques dans MATIP*

A.2 Extrait de code VHDL du fichier *HT_stat.vhd* pour description des tâches statiques dans MATIP

```
1  --
       ------------------------------------------------------------------------------
```

```
-- Company:
-- Engineer:
--
-- Create Date:    12:04:21 04/22/2013
-- Design Name:
-- Module Name:    HT_process - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--
    ----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;
library NocLib ;

use std.textio.all;
use NocLib.CoreTypes.all;
Library MPI_HCL;
use MPI_HCL.Packet_type.all;
use MPI_HCL.MPI_RMA.all;
use work.Hcl_Arch_conf.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity HT_stat is
    generic (Task_id : natural:=0);
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
        en : in std_logic; -- active la tache
           Interf_i : in  core_i; --signaux pour l'interface IO
        Interf_o : out  core_o; --signaux pour l'interface IO
           mem_o : out typ_dpram_o; -- signaux pour l'acces a la memoire
        mem_i : in typ_dpram_i; -- signaux pour l'acces a la memoire
        ct_out :out unsigned(7 downto 0);
        ht_state : out typ_mae;
        PE_in : in STD_LOGIC_VECTOR (Word-1 downto 0); --port GPIO pour le PE
        PE_out : out STD_LOGIC_VECTOR (Word-1 downto 0) --port GPIO pour le PE

        );
end HT_stat;

architecture A1 of HT_stat is
    constant simres : time :=1 ps; --simulation resolution
      constant realres : real :=1.0e-3; --ramener le temps en ns

--=================================================================
-- ===========      Declaration des signaux et composants ==========
```

```
63 --=================================================================
64   signal sram : typ_dpram;
65   signal MyGroup,MyWaitGroup:mpi_group;
66   signal MyWin : mpi_win;
67   signal MyRank :std_logic_vector(3 downto 0);
68   signal intercomm,array_of_errcodes : natural;
69   signal ct_state:natural :=0;
70   signal Libr : Core_io;  --regroupe tous les signaux IO de la bibliotheque
71   signal RunState : typ_mae;  --constitue la machine a etat de la tache
       materielle
72   COMPONENT hanning is
73     port (
74       ap_clk_0 : in STD_LOGIC;
75       ap_rst_0 : in STD_LOGIC;
76       ap_ctrl_0_start : in STD_LOGIC;
77       ap_ctrl_0_done : out STD_LOGIC;
78       ap_ctrl_0_ready : out STD_LOGIC;
79       ap_ctrl_0_idle : out STD_LOGIC;
80       frameOut_0_wr_data : out STD_LOGIC_VECTOR ( 7 downto 0 );
81       frameOut_0_full_n : in STD_LOGIC;
82       frameOut_0_wr_en : out STD_LOGIC;
83       frameIn_0_rd_data : in STD_LOGIC_VECTOR ( 7 downto 0 );
84       frameIn_0_empty_n : in STD_LOGIC;
85       frameIn_0_rd_en : out STD_LOGIC
86     );
87   END COMPONENT;
88
89 --=================================================================
90 -- ===========  Debut du contenu de la tache              ==========
91 --=================================================================
92
93 begin
94
95
96
97
98 --=================================================================
99 -- ===========      Instantiation de l'IP de la tache de traitemet =========
100 --=================================================================
101     utthanning: hanning port map (
102       ap_clk_0 => ap_clki,
103       ap_ctrl_0_done => ap_doneo,
104       ap_ctrl_0_idle => ap_idleo,
105       ap_ctrl_0_ready => ap_readyo,
106       ap_ctrl_0_start => ap_starti,
107       ap_rst_0 => ap_rsti,
108       frameIn_0_empty_n => frameIn_empty_ni,
109       frameIn_0_rd_data=>  frameIn_douti,
110       frameIn_0_rd_en => frameIn_reado,
111       frameOut_0_full_n => frameOut_full_ni,
112       frameOut_0_wr_data => frameOut_dino,
113       frameOut_0_wr_en => frameOut_writeo
114     );
115
116
117 --=================================================================
118 --===========  Process de gestion de la tache de traitement ========
119 --=================================================================
120  pPutGet:process(clk,reset,en)
121
122
123 --======================  Declaration  des objets a utiliser
    ===================
```

```
124
125  variable timeout,dlen,dcount : natural range 0 to 511;
126  variable adrToset,SrcAdr,DestAdr : std_logic_vector(ADRLEN-1 downto 0);
127  variable mywin : Mpi_win;
128  variable command , argv , maxprocs , info , root : natural:=0; --variable pour
       le spawn
129  variable comm : natural :=0;
130  variable adr_desti: natural :=0;
131  variable iack,spawn : std_logic:='0';
132  variable  adresse,adresse_rd :natural range 0 to 65536;
133  variable status_reg,config_reg,param :std_logic_vector(Word-1 downto 0):=(
       others=>'0');
134
135
136 --======================  Contenu du process  ===================
137  begin
138  --=== Partie combinatoire du process ===================================
139
140
141
142
143
144
145  if (clk'event and clk='1') and en='1' then
146
147 --======================  Machine a etat de gestion de la tache de traitement
       ====================
148
149    case  RunState is
150      when start =>
151
152      when Fillmem =>
153
154
155      when nextfill  =>
156
157      when InitApp =>
158
159      when GetRank =>
160
161
162      when Wincreate =>
163
164
165      when WinPost =>
166
167      when WinStart =>
168
169      when putdata =>
170
171
172      when getdata =>
173
174
175      when WinWait =>
176      when GetCmd=>
177
178      when MpiSpawn =>
179
180      when finalize =>
181      when st_timeout =>
182
183          sram.0.we<='0';
```

```
          sram.0.ena<='0';
          sram.0.enb<='0';

        RunState<=start;
      end case;
      ct_state<=ct; --pour debogage
      --ct_out<=ct;
    end if;
  end if;
  --=== Fin de la partie combinatoire du process ========================
  end process pPutGet;
--ct_out<=ct_state;
-- process ajoute pour le debug sur la carte
-- afin de signaler l'activite d'une tache demarree par MPI_Spawn
ht_state<=runstate;
MyWaitGroup<=MyGroup;
out_proc:process(clk)
variable compteur:natural;
begin
if rising_edge(clk) then
 if reset='1' then
   ct_out<=x"00";
   compteur:=0;
elsif compteur>=5000000 then
  compteur:=0;
  ct_out<=to_unsigned(ct_state,8);
elsif compteur<=2500000 then
  ct_out<=to_unsigned(ct_state+128,8);
  compteur:=compteur+1;
else
  ct_out<=to_unsigned(ct_state,8);
  compteur:=compteur+1;
end if;
end if;
end process;
end;
```

Listing A.2: *Extrait de code VHDL du fichier HT_stat.vhd pour description des tâches statiques dans MATIP*

A.3 Extrait de code VHDL du fichier *HT_dyn.vhd* pour description des tâches dynamiques dans MATIP

```
--
    ----------------------------------------------------------------------------------


-- Company:
-- Engineer:
--
-- Create Date:    12:04:21 04/22/2013
-- Design Name:
-- Module Name:    HT_process - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
```

```
18 --
19 --
    ----------------------------------------------------------------------------

20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 USE ieee.numeric_std.ALL;
23 library NocLib ;
24
25 use std.textio.all;
26 use NocLib.CoreTypes.all;
27 Library MPI_HCL;
28 use MPI_HCL.Packet_type.all;
29 use MPI_HCL.MPI_RMA.all;
30 use work.Hcl_Arch_conf.all;
31 -- Uncomment the following library declaration if using
32 -- arithmetic functions with Signed or Unsigned values
33 --use IEEE.NUMERIC_STD.ALL;
34
35 -- Uncomment the following library declaration if instantiating
36 -- any Xilinx primitives in this code.
37 --library UNISIM;
38 --use UNISIM.VComponents.all;
39
40 entity HT_dyn is
41     generic (Task_id : natural:=0);
42     Port ( clk : in  STD_LOGIC;
43            reset : in  STD_LOGIC;
44         en : in std_logic; -- active la tache
45            Interf_i : in  core_i; --signaux pour l'interface IO
46         Interf_o : out  core_o; --signaux pour l'interface IO
47            mem_o : out typ_dpram_o; -- signaux pour l'acces a la memoire
48         mem_i : in typ_dpram_i; -- signaux pour l'acces a la memoire
49         ct_out : out unsigned(7 downto 0);
50         ht_state : out typ_mae;
51        PE_in : in STD_LOGIC_VECTOR (Word-1 downto 0); --port GPIO pour le PE
52         PE_out : out STD_LOGIC_VECTOR (Word-1 downto 0) --port GPIO pour le PE
53         );
54 end HT_dyn;
55
56 architecture Behavioral of HT_dyn is
57   constant simres : time :=1 ps; --simulation resolution
58       constant realres : real :=1.0e-3; --ramener le temps en ns
59
60 --=================================================================
61 -- ===========      Declaration des signaux et composants ==========
62 --=================================================================
63   signal sram : typ_dpram;
64   signal MyGroup,MyWaitGroup:mpi_group;
65   signal MyWin : mpi_win;
66   signal MyRank :std_logic_vector(3 downto 0);
67   signal intercomm,array_of_errcodes : natural;
68   signal ct_state:natural :=0;
69   signal Libr : Core_io;  --regroupe tous les signaux IO de la bibliotheque
70   signal RunState : typ_mae;  --constitue la machine a etat de la tache
     materielle
71   COMPONENT hanning is
72     port (
73       ap_clk_0 : in STD_LOGIC;
74       ap_rst_0 : in STD_LOGIC;
75       ap_ctrl_0_start : in STD_LOGIC;
76       ap_ctrl_0_done : out STD_LOGIC;
77       ap_ctrl_0_ready : out STD_LOGIC;
```

```
78        ap_ctrl_0_idle : out STD_LOGIC;
79        frameOut_0_wr_data : out STD_LOGIC_VECTOR ( 7 downto 0 );
80        frameOut_0_full_n : in STD_LOGIC;
81        frameOut_0_wr_en : out STD_LOGIC;
82        frameIn_0_rd_data : in STD_LOGIC_VECTOR ( 7 downto 0 );
83        frameIn_0_empty_n : in STD_LOGIC;
84        frameIn_0_rd_en : out STD_LOGIC
85      );
86    END COMPONENT;
87
88  --=====================================================================
89  -- ===========  Debut du contenu de la tache              ==========
90  --=====================================================================
91
92  begin
93
94
95
96
97  --=====================================================================
98  -- ===========      Instantiation de l'IP de la tache de traitemet =========
99  --=====================================================================
100     utthanning: hanning port map (
101       ap_clk_0 => ap_clki,
102       ap_ctrl_0_done => ap_doneo,
103       ap_ctrl_0_idle => ap_idleo,
104       ap_ctrl_0_ready => ap_readyo,
105       ap_ctrl_0_start => ap_starti,
106       ap_rst_0 => ap_rsti,
107       frameIn_0_empty_n => frameIn_empty_ni,
108       frameIn_0_rd_data=>  frameIn_douti,
109       frameIn_0_rd_en => frameIn_reado,
110       frameOut_0_full_n => frameOut_full_ni,
111       frameOut_0_wr_data => frameOut_dino,
112       frameOut_0_wr_en => frameOut_writeo
113     );
114
115
116 --=====================================================================
117 --===========  Process de gestion de la tache de traitement ========
118 --=====================================================================
119  pPutGet:process(clk,reset,en)
120
121
122 --======================  Declaration  des objets a utiliser
        ===================
123
124   variable timeout,dlen,dcount : natural range 0 to 511;
125   variable adrToset,SrcAdr,DestAdr : std_logic_vector(ADRLEN-1 downto 0);
126   variable mywin : Mpi_win;
127   variable command , argv , maxprocs , info , root : natural:=0; --variable pour
        le spawn
128   variable comm : natural :=0;
129   variable adr_desti: natural :=0;
130   variable iack,spawn : std_logic:='0';
131   variable  adresse,adresse_rd :natural range 0 to 65536;
132   variable status_reg,config_reg,param :std_logic_vector(Word-1 downto 0):=(
        others=>'0');
133
134
135  --======================  Contenu du process  ===================
136   begin
137   --=== Partie combinatoire du process ==================================
```

```
138
139
140
141
142
143
144   if (clk'event and clk='1') and en='1' then
145
146  --======================  Machine a etat de gestion de la tache de traitement
       ====================
147
148     case  RunState is
149       when start =>
150
151       when Fillmem =>
152
153
154       when nextfill  =>
155
156       when InitApp =>
157
158       when GetRank =>
159
160
161       when Wincreate =>
162
163
164       when WinPost =>
165
166       when WinStart =>
167
168       when putdata =>
169
170
171       when getdata =>
172
173
174       when WinWait =>
175       when GetCmd=>
176
177       when MpiSpawn =>
178
179       when finalize =>
180       when st_timeout =>
181
182           sram.0.we<='0';
183           sram.0.ena<='0';
184           sram.0.enb<='0';
185
186         RunState<=start;
187       end case;
188       ct_state<=ct; --pour debogage
189       --ct_out<=ct;
190     end if;
191   end if;
192   --=== Fin de la partie combinatoire du process ==========================
193   end process pPutGet;
194 -- process ajoute pour le debug sur la carte
195 -- afin de signaler l'activite d'une tache demarree par MPI_Spawn
196 out_proc:process(clk)
197
198 begin
199
```

```
200 if rising_edge(clk) then
201  if Pe_in(0)='1' then
202    ct_out<=x"00";
203    compteur<=0;
204 elsif compteur>=5000000 then
205   compteur<=0;
206   ct_out<=to_unsigned(ct_state,8);
207 elsif compteur<=2500000 then
208   ct_out<=to_unsigned(ct_state+128,8);
209   compteur<=compteur+1;
210 else
211   ct_out<=to_unsigned(ct_state,8);
212   compteur<=0;
213 end if;
214 end if;
215 end process;
216 end Behavioral;
```

Listing A.3: *Extrait de code VHDL du fichier HT_ dyn.vhd pour description des tâches dynamiques dans MATIP*

A.4 Extrait du code source HLS de la fonction de Hanning

```
1  #include <cmath>
2  #include <iostream>
3  #include <ap_fixed.h>
4
5  typedef ap_fixed<32, 16, AP_RND> reel_type;
6
7  static const reel_type interpolation_depth = 0.5;
8  static const reel_type MinimumPitch = 75.0; // candidates below this frequency
       will not be recruited.
9  static const reel_type periodsPerWindow = 3.0;
10 static const reel_type fsampleRate = 8000.0;
11 static const reel_type ceiling = 500.0; // candidates above this frequency will
       be ignored.
12 static const unsigned nsamp_window = 318;
13 static const unsigned nsampFFT = 512;
14 static const reel_type NUMpi = 3.14159265358979323846264338327950288;
15
16 void hanningfix(reel_type frameIn[nsamp_window], reel_type frameOut[nsamp_window
       ])
17 {
18   #pragma HLS INTERFACE ap_fifo port = frameOut
19   #pragma HLS INTERFACE ap_fifo port = frameIn
20   #pragma HLS DATAFLOW
21   reel_type window[318] = {0.0000, 0.0001, 0.0004, 0.0009, 0.0016, 0.0024,
       0.0035, 0.0047, 0.0062, 0.0078, 0.0097,...};
22   reel_type frameInTmp[nsamp_window];
23   store_loop:
24   for (int j = 0; j < nsamp_window; j++)
25   {
26     #pragma HLS INLINE off
27     #pragma HLS PIPELINE
28     frameOut[j] = frameIn[j] * window[j];
29   }
30 }
```

Listing A.4: *Extrait du code source HLS de la fonction de Hanning*

A.5 Extrait du code VHDL d'intégration des IPs matérielles dans la TIC

```
1
2 --.....................................
3  ---  Importation des composants des IPs
4
5   COMPONENT    hanningwindow is
6 port (
7   ap_clk_0 : in STD_LOGIC;
8   ap_rst_0 : in STD_LOGIC;
9   frameIn_V_PORTA_0_clk : out STD_LOGIC;
10  frameIn_V_PORTA_0_rst : out STD_LOGIC;
11  frameIn_V_PORTA_0_en : out STD_LOGIC;
12  frameIn_V_PORTA_0_we : out STD_LOGIC_VECTOR ( 3 downto 0 );
13  frameIn_V_PORTA_0_addr : out STD_LOGIC_VECTOR ( 31 downto 0 );
14  frameIn_V_PORTA_0_din : out STD_LOGIC_VECTOR ( 31 downto 0 );
15  frameIn_V_PORTA_0_dout : in STD_LOGIC_VECTOR ( 31 downto 0 );
16  ap_ctrl_0_start : in STD_LOGIC;
17  ap_ctrl_0_done : out STD_LOGIC;
18  ap_ctrl_0_ready : out STD_LOGIC;
19  ap_ctrl_0_idle : out STD_LOGIC;
20  frameOut_V_PORTA_0_clk : out STD_LOGIC;
21  frameOut_V_PORTA_0_rst : out STD_LOGIC;
22  frameOut_V_PORTA_0_en : out STD_LOGIC;
23  frameOut_V_PORTA_0_we : out STD_LOGIC_VECTOR ( 3 downto 0 );
24  frameOut_V_PORTA_0_addr : out STD_LOGIC_VECTOR ( 31 downto 0 );
25  frameOut_V_PORTA_0_din : out STD_LOGIC_VECTOR ( 31 downto 0 );
26  frameOut_V_PORTA_0_dout : in STD_LOGIC_VECTOR ( 31 downto 0 )
27 );
28 END COMPONENT;
29
30 component fixfft is
31 port (
32   ap_clk_0 : in STD_LOGIC;
33   ap_rst_0 : in STD_LOGIC;
34   ap_ctrl_0_start : in STD_LOGIC;
35   ap_ctrl_0_done : out STD_LOGIC;
36   ap_ctrl_0_ready : out STD_LOGIC;
37   ap_ctrl_0_idle : out STD_LOGIC;
38   x_V_PORTA_0_clk : out STD_LOGIC;
39   x_V_PORTA_0_rst : out STD_LOGIC;
40   x_V_PORTA_0_en : out STD_LOGIC;
41   x_V_PORTA_0_we : out STD_LOGIC_VECTOR ( 3 downto 0 );
42   x_V_PORTA_0_addr : out STD_LOGIC_VECTOR ( 31 downto 0 );
43   x_V_PORTA_0_din : out STD_LOGIC_VECTOR ( 31 downto 0 );
44   x_V_PORTA_0_dout : in STD_LOGIC_VECTOR ( 31 downto 0 );
45   y_V_PORTA_0_clk : out STD_LOGIC;
46   y_V_PORTA_0_rst : out STD_LOGIC;
47   y_V_PORTA_0_en : out STD_LOGIC;
48   y_V_PORTA_0_we : out STD_LOGIC_VECTOR ( 3 downto 0 );
49   y_V_PORTA_0_addr : out STD_LOGIC_VECTOR ( 31 downto 0 );
50   y_V_PORTA_0_din : out STD_LOGIC_VECTOR ( 31 downto 0 );
51   y_V_PORTA_0_dout : in STD_LOGIC_VECTOR ( 31 downto 0 )
52 );
53 end component fixfft;
54
55 component ifft is
56 port (
57   ap_ctrl_0_start : in STD_LOGIC;
58   ap_ctrl_0_done : out STD_LOGIC;
59   ap_ctrl_0_ready : out STD_LOGIC;
60   ap_ctrl_0_idle : out STD_LOGIC;
61   ap_clk_0 : in STD_LOGIC;
62   ap_rst_0 : in STD_LOGIC;
63   x_V_PORTA_0_clk : out STD_LOGIC;
```

```
  x_V_PORTA_0_rst : out STD_LOGIC;
  x_V_PORTA_0_en : out STD_LOGIC;
  x_V_PORTA_0_we : out STD_LOGIC_VECTOR ( 3 downto 0 );
  x_V_PORTA_0_addr : out STD_LOGIC_VECTOR ( 31 downto 0 );
  x_V_PORTA_0_din : out STD_LOGIC_VECTOR ( 31 downto 0 );
  x_V_PORTA_0_dout : in STD_LOGIC_VECTOR ( 31 downto 0 );
  y_V_PORTA_0_clk : out STD_LOGIC;
  y_V_PORTA_0_rst : out STD_LOGIC;
  y_V_PORTA_0_en : out STD_LOGIC;
  y_V_PORTA_0_we : out STD_LOGIC_VECTOR ( 3 downto 0 );
  y_V_PORTA_0_addr : out STD_LOGIC_VECTOR ( 31 downto 0 );
  y_V_PORTA_0_din : out STD_LOGIC_VECTOR ( 31 downto 0 );
  y_V_PORTA_0_dout : in STD_LOGIC_VECTOR ( 31 downto 0 )
);
end component ifft;

component squarett is
port (
  ap_ctrl_0_start : in STD_LOGIC;
  ap_ctrl_0_done : out STD_LOGIC;
  ap_ctrl_0_idle : out STD_LOGIC;
  ap_ctrl_0_ready : out STD_LOGIC;
  ap_clk_0 : in STD_LOGIC;
  ap_rst_0 : in STD_LOGIC;
  x_V_PORTA_0_clk : out STD_LOGIC;
  x_V_PORTA_0_rst : out STD_LOGIC;
  x_V_PORTA_0_en : out STD_LOGIC;
  x_V_PORTA_0_we : out STD_LOGIC_VECTOR ( 3 downto 0 );
  x_V_PORTA_0_addr : out STD_LOGIC_VECTOR ( 31 downto 0 );
  x_V_PORTA_0_din : out STD_LOGIC_VECTOR ( 31 downto 0 );
  x_V_PORTA_0_dout : in STD_LOGIC_VECTOR ( 31 downto 0 );
  y_V_PORTA_0_clk : out STD_LOGIC;
  y_V_PORTA_0_rst : out STD_LOGIC;
  y_V_PORTA_0_en : out STD_LOGIC;
  y_V_PORTA_0_we : out STD_LOGIC_VECTOR ( 3 downto 0 );
  y_V_PORTA_0_addr : out STD_LOGIC_VECTOR ( 31 downto 0 );
  y_V_PORTA_0_din : out STD_LOGIC_VECTOR ( 31 downto 0 );
  y_V_PORTA_0_dout : in STD_LOGIC_VECTOR ( 31 downto 0 )
);
end component squarett;




.......................................

---  Instanciation des IPs


 treat1: if Task_Id=1 generate
hanningbram_i: hanningwindow port map (
  ap_clk_0 =>ap_clk_0_in,
  ap_rst_0 =>ap_rst_0_in,
  ap_ctrl_0_start =>ap_ctrl_0_start_in,
  ap_ctrl_0_done =>ap_ctrl_0_done_out,
  ap_ctrl_0_ready =>ap_ctrl_0_ready_out,
  ap_ctrl_0_idle =>ap_ctrl_0_idle_out,
  frameIn_V_PORTA_0_clk =>frameIn_V_PORTA_0_clk_out,
  frameIn_V_PORTA_0_rst =>frameOut_V_PORTA_0_rst_out,
  frameIn_V_PORTA_0_en =>frameOut_V_PORTA_0_en_out,
  frameIn_V_PORTA_0_we =>frameIn_V_PORTA_0_we_out,
  frameIn_V_PORTA_0_addr =>frameIn_V_PORTA_0_addr_out,
  frameIn_V_PORTA_0_din =>frameIn_V_PORTA_0_din_out,
```

```
127   frameIn_V_PORTA_0_dout =>frameIn_V_PORTA_0_dout_in,
128   frameOut_V_PORTA_0_clk =>frameOut_V_PORTA_0_clk_out,
129   frameOut_V_PORTA_0_rst =>frameOut_V_PORTA_0_rst_out,
130   frameOut_V_PORTA_0_en =>frameOut_V_PORTA_0_en_out,
131   frameOut_V_PORTA_0_we =>frameOut_V_PORTA_0_we_out,
132   frameOut_V_PORTA_0_addr =>frameOut_V_PORTA_0_addr_out,
133   frameOut_V_PORTA_0_din =>frameOut_V_PORTA_0_din_out,
134   frameOut_V_PORTA_0_dout =>frameOut_V_PORTA_0_dout_in
135 );
136 end generate treat1;
137
138 treat2: if Task_Id=2 generate
139 fixfft_i: fixfft port map (
140   ap_clk_0 =>ap_clk_0_infft,
141   ap_rst_0 =>ap_rst_0_infft,
142   ap_ctrl_0_start =>ap_ctrl_0_start_infft,
143   ap_ctrl_0_done =>ap_ctrl_0_done_outfft,
144   ap_ctrl_0_ready =>ap_ctrl_0_ready_outfft,
145   ap_ctrl_0_idle =>ap_ctrl_0_idle_outfft,
146   x_V_PORTA_0_clk =>x_V_PORTA_0_clk_out,
147   x_V_PORTA_0_rst =>y_V_PORTA_0_rst_out,
148   x_V_PORTA_0_en =>y_V_PORTA_0_en_out,
149   x_V_PORTA_0_we =>x_V_PORTA_0_we_out,
150   x_V_PORTA_0_addr =>x_V_PORTA_0_addr_out,
151   x_V_PORTA_0_din =>x_V_PORTA_0_din_out,
152   x_V_PORTA_0_dout =>x_V_PORTA_0_dout_in,
153   y_V_PORTA_0_clk =>y_V_PORTA_0_clk_out,
154   y_V_PORTA_0_rst =>y_V_PORTA_0_rst_out,
155   y_V_PORTA_0_en =>y_V_PORTA_0_en_out,
156   y_V_PORTA_0_we =>y_V_PORTA_0_we_out,
157   y_V_PORTA_0_addr =>y_V_PORTA_0_addr_out,
158   y_V_PORTA_0_din =>y_V_PORTA_0_din_out,
159   y_V_PORTA_0_dout =>y_V_PORTA_0_dout_in
160 );
161 end generate treat2;
162
163 treat3: if Task_Id=3 generate
164 square_id: squarett port map (
165   ap_clk_0 =>ap_clk_0_insquare,
166   ap_rst_0 =>ap_rst_0_insquare,
167   ap_ctrl_0_start =>ap_ctrl_0_start_insquare,
168   ap_ctrl_0_done =>ap_ctrl_0_done_outsquare,
169   ap_ctrl_0_ready =>ap_ctrl_0_ready_outsquare,
170   ap_ctrl_0_idle =>ap_ctrl_0_idle_outsquare,
171   x_V_PORTA_0_clk =>sx_V_PORTA_0_clk_out,
172   x_V_PORTA_0_rst =>sx_V_PORTA_0_rst_out,
173   x_V_PORTA_0_en =>sx_V_PORTA_0_en_out,
174   x_V_PORTA_0_we =>sx_V_PORTA_0_we_out,
175   x_V_PORTA_0_addr =>sx_V_PORTA_0_addr_out,
176   x_V_PORTA_0_din =>sx_V_PORTA_0_din_out,
177   x_V_PORTA_0_dout =>sx_V_PORTA_0_dout_in,
178   y_V_PORTA_0_clk =>sy_V_PORTA_0_clk_out,
179   y_V_PORTA_0_rst =>sy_V_PORTA_0_rst_out,
180   y_V_PORTA_0_en =>sy_V_PORTA_0_en_out,
181   y_V_PORTA_0_we =>sy_V_PORTA_0_we_out,
182   y_V_PORTA_0_addr =>sy_V_PORTA_0_addr_out,
183   y_V_PORTA_0_din =>sy_V_PORTA_0_din_out,
184   y_V_PORTA_0_dout =>sy_V_PORTA_0_dout_in
185 );
186 end generate treat3;
187
```

```
188 --.............. ETC ......
```

Listing A.5: *Extrait du code VHDL d'intégration des IPs matérielles dans la TIC*

A.6 Extrait de code VHDL de connexion des canaux de communication aux tâches 0 et 1 dans l'architecture MATIP

```
1
2
3  --.............. il y a du code ici ......
4
5
6
7 --============================================================
8 -- Creation des taches de communication axi-stream
9 --============================================================
10
11 -- Interface d'entree des donnees (AXI-Stream)
12   xmatip_in_interface :  axi_io_interface
13     port map (
14       M_AXIS_0_tdata   => xmatip_in_data,
15       M_AXIS_0_tid => xmatip_in_tid,
16       M_AXIS_0_tready  => xmatip_in_ready,
17       M_AXIS_0_tvalid  => xmatip_in_valid,
18       S_AXIS_0_tdata  => s_axis_tdata ,
19       S_AXIS_0_tid  => s_axis_tid,
20       S_AXIS_0_tready => s_axis_tready,
21       S_AXIS_0_tvalid =>s_axis_tvalid ,
22       s_axis_aclk_0=> clkm,
23       s_axis_aresetn_0=> rstn);
24
25  -- Interface de sortie des donnees    (AXI-Stream)
26     xmatip_out_interface :  axi_io_interface
27          port map (
28       M_AXIS_0_tdata   => m_axis_tdata,
29       M_AXIS_0_tid => m_axis_tid,
30       M_AXIS_0_tready  => m_axis_tready,
31       M_AXIS_0_tvalid  => m_axis_tvalid,
32       S_AXIS_0_tdata  => xmatip_out_data ,
33       S_AXIS_0_tid  => xmatip_out_tid,
34       S_AXIS_0_tready => xmatip_out_ready,
35       S_AXIS_0_tvalid =>xmatip_out_valid ,
36       s_axis_aclk_0=> clkm,
37       s_axis_aresetn_0=> rstn);
38
39 MATIP_Out: PE   Generic map (DestId=>0,
40                 use_dyn=>0)
41       Port Map (
42         Instruction => dmux_instruction(1),
43            Instruction_en =>dmux_instruction_en(1),
44         Core_PushOut => MPi_Node_out(1).PushOut,
45            clk =>clkm,
46            reset =>reset,
47         CE => '1',
48         ct_out=>ct_tab(1),
49         PE_Out=>Sys_out(1),
50         PE_In=>sys_In(1),
51         Core_RAM_Data_Out =>mux_ram_d(1).i.Data_out,
52         Core_RAM_Data_Out2=>mux_ram_s(1).i.Data_out,
53            Core_RAM_Data_IN => mux_ram(1).o.data_in,
54            Core_RAM_WE => mux_ram(1).o.we,
55            Core_RAM_EN => mux_ram(1).o.enb,
```

```
56            Core_RAM_Address_Wr => mux_ram(1).o.addr_wr,
57            Core_RAM_Address_Rd => mux_ram(1).o.addr_rd,
58            Core_Hold_req => mux_hold_req(1),
59            Core_Hold_Ack => dmux_hold_ack(1),
60
61             -- axi4 stream master (data output)
62             m_axis_tdata   => xmatip_out_data ,
63             m_axis_tid => xmatip_out_tid,
64             m_axis_tvalid =>xmatip_out_valid ,
65             m_axis_tready => xmatip_out_ready,
66             need_toput_read_data=> need_toput_read_tab_data(0),
67             need_toput_read_addr=> need_toput_read_tab_addr(0),
68             need_toput_read_en=>need_toput_read_tab_en(0),
69             need_toput_write_addr=> need_toput_write_tab_addr(0),
70             need_toput_write_data=>need_toput_write_tab_data(0),
71             need_toput_write_en=> need_toput_write_tab_en(0),
72
73             ready_put_read_data=> ready_put_read_tab_data(0),
74             ready_put_read_addr=> ready_put_read_tab_addr(0),
75             ready_put_read_en=>ready_put_read_tab_en(0),
76             ready_put_write_addr=> ready_put_write_tab_addr(0),
77             ready_put_write_data=>ready_put_write_tab_data(0),
78             ready_put_write_en=>ready_put_write_tab_en(0),
79
80             done_put_read_data=>done_put_read_tab_data(0),
81             done_put_read_addr=>done_put_read_tab_addr(0),
82             done_put_read_en=>done_put_read_tab_en(0),
83             done_put_write_addr=> done_put_write_tab_addr(0),
84             done_put_write_data=>done_put_write_tab_data(0),
85             done_put_write_en=> done_put_write_tab_en(0),
86
87             need_toget_read_data=> need_toget_read_tab_data(0),
88             need_toget_read_addr=> need_toget_read_tab_addr(0),
89             need_toget_read_en=>need_toget_read_tab_en(0),
90             need_toget_write_addr=> need_toget_write_tab_addr(0),
91             need_toget_write_data=>need_toget_write_tab_data(0),
92             need_toget_write_en=> need_toget_write_tab_en(0),
93
94             ready_get_read_data=> ready_get_read_tab_data(0),
95             ready_get_read_addr=> ready_get_read_tab_addr(0),
96             ready_get_read_en=>ready_get_read_tab_en(0),
97             ready_get_write_addr=> ready_get_write_tab_addr(0),
98             ready_get_write_data=>ready_get_write_tab_data(0),
99             ready_get_write_en=>ready_get_write_tab_en(0),
100
101            done_get_read_data=>done_get_read_tab_data(0),
102            done_get_read_addr=>done_get_read_tab_addr(0),
103            done_get_read_en=>done_get_read_tab_en(0),
104            done_get_write_addr=> done_get_write_tab_addr(0),
105            done_get_write_data=>done_get_write_tab_data(0),
106            done_get_write_en=> done_get_write_tab_en(0),
107
108            data_length_read_data=>data_length_read_tab_data(0),
109            data_length_read_addr=>data_length_read_tab_addr(0),
110            data_length_read_en=>data_length_read_tab_en(0),
111            data_length_write_addr=> data_length_write_tab_addr(0),
112            data_length_write_data=>data_length_write_tab_data(0),
113            data_length_write_en=> data_length_write_tab_en(0)
114
115 );
116
117
118 MATIP_In: PE  Generic map (DestId=>1,
```

```
119                 use_dyn=>0)
120         Port Map (
121           Instruction => dmux_instruction(2),
122              Instruction_en =>dmux_instruction_en(2),
123           Core_PushOut => MPi_Node_out(2).PushOut,
124              clk =>clkm,
125              reset =>reset,
126           CE => '1',
127           ct_out=>ct_tab(2),
128           PE_Out=>Sys_out(2),
129           PE_In=>sys_In(2),
130           Core_RAM_Data_Out =>mux_ram_d(2).i.Data_out,
131           Core_RAM_Data_Out2=>mux_ram_s(2).i.Data_out,
132              Core_RAM_Data_IN => mux_ram(2).o.data_in,
133              Core_RAM_WE => mux_ram(2).o.we,
134              Core_RAM_EN => mux_ram(2).o.enb,
135              Core_RAM_Address_Wr => mux_ram(2).o.addr_wr,
136              Core_RAM_Address_Rd => mux_ram(2).o.addr_rd,
137              Core_Hold_req => mux_hold_req(2),
138              Core_Hold_Ack => dmux_hold_ack(2),
139                                  -- axi4 stream slave (data input)
140               s_axis_tdata => xmatip_in_data,
141               s_axis_tid  =>xmatip_in_tid,
142               s_axis_tvalid =>xmatip_in_valid,
143               s_axis_tready =>xmatip_in_ready,
144
145               m_axis_tready =>'0' ,
146
147
148
149               need_toput_read_data=> need_toput_read_tab_data(1),
150               need_toput_read_addr=> need_toput_read_tab_addr(1),
151               need_toput_read_en=>need_toput_read_tab_en(1),
152               need_toput_write_addr=> need_toput_write_tab_addr(1),
153               need_toput_write_data=>need_toput_write_tab_data(1),
154               need_toput_write_en=> need_toput_write_tab_en(1),
155
156               ready_put_read_data=> ready_put_read_tab_data(1),
157               ready_put_read_addr=> ready_put_read_tab_addr(1),
158               ready_put_read_en=>ready_put_read_tab_en(1),
159               ready_put_write_addr=> ready_put_write_tab_addr(1),
160               ready_put_write_data=>ready_put_write_tab_data(1),
161               ready_put_write_en=>ready_put_write_tab_en(1),
162
163               done_put_read_data=>done_put_read_tab_data(1),
164               done_put_read_addr=>done_put_read_tab_addr(1),
165               done_put_read_en=>done_put_read_tab_en(1),
166               done_put_write_addr=> done_put_write_tab_addr(1),
167               done_put_write_data=>done_put_write_tab_data(1),
168               done_put_write_en=> done_put_write_tab_en(1),
169
170               need_toget_read_data=> need_toget_read_tab_data(1),
171               need_toget_read_addr=> need_toget_read_tab_addr(1),
172               need_toget_read_en=>need_toget_read_tab_en(1),
173               need_toget_write_addr=> need_toget_write_tab_addr(1),
174               need_toget_write_data=>need_toget_write_tab_data(1),
175               need_toget_write_en=> need_toget_write_tab_en(1),
176
177               ready_get_read_data=> ready_get_read_tab_data(1),
178               ready_get_read_addr=> ready_get_read_tab_addr(1),
179               ready_get_read_en=>ready_get_read_tab_en(1),
180               ready_get_write_addr=> ready_get_write_tab_addr(1),
181               ready_get_write_data=>ready_get_write_tab_data(1),
```

```
182             ready_get_write_en=>ready_get_write_tab_en(1),
183
184             done_get_read_data=>done_get_read_tab_data(1),
185             done_get_read_addr=>done_get_read_tab_addr(1),
186             done_get_read_en=>done_get_read_tab_en(1),
187             done_get_write_addr=> done_get_write_tab_addr(1),
188             done_get_write_data=>done_get_write_tab_data(1),
189             done_get_write_en=> done_get_write_tab_en(1) ,
190
191             data_length_read_data=>data_length_read_tab_data(1),
192             data_length_read_addr=>data_length_read_tab_addr(1),
193             data_length_read_en=>data_length_read_tab_en(1),
194             data_length_write_addr=> data_length_write_tab_addr(1),
195             data_length_write_data=>data_length_write_tab_data(1),
196             data_length_write_en=> data_length_write_tab_en(1)
197 );
198 --============================================================
199 S_Grp:for i in 3 to STATIC_HT generate
200 S: PE   Generic map (DestId=>i-1,
201                      use_dyn=>0)
202       Port Map (
203         Instruction => dmux_instruction(i),
204            Instruction_en =>dmux_instruction_en(i),
205         Core_PushOut => MPi_Node_out(i).PushOut,
206            clk =>clkm,
207            reset =>reset,
208         CE => '1',
209         ct_out=>ct_tab(i),
210         PE_Out=>Sys_out(i),
211         PE_In=>sys_In(i),
212         Core_RAM_Data_Out =>mux_ram_d(i).i.Data_out,
213         Core_RAM_Data_Out2=>mux_ram_s(i).i.Data_out,
214            Core_RAM_Data_IN => mux_ram(i).o.data_in,
215            Core_RAM_WE => mux_ram(i).o.we,
216            Core_RAM_EN => mux_ram(i).o.enb,
217            Core_RAM_Address_Wr => mux_ram(i).o.addr_wr,
218            Core_RAM_Address_Rd => mux_ram(i).o.addr_rd,
219            Core_Hold_req => mux_hold_req(i),
220            Core_Hold_Ack => dmux_hold_ack(i),
221
222             m_axis_tready =>'0' ,
223             need_toput_read_data=> need_toput_read_tab_data(i-1),
224             need_toput_read_addr=> need_toput_read_tab_addr(i-1),
225             need_toput_read_en=>need_toput_read_tab_en(i-1),
226             need_toput_write_addr=> need_toput_write_tab_addr(i-1),
227             need_toput_write_data=>need_toput_write_tab_data(i-1),
228             need_toput_write_en=> need_toput_write_tab_en(i-1),
229
230             ready_put_read_data=> ready_put_read_tab_data(i-1),
231             ready_put_read_addr=> ready_put_read_tab_addr(i-1),
232             ready_put_read_en=>ready_put_read_tab_en(i-1),
233             ready_put_write_addr=> ready_put_write_tab_addr(i-1),
234             ready_put_write_data=>ready_put_write_tab_data(i-1),
235             ready_put_write_en=>ready_put_write_tab_en(i-1),
236
237             done_put_read_data=>done_put_read_tab_data(i-1),
238             done_put_read_addr=>done_put_read_tab_addr(i-1),
239             done_put_read_en=>done_put_read_tab_en(i-1),
240             done_put_write_addr=> done_put_write_tab_addr(i-1),
241             done_put_write_data=>done_put_write_tab_data(i-1),
242             done_put_write_en=> done_put_write_tab_en(i-1),
243
244             need_toget_read_data=> need_toget_read_tab_data(i-1),
```

```
245             need_toget_read_addr=> need_toget_read_tab_addr(i-1),
246             need_toget_read_en=>need_toget_read_tab_en(i-1),
247             need_toget_write_addr=> need_toget_write_tab_addr(i-1),
248             need_toget_write_data=>need_toget_write_tab_data(i-1),
249             need_toget_write_en=> need_toget_write_tab_en(i-1),
250
251             ready_get_read_data=> ready_get_read_tab_data(i-1),
252             ready_get_read_addr=> ready_get_read_tab_addr(i-1),
253             ready_get_read_en=>ready_get_read_tab_en(i-1),
254             ready_get_write_addr=> ready_get_write_tab_addr(i-1),
255             ready_get_write_data=>ready_get_write_tab_data(i-1),
256             ready_get_write_en=>ready_get_write_tab_en(i-1),
257
258             done_get_read_data=>done_get_read_tab_data(i-1),
259             done_get_read_addr=>done_get_read_tab_addr(i-1),
260             done_get_read_en=>done_get_read_tab_en(i-1),
261             done_get_write_addr=> done_get_write_tab_addr(i-1),
262             done_get_write_data=>done_get_write_tab_data(i-1),
263             done_get_write_en=> done_get_write_tab_en(i-1),
264
265             data_length_read_data=>data_length_read_tab_data(i-1),
266             data_length_read_addr=>data_length_read_tab_addr(i-1),
267             data_length_read_en=>data_length_read_tab_en(i-1),
268             data_length_write_addr=> data_length_write_tab_addr(i-1),
269             data_length_write_data=>data_length_write_tab_data(i-1),
270             data_length_write_en=> data_length_write_tab_en(i-1)
271 );
272
273 end generate S_Grp;
274 dyn_mod: if dyn_allowed='1' generate
275 D_Grp:for i in STATIC_HT+1 to NOC_SIZE generate
276 D: PE   Generic map (DestId=>i-1,
277             use_dyn=>1)
278       Port Map (
279         Instruction => dmux_instruction(i),--MPi_Node_in(i).Instruction,
280            Instruction_en =>dmux_instruction_en(i),-- MPi_Node_in(i).
    Instruction_en,
281         Core_PushOut => MPi_Node_out(i).PushOut,
282            clk =>clkm,
283            reset =>reset,
284         CE => '0',
285         ct_out=>ct_tab(i),
286         PE_Out=>Sys_out(i),
287         PE_In=>sys_In(i),
288            Core_RAM_Data_Out2 =>mux_ram_s(i).i.Data_out,
289            Core_RAM_Data_Out =>mux_ram_d(i).i.Data_out,
290            Core_RAM_Data_IN => mux_ram(i).o.data_in,
291            Core_RAM_WE => mux_ram(i).o.we,
292            Core_RAM_EN => mux_ram(i).o.enb,
293            Core_RAM_Address_Wr => mux_ram(i).o.addr_wr,
294            Core_RAM_Address_Rd => mux_ram(i).o.addr_rd,
295            Core_Hold_req => mux_hold_req(i),
296            Core_Hold_Ack => dmux_hold_ack(i),
297            m_axis_tready =>'0' ,
298             need_toput_read_data=> need_toput_read_tab_data(i-1),
299             need_toput_read_addr=> need_toput_read_tab_addr(i-1),
300             need_toput_read_en=>need_toput_read_tab_en(i-1),
301             need_toput_write_addr=> need_toput_write_tab_addr(i-1),
302             need_toput_write_data=>need_toput_write_tab_data(i-1),
303             need_toput_write_en=> need_toput_write_tab_en(i-1),
304
305             ready_put_read_data=> ready_put_read_tab_data(i-1),
306             ready_put_read_addr=> ready_put_read_tab_addr(i-1),
```

```
307             ready_put_read_en=>ready_put_read_tab_en(i-1),
308             ready_put_write_addr=> ready_put_write_tab_addr(i-1),
309             ready_put_write_data=>ready_put_write_tab_data(i-1),
310             ready_put_write_en=>ready_put_write_tab_en(i-1),
311
312             done_put_read_data=>done_put_read_tab_data(i-1),
313             done_put_read_addr=>done_put_read_tab_addr(i-1),
314             done_put_read_en=>done_put_read_tab_en(i-1),
315             done_put_write_addr=> done_put_write_tab_addr(i-1),
316             done_put_write_data=>done_put_write_tab_data(i-1),
317             done_put_write_en=> done_put_write_tab_en(i-1),
318
319             need_toget_read_data=> need_toget_read_tab_data(i-1),
320             need_toget_read_addr=> need_toget_read_tab_addr(i-1),
321             need_toget_read_en=>need_toget_read_tab_en(i-1),
322             need_toget_write_addr=> need_toget_write_tab_addr(i-1),
323             need_toget_write_data=>need_toget_write_tab_data(i-1),
324             need_toget_write_en=> need_toget_write_tab_en(i-1),
325
326             ready_get_read_data=> ready_get_read_tab_data(i-1),
327             ready_get_read_addr=> ready_get_read_tab_addr(i-1),
328             ready_get_read_en=>ready_get_read_tab_en(i-1),
329             ready_get_write_addr=> ready_get_write_tab_addr(i-1),
330             ready_get_write_data=>ready_get_write_tab_data(i-1),
331             ready_get_write_en=>ready_get_write_tab_en(i-1),
332
333             done_get_read_data=>done_get_read_tab_data(i-1),
334             done_get_read_addr=>done_get_read_tab_addr(i-1),
335             done_get_read_en=>done_get_read_tab_en(i-1),
336             done_get_write_addr=> done_get_write_tab_addr(i-1),
337             done_get_write_data=>done_get_write_tab_data(i-1),
338             done_get_write_en=> done_get_write_tab_en(i-1),
339
340             data_length_read_data=>data_length_read_tab_data(i-1),
341             data_length_read_addr=>data_length_read_tab_addr(i-1),
342             data_length_read_en=>data_length_read_tab_en(i-1),
343             data_length_write_addr=> data_length_write_tab_addr(i-1),
344             data_length_write_data=>data_length_write_tab_data(i-1),
345             data_length_write_en=> data_length_write_tab_en(i-1)
346
347 );
348
349 end generate D_Grp;
350 end generate dyn_mod;
351
352
353
354  --.............. il y a du code ici ......
```

Listing A.6: *Extrait de code VHDL de connexion des canaux de communication aux tâches 0 et 1 dans l'architecture MATIP*

A.7 Extrait de la tâche de gestion des communications entrantes implémentée en VHDL

```
1
2  --.............. il y a du code ici ......
3
4
5 when Compute=>
6      case Inputstatematip is
7        when Start =>
```

```
          if (s_axis_tvalid='1')  then
           Inputstatematip<=ReadInstCode;
           ReadInstCodeState<=StartRead;
          end if;
        when ReadInstCode =>
         --============================
         -- Lecture du code instruction
         --============================
                case ReadInstCodeState is

                  when StartRead =>
                    if (s_axis_tvalid='1')  then
                     s_axis_tready<='1';
                    InstCode<=s_axis_tdata;
                    ReadInstCodeState<=Cycle2;

                     end if;
                  when Cycle2 =>
                  s_axis_tready<='0';
                  myinstruction:=to_integer(unsigned(InstCode));
                   ReadInstCodeState<=StartRead;
             --=====================================
            -- Decodage du code instruction
            --=============================================
                        ReadInstCodeState<=StartRead;
                        case myinstruction is
                          when 0 =>  comPrimitive<=MPI_Rank;
                          when 1 =>  comPrimitive<=MPI_size;
                          when 2 =>  comPrimitive<=MPI_Barrier;
                          when 3 =>  comPrimitive<=MPI_Barrier_reached;
                          when 4 =>  comPrimitive<=HCL_Ack;
                          when 5 =>  comPrimitive<=MPI_Put_Request;
                                  Inputstatematip<=ReadOriginCount;
                                  ReadOriginCountState<=StartRead;
                          when 6 =>  comPrimitive<=MPI_Get_Request;
                                  Inputstatematip<=ReadoriginRank;
                                  ReadoriginRankState<=StartRead;
                          when 7 =>  comPrimitive<=MPI_Get_Response;
                                  Inputstatematip<=ReadoriginRank;
                                  ReadoriginRankState<=StartRead;
                          when 8 =>  comPrimitive<=MPI_Init;
                          when 9 =>  comPrimitive<=MPI_Spawn;
                          when 10 =>  comPrimitive<=MPI_Finalize;
                          when 11 =>  comPrimitive<=MPI_Win_Start;
                          when 12 =>  comPrimitive<=MPI_Win_Completed;
                          when 13 =>  comPrimitive<=MPI_Win_Post;
                          when 14 =>  comPrimitive<=MPI_Win_Wait;
                          when 15 =>  comPrimitive<=MPI_Send;
                          when 16 =>  comPrimitive<=MPI_Receiv;

                         when others =>
                           Inputstatematip<=ReadInstCode;
                           ReadInstCodeState<=StartRead;
                         end case;

                       when others =>
                         Inputstatematip<=ReadInstCode;
                         ReadInstCodeState<=StartRead;
                       end case;

                  when others =>

                end case;
```

```
71
72        when ReadoriginRank =>
73    --Lecture du parametre  origin_rank
74                  case ReadoriginRankState is
75
76                     when StartRead =>
77                      if (s_axis_tvalid='1')  then
78                        s_axis_tready<='1';
79                        origin_rank<=s_axis_tdata;
80                        ReadoriginRankState<=Cycle2;
81                        end if;
82                     when Cycle2 =>
83                     origin_rank<=s_axis_tdata;
84                     s_axis_tready<='0';
85                             Inputstatematip<=ReadOriginCount;
86                             ReadOriginCountState<=StartRead;
87
88                    when others =>
89                   end case;
90            when ReadOriginCount =>
91                --Lecture du parametre  OriginCount
92                  case ReadOriginCountState is
93
94                     when StartRead =>
95                      if (s_axis_tvalid='1')  then
96                        s_axis_tready<='1';
97                        origin_count<=s_axis_tdata;
98                        ReadOriginCountState<=Cycle2;
99                        end if;
100                    when Cycle2 =>
101                    origin_count<=s_axis_tdata;
102                    s_axis_tready<='0';
103                            Inputstatematip<=ReadOriginDatatype;
104                            ReadOriginDatatypeState<=StartRead;
105
106                   when others =>
107                  end case;
108
109
110
111           when ReadOriginDatatype =>
112                  -- Lecture du parametre origin_datatype
113                case ReadOriginDatatypeState is
114
115                     when StartRead =>
116                      if (s_axis_tvalid='1')  then
117                        s_axis_tready<='1';
118                        origin_datatype<=s_axis_tdata;
119     ReadOriginDatatypeState<=Cycle2;
120                      end if;
121
122                     when Cycle2 =>
123                    s_axis_tready<='0';
124                    origin_datatype<=s_axis_tdata;
125                     Inputstatematip<=ReadTargetRank;
126                     ReadTargetRankState<=StartRead;
127
128                   when others =>
129
130                  end case;
131
132           when ReadTargetRank =>
133                  -- Lecture du parametre target_rank
```

```
134                 case ReadTargetRankState is
135                      when StartRead =>
136                       if (s_axis_tvalid='1')  then
137                         s_axis_tready<='1';
138                         target_rank<=s_axis_tdata;
139                        ReadTargetRankState<=Cycle2;
140                      end if;
141                      when Cycle2 =>
142                     s_axis_tready<='0';
143                     target_rank<=s_axis_tdata;
144                      Inputstatematip<=ReadTargetDisp;
145                      ReadTargetDispState<=StartRead;
146
147                      when others =>
148
149                    end case;
150
151             when ReadTargetDisp =>
152                 -- Lecture du parametre target_disp
153                 case ReadTargetDispState is
154
155                      when StartRead =>
156                       if (s_axis_tvalid='1')  then
157                         s_axis_tready<='1';
158                         target_disp<=s_axis_tdata;
159                         ReadTargetDispState<=Cycle2;
160                      end if;
161                      when Cycle2 =>
162                      s_axis_tready<='0';
163                      target_disp<=s_axis_tdata;
164                      Inputstatematip<=ReadTargetCount;
165                      ReadTargetCountState<=StartRead;
166
167                      when others =>
168
169                    end case;
170
171
172             when ReadTargetCount =>
173                   -- Lecture du parametre target_count
174                 case ReadTargetCountState is
175
176                      when StartRead =>
177                       if (s_axis_tvalid='1')  then
178                         s_axis_tready<='1';
179                         target_count<=s_axis_tdata;
180                         ReadTargetCountState<=Cycle2;
181                      end if;
182                      when Cycle2 =>
183                      s_axis_tready<='0';
184                      target_count<=s_axis_tdata;
185                      Inputstatematip<=ReadTargetDatatype;
186                      ReadTargetDatatypeState<=StartRead;
187
188                      when others =>
189
190                    end case;
191
192             when ReadTargetDatatype =>
193                   -- Lecture du parametre target_datatype
194                 case ReadTargetDatatypeState is
195
196                      when StartRead =>
```

```
197                      if (s_axis_tvalid='1')  then
198                        s_axis_tready<='1';
199                        target_datatype<=s_axis_tdata;
200                        ReadTargetDatatypeState<=Cycle2;
201                      end if;
202                      when Cycle2 =>
203                      target_datatype<=s_axis_tdata;
204                      s_axis_tready<='0';
205                      ----Case mpi_put
206                      if(comPrimitive=MPI_Put_Request) then
207                          Inputstatematip<=ReadData;
208                          ReadDataState<=StartRead;
209                          datacounter<=0;
210                      end if;
211                      ----Case mpi_get
212                      if(comPrimitive=MPI_Get_Request) then
213
214                          theMappedRank:=mappedRank ( target_rank,globalranks,
    destrankout);
215                          need_toget_read_addr<= std_logic_vector(to_unsigned(
    theMappedRank, 16));
216                          Inputstatematip<=Proceed;
217                          ProceedState<=StartGetRead;
218                      end if;
219
220                    when others =>
221
222                  end case;
223
224            when ReadData =>
225                -- Lecture des donnees
226               case ReadDataState is
227
228                    when StartRead =>
229                     if (s_axis_tvalid='1')  then
230                       s_axis_tready<='1';
231                       theMappedRank:=mappedRank ( target_rank,globalranks,
    destrankout);
232                       input_dataframe(datacounter)<=s_axis_tdata;
233                       ReadDataState<=Cycle2;
234                    end if;
235
236                    when Cycle2 =>
237                    s_axis_tready<='0';
238                    if (datacounter<to_integer(unsigned(origin_count))-1)  then
239                    Inputstatematip<=ReadData;
240                    ReadDataState<=StartRead;
241                    datacounter<=datacounter+1;
242                    else
243                    Inputstatematip<=Proceed;
244                     ProceedState<=StartRead;
245                    datacounter<=datacounter+1;
246                    end if;
247                    need_toput_read_en<='1';
248                   when others =>
249
250                  end case;
251
252           when Proceed =>
253          --=============================================================
254     -- Execution de la requete en fonction du code instruction de la trame
255          --
    ===================================================================
```

```
256                 case ProceedState is
257
258                      when StartRead =>
259                     --synchflag, syncflagcpy
260                           dlen:=to_integer(unsigned(origin_count));--318;
261                               destRank:=theMappedRank;
262                             adresse:=512;
263                             -- Mygroup.grp<=x"0001";-- rank 0
264                              srcAdr:=x"0200";
265                              DestAdr:=X"0200";
266                              curSrcAdr:=512;
267                              curDestAdr:=512;
268                          synchflag := need_toput_read_data;
269                            dcount:=0;
270                             HanninState <=Reseth;
271                             toto<=input_dataframe(dcount)(7 downto  0);
272                             DataState<=Soc1;
273                            RunState<=Fillmem;
274                      when Cycle1 =>
275                          theMappedRank:=mappedRank ( target_rank,globalranks,
    destrankout);
276                          need_toput_write_addr<= std_logic_vector(to_unsigned(
    theMappedRank, 16));
277                          synchflag(Task_Id):='1';
278                          need_toput_write_data <=synchflag;
279                          need_toput_write_en<='1';
280                          ProceedState<=Cycle2;
281                          ready_put_read_addr<=std_logic_vector(to_unsigned(
    theMappedRank, 16));
282                         ready_put_read_en<='1';
283                      when Cycle2 =>
284                         need_toput_write_en<='0';
285                         done_put_read_en<='1';
286                         if (ready_put_read_data(Task_Id)='1') then
287                          synchflag := need_toput_read_data;
288                          need_toput_write_addr<= std_logic_vector(to_unsigned(
    theMappedRank, 16));
289                          synchflag(Task_Id):='0';
290                          need_toput_write_data <=synchflag;
291                          need_toput_write_en<='1';
292
293
294                             dlen:=to_integer(unsigned(origin_count));--318;
295                                destRank:=theMappedRank;
296                              adresse:=512;
297                             -- Mygroup.grp<=x"0001";-- rank 0
298                              srcAdr:=x"0200";
299                              DestAdr:=X"0200";
300                              curSrcAdr:=512;
301                              curDestAdr:=512;
302                               Runstate<=WinStart;
303
304
305                         else
306                           ProceedState<=Cycle2;
307                         end if;
308                      when PutDoneRead =>
309                       need_toput_write_en<='0';
310                       done_put_read_addr<= std_logic_vector(to_unsigned(
    theMappedRank, 16));
311                        syncdoneflag:=done_put_read_data;
312                        Inputstatematip<=Proceed;
313                        ProceedState<=PutDoneWrite;
```

```
314                     when PutDoneWrite =>
315                      done_put_write_addr <= std_logic_vector(to_unsigned(
    theMappedRank, 16));
316                      syncdoneflag(Task_Id):='1';
317                       done_put_write_en <='1';
318                       done_put_write_data <=syncdoneflag;
319                     -- Runstate <=finalize;
320                       Inputstatematip <=Start;
321
322                      --type typ_mae_read is (StartRead, Cycle1, Cycle2,
    PutDoneRead, PutDoneWrite, StartGetRead,GetCycle1, GetCycle2,  GetDone);
323                  when StartGetRead =>
324                              synchflag := need_toget_read_data;
325                              need_toget_write_addr <= std_logic_vector(to_unsigned
    (theMappedRank, 16));
326                              synchflag(Task_Id):='1';
327                              need_toget_write_data <=synchflag;
328                              need_toget_write_en <='1';
329                              ready_get_read_addr <=std_logic_vector(to_unsigned(
    theMappedRank, 16));
330                              ready_get_read_en <='1';
331
332                              ProceedState <=GetCycle2;
333
334
335                     when GetCycle2 =>
336
337
338                        need_toget_write_en <='0';
339                        --done_get_read_en <='1';
340                        if (ready_get_read_data(Task_Id)='1') then
341                        synchflag := need_toget_read_data;
342                         need_toget_write_addr <= std_logic_vector(to_unsigned(
    theMappedRank, 16));
343                         synchflag(Task_Id):='0';
344                         need_toget_write_data <=synchflag;
345                         need_toget_write_en <='1';
346
347                          dlen:=to_integer(unsigned(origin_count));--318;
348                          destRank:=theMappedRank;
349                          adresse:=540;
350                          srcAdr:=x"0200";
351                          DestAdr:=x"021C";
352                          curSrcAdr:=512;
353                          curDestAdr:=540;
354                        Runstate <=WinStart;
355                        else
356                          ProceedState <=GetCycle2;
357                        end if;
358
359                  when GetDone =>
360                        need_toget_write_en <='0';
361                        done_get_read_addr <= std_logic_vector(to_unsigned(
    theMappedRank, 16));
362                        syncdoneflag:=done_get_read_data;
363                        Inputstatematip <=Proceed;
364                        ProceedState <=GetDoneWrite;
365               when GetDoneWrite =>
366                        done_get_write_addr <= std_logic_vector(to_unsigned(
    theMappedRank, 16));
367                        syncdoneflag(Task_Id):='1';
368                         done_get_write_en <='1';
369                         done_get_write_data <=syncdoneflag;
```

```
370                       ProceedState<=WriteInstCode;
371              when WriteInstCode =>
372                 input_dataframe(0)<="00000000000000000000000000000101"; --
    ecriture WriteInstCode
373                 input_dataframe(1)<=target_count; --as origin count of
    communication frame
374                 input_dataframe(2)<=target_datatype;  --as origin datatype of
    communication frame
375                 input_dataframe(3)<=origin_rank;  --as target rank of
    communication frame
376                 input_dataframe(4)<= target_disp;
377                 input_dataframe(5)<= origin_count;
378                 input_dataframe(6)<= origin_datatype;
379                 theMappedRank:=mappeddestRankOut(origin_rank,globalranks,
    destrankout);
380                 target_rank<=origin_rank;
381                 comPrimitive<=MPI_Put_Request;
382
383                 dlen:=7;--318;
384                              destRank:=theMappedRank;
385                            adresse:=512;
386                           -- Mygroup.grp<=x"0001";-- rank 0
387                            srcAdr:=x"0200";
388                            DestAdr:=X"0200";
389                            curSrcAdr:=512;
390                            curDestAdr:=512;
391                         synchflag := need_toput_read_data;
392                           dcount:=0;
393                            HanninState <=Reseth;
394                            toto<=input_dataframe(dcount)(7 downto  0);
395                            DataState<=Soc1;
396                           RunState<=Fillmem;
397                      when others =>
398
399                    end case;
400
401
402
403            when others =>
404
405          end case;
406
407
408  --.............. il y a du code ici ......
```

Listing A.7: *Extrait de la tâche de gestion des communications entrantes implémentée en VHDL*

A.8 Extrait de la tâche de gestion des communications sortantes implémentée en VHDL

```
1  listenposition:=0;
2
3   --.............. il y a du code ici ......
4
5    when Compute=>
6           --============================
7           -- Machine a etat de la tache des
8           -- communications sortantes
9           --============================
10   -- type typ_mae_xmatip_output is (Start,Listen, GetReady, Proceed, Done );
11
12           case Outputstatematip is
```

```
13                when Start =>
14                      -- initialisation
15                      need_toput_read_en<='1';
16                      need_toput_read_addr<= std_logic_vector(to_unsigned(
    Task_Id, 16));
17                      Outputstatematip<= Listen;
18
19              when Listen =>
20                      -- ecoute des demandes d'envoie des requetes
21                      -- a l'exterieur
22                   if need_toput_read_data(listenposition)='1' then
23                         ready_put_read_en<='1';
24                         ready_put_read_addr<= std_logic_vector(to_unsigned(
    Task_Id, 16));
25                         Outputstatematip<= GetReady;
26
27
28                   else
29                    if listenposition=STATIC_HT-1 then
30                    listenposition:=0;
31                    else
32                      listenposition:=listenposition+1;
33                     end if;
34
35                   end if;
36             when GetReady =>
37              --GetReady consiste a lancer l'attente
38              -- des donnees aupres de la tache qui envoie les data
39       -- Etapes: GetReady->Wincreate->Winpost->WinWait-> Proceed
40               RunState<=Wincreate;
41
42
43            when Proceed =>
44                    -- Lecture des donnees recues et ecriture sur le canal
45                    -- Ensuite on passe a la  tache suivante
46                    -- Etapes: Start->Read->Starwrite->Donewrite
47
48                       case Writestate is
49                  when Start =>
50                   -- Initialisations
51                   ready_put_write_en<='0';
52                     if listenposition=STATIC_HT-1 then
53                        listenposition:=0;
54                        else
55                          listenposition:=listenposition+1;
56                         end if;
57                   m_axis_tid<="00000000000000000000000000000000";
58                   Writestate<=Read1;
59                    readmem(ct,Libr,SRam,std_logic_vector(to_unsigned(adresse,
    ADRLEN)),param1);
60                    readaddr<=adresse;
61                   when Read1 =>
62                   --Lecture en memoire de la trame a lettre sur le canal
63                         readmem(ct,Libr,SRam,std_logic_vector(to_unsigned(
    adresse,ADRLEN)),param1);
64                         readaddr<=adresse;
65
66                    if ct=0 then
67
68                         if dcount>=dlen  then
69                          dcount:=0;
70                          Runstate<=Compute;
71                          Writestate<=Starwrite;
```

```
72                              loadcount:=0;
73
74                             else
75
76
77                         case Inputstate is
78                                   when Soc0 =>
79                                    Inputstate<=Soc1;
80                                    when Soc1 =>
81                                     param2:=param1;
82                                     wordread<=param1;
83                                     if (passed='0')then
84                                     Inputstate<=Soc1;
85                                      passed:='1';
86                                     else
87                                     Inputstate<=Soc2;
88                                      end if;
89                                    when Soc2 =>
90                                    param3:=param1;
91                                     wordread<=param1;
92                                     Inputstate<=Soc3;
93                                    when Soc3 =>
94                                      param4:=param1;
95                                       wordread<=param1;
96                                      Inputstate<=Soc4;
97                                    when Soc4 =>
98                                       param5:=param1;
99                                       wordread<=param1;
100                                     if (dcount<dlen) then
101                                     input_dataframe(dcount)(7 downto  0)<=param2;
102                                     input_dataframe(dcount)(15 downto  8)<=param3;
103                                     input_dataframe(dcount)(23 downto  16)<=param4
    ;
104                                     input_dataframe(dcount)(31 downto  24)<=param5
    ;
105                                     end if;
106                                    Inputstate<=Soc1;
107                                    dcount:=dcount+1;
108                                    end case;
109
110
111                               adresse:=adresse+1;
112                               readaddr<=adresse;
113                                         RunState<=Compute;
114                                    end if;
115                                end if;
116
117                            when Starwrite =>
118                              -- Ecriture de la trame sur le canal
119                                 if dcount>=dlen  then
120                                 --sortie lorsque toute la trame est transmise
121                                  m_axis_tvalid<='0';
122                                   RunState<=Compute;
123                                     RunState<=Compute;
124                                     Outputstatematip<= Listen;
125                                  else
126
127                                   m_axis_tvalid<='1';
128                                    m_axis_tdata<=input_dataframe(dcount);
129                                    Writestate<=Donewrite;
130                                    dcount:=dcount+1;
131
132                                   end if;
```

```
133
134                         when Donewrite =>
135                               m_axis_tvalid<='0';
136                               Writestate<=Starwrite;
137
138                     when others =>
139
140                      end case;
141
142              when others => NULL ;
143          end case;
144 --.............. il y a du code ici ......
```

Listing A.8: *Extrait de la tâche de gestion des communications sortantes implémentée en VHDL*

A.9 Package VHDL de spécification des constantes et machines à états de l'architecture XMATIP

```
1
2
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  -- Uncomment the following library declaration if using
8  -- arithmetic functions with Signed or Unsigned values
9  --use IEEE.NUMERIC_STD.ALL;
10
11 -- Uncomment the following library declaration if instantiating
12 -- any Xilinx leaf cells in this code.
13 --library UNISIM;
14 --use UNISIM.VComponents.all;
15
16 package Xmatip_Packet_type is
17 --  Port ( );
18
19
20
21     type typ_flag is array (0 to 15) of STD_LOGIC_VECTOR ( 15 downto 0 );
22     type flags is array (0 to 15) of STD_LOGIC_VECTOR ( 15 downto 0 );
23   type CommSync is record
24     need_toput  : flags;
25     ready_put : flags;
26     put_done  : flags;
27     need_toget : flags;
28     ready_get : flags;
29     get_done : flags;
30   end record CommSync;
31    Constant CODE_MPI_Rank                 : integer:=0;
32    Constant CODE_MPI_size                 : integer:=1;
33    Constant CODE_MPI_Barrier                : integer:=2;
34    Constant CODE_MPI_Barrier_reached            : integer:=3;
35    Constant CODE_HCL_Ack                  : integer:=4;
36    Constant CODE_MPI_Put_Request                    : integer:=5;
37    Constant CODE_MPI_Get_Request                  : integer:=6;
38    Constant CODE_MPI_Get_Response              : integer:=7;
39    Constant CODE_MPI_Init                 : integer:=8;
40    Constant CODE_MPI_Spawn                : integer:=9;
41    Constant CODE_MPI_Finalize                 : integer:=10;
42    Constant CODE_MPI_Win_Start              : integer:=11;
43    Constant CODE_MPI_Win_Completed            : integer:=12;
44    Constant CODE_MPI_Win_Post               : integer:=13;
```

```
45    Constant CODE_MPI_Win_Wait                : integer:=14;
46    Constant CODE_MPI_Send                  : integer:=15;
47    Constant CODE_MPI_Receiv                : integer:=16;
48
49
50          --
     ============================================================================
51        -- BEGIN XMATIP USEFUL ELEMENTS
52        --============================================================================
53 type typ_mae_xmatip_input is (Start,ReadInstCode,ReadOriginCount,
     ReadOriginDatatype,ReadoriginRank, ReadTargetRank, ReadTargetDisp,
     ReadTargetCount, ReadTargetDatatype, ReadData,Proceed );
54
55 type typ_mae_xmatip_output is (Start,Listen, GetReady, Proceed, Done );
56
57 type typ_mae_operator is (NoP,OpOR, OpAnd, OpNot, OpXor );
58
59 type typ_mae_instruction is (Unknown,MPI_Rank,MPI_size, MPI_Barrier,
     MPI_Barrier_reached, HCL_Ack,MPI_Put_Request, MPI_Get_Request,
     MPI_Get_Response, MPI_Bcast, MPI_Init, MPI_Spawn, MPI_Finalize,
     MPI_Win_Start, MPI_Win_Completed, MPI_Win_Post, MPI_Win_Wait, MPI_Send,
     MPI_Receiv );
60
61 type typ_mae_read is (StartRead, Cycle1, Cycle2,PutDoneRead, PutDoneWrite,
     StartGetRead,GetCycle1, GetCycle2,  GetDone, GetDoneWrite, ResponseGetStart,
     ResponseGetWincreate, ResponseGetDone, WriteInstCode,WriteOriginCount,
     WriteOriginDatatype,WriteoriginRank, WriteTargetRank, WriteTargetDisp,
     WriteTargetCount, WriteTargetDatatype);
62
63 type typ_mae_write is (Start, Read1, Read2, Starwrite,Donewrite);
64
65 type typ_mae_xmatip_put is (Init,WriteHeadParam,PutRequest,Proceed,
     CompleteRequest, StartgetResponse, GetResponse, CompleteGet, Done);
66
67 type type_dataframe is array (0 to 511) of STD_LOGIC_VECTOR ( 31 downto 0 );
68
69 type ranklist is array (0 to 13) of natural;
70
71   --============================================================================
72
73 end Xmatip_Packet_type;
74
75  package body Xmatip_Packet_type is
76
77
78 end Xmatip_Packet_type;
```

Listing A.9: *Package VHDL de spécification des constantes et machines à états de l'architecture XMATIP*