Interface cerveau-machine : de nouvelles perspectives grâce à l'accélération matérielle

Erwan Libessart

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Electronique*

Par

Erwan LIBESSART

Interface cerveau-machine : de nouvelles perspectives grâce à l'accélération matérielle

Thèse présentée et soutenue à Brest, le 30 novembre 2018
Unité de recherche : Lab-STICC
Thèse N° : 2018IMTA0105

Rapporteurs avant soutenance :

Bertrand Granado	Professeur, Sorbonne Université
Olivier Romain	Professeur, Université de Cergy Pontoise

Composition du Jury :

Présidente :	Fan Yang	Professeur, Université de Bourgogne
Examinateurs :	Bertrand Granado	Professeur, Sorbonne Université
	Olivier Romain	Professeur, Université de Cergy Pontoise
	Matthieu Arzel	Maître de Conférences, IMT Atlantique
	Cyril Lahuec	Maître de Conférences (HDR), IMT Atlantique
Dir. de thèse :	Francesco Andriulli	Professeur, Politecnico di Torino

Remerciements

Ce manuscrit est le résultat de trois années de travail. Je tiens à remercier les personnes qui ont participé à ces travaux, de près comme de loin. Tout d'abord, un grand merci aux membres de mon jury de thèse : Madame Fan Yang ainsi que Messieurs Bertrand Granado, Olivier Romain, Francesco Andriulli, Matthieu Arzel et Cyril Lahuec. C'était un réel plaisir d'échanger avec vous lors de cette journée.

Merci également à l'ensemble du personnel du département Electronique de Télécom Bretagne, devenu IMT Atlantique entretemps. Ces trois années parmi vous ont été très enrichissantes pour moi, que ce soit du point de vue personnel ou professsionnel.

Je tiens à remercier ma famille pour son soutien sans faille tout au long de ses années. Je sais que cela n'a pas toujours été simple, mais maintenant, vous êtes rodés pour la suite !

Ensuite, un grand merci à tous les thésards du département pour m'avoir soutenu et supporté. Une pensée particulière pour Franck, André, Benoît, Valentin, les deux Paul, Pierre, ainsi que Marie-Josépha et Benoît qui ont eu la rude tâche de partager le bureau avec moi durant la fin de ma thèse.

Enfin, je tiens à remercier chaleureusement mon équipe encadrante composée de Matthieu Arzel et Cyril Lahuec comme encadrants et de Francesco Andriuilli comme directeur. Merci pour votre confiance et vos conseils, qui m'ont permis d'arriver là où je suis aujourd'hui. C'était un réel plaisir de travailler avec vous et j'espère sincèrement que l'occasion se représentera dans le futur.

Résumé

Les interfaces cerveau-machine (ICM) permettent de contrôler un appareil électronique grâce aux signaux cérébraux. Plusieurs méthodes de mesure de ces signaux, invasives ou non, peuvent être utilisées. L'électro-encéphalographie (EEG) est la méthode non-invasive la plus étudiée car elle propose une bonne résolution temporelle et le matériel nécessaire est bien moins volumineux que les systèmes de mesure des champs magnétiques. L'EEG a cependant une faible résolution spatiale, ce qui limite les performances des ICM utilisant cette méthode de mesure. Ce souci de résolution spatiale peut être réglé en utilisant le problème inverse de l'EEG, qui permet de passer des potentiels mesurés en surface à une distribution volumique des sources de courant dans le cerveau.

Le principal verrou de cette technique est le temps nécessaire (plusieurs heures) pour calculer avec une station de travail la matrice permettant de résoudre le problème inverse. Dans le cadre de cette thèse, nous avons étudié les solutions actuelles pour accélérer matériellement la conception de cette matrice. Nous avons ainsi proposé, conçu et testé une architecture électronique dédiée à ces traitements pour ICM. Les premiers résultats démontrent que notre solution permet de passer de plusieurs heures de calcul sur une station de travail à quelques minutes sur circuit reconfigurable. Cette accélération des traitements d'imagerie par EEG facilitera grandement la recherche sur l'utilisation du problème inverse et ouvrira ainsi de nouvelles perspectives pour le domaine de l'ICM.

Abstract

Brain-Computer Interfaces (BCI) are systems that use brain activity to control an external device. Various techniques can be used to collect the neural signals. The measurement can be invasive or non-invasive. Electroencephalography (EEG) is the most studied non-invasive method. Indeed, EEG offers a fine temporal resolution and ease of use but its spatial resolution limits the performances of BCI based on EEG. The spatial resolution of EEG can be improved by solving the EEG inverse problem, which allows to determine the distribution of electrical sources in the brain from EEG.

Currently, the main difficulty is the time needed (several hours) to compute the matrix which is used to solve the EEG inverse problem. This document describes the proposed solution to provide a hardware acceleration of the matrix computation. A dedicated electronic architecture has been implemented and tested. First results show that the proposed architecture divides the calculation time by a factor of 60 on a programmable circuit. This acceleration opens up new perspectives for EEG BCI.

Table des matières

Table des figures

Liste des tableaux

Introduction

La notion d'interface homme-machine, qui englobe tous les moyens et techniques employés par l'homme pour communiquer avec une machine, est omniprésente dans le monde actuel. Si le clavier et la souris d'ordinateur sont maintenant utilisés quasi quotidiennement par une partie de la population, de nombreuses études cherchent toujours à en améliorer l'ergonomie. Cette amélioration peut passer par l'utilisation de nouvelles interfaces telles que la voix, la vision ou encore le positionnement de l'utilisateur dans le cas de la réalité virtuelle [12, 13, 14, 15]. Cependant, il faut également pouvoir répondre aux cas où toutes ces interfaces ne sont pas utilisables. Il est donc intéressant d'apporter un autre moyen de communication pour des utilisateurs ne pouvant pas générer les mouvements musculaires nécessaires aux interfaces homme-machine usuelles. C'est à ce défi que répond le domaine de l'ICM.

L'ICM, aussi appelée interface cerveau-ordinateur ou interface neuronale directe, désigne un moyen de communication utilisant l'activité cérébrale. Il est important de noter que cela s'effectue de manière totalement indépendante des périphériques usuels du cerveau : les nerfs périphériques et les muscles. Le but de l'ICM est donc de fournir au cerveau un autre intermédiaire de communication, sous contrôle de l'utilisateur et non pas "d'écouter" son activité cérébrale à son insu. Il existe plusieurs domaines d'application. Le premier qui vient à l'esprit est forcément le domaine médical, où les ICM peuvent permettre à des personnes handicapées de contrôler un fauteuil, une prothèse ou encore un lit hospitalier [16]. Mais cette technologie n'est pas forcément destinée qu'aux personnes en situation de handicap. En effet, plusieurs applications peuvent être envisagées notamment dans le divertissement, l'authentification ou encore le neuromarketing [17]. Le monde de l'ICM semble donc avoir, du point de vue applicatif, le potentiel d'intéresser le grand public. Pourtant, son utilisation est actuellement loin d'être répandue dans la vie quotidienne de chacun. Existe-t-il des verrous qui empêchent l'essor de cette technologie ?

L'idée de pouvoir contrôler un dispositif avec les signaux générés par l'activité du cerveau est née en 1929, suite aux travaux de Berger sur l'EEG [18]. La notion d'ICM est apparue dans les années 1970 avec les travaux de Vidal [19, 20]. Ces recherches étaient effectuées dans le cadre d'un contrat avec l'Agence pour les projets de recherche avancée de défense (Defense Advanced Research Projects Agency, DARPA), également impliquée dans le développement des premières versions d'Internet. Ces travaux ont permis de prouver qu'il était possible d'utiliser les signaux venant de l'activité cérébrale pour transmettre la volonté de l'utilisateur de manière efficace et ont initié des décennies de recherche.

Pour qu'une ICM soit performante, il faut que la méthode de mesure de ces signaux ait une bonne résolution spatiale et temporelle. Est-il cepandant possible d'assurer cette

résolution tout en ayant une solution à la fois portable et peu coûteuse ? Suite aux travaux de Berger, 2 voies distinctes pour l'utilisation de l'ICM se sont développées : celle utilisant des méthodes d'acquisition invasives et l'autre utilisant des méthodes non-invasives.

Une ICM invasive nécessite une opération neurochirurgicale. Le principe consiste en effet à placer des électrodes directement sur la matière grise pour avoir le signal le moins bruité possible. Cette méthode a la particularité de permettre une ICM dans le sens cerveau vers machine mais aussi de machine vers cerveau. En effet, les électrodes peuvent par exemple être utilisées pour transmettre des signaux d'une caméra vers le cortex visuel, afin d'apporter une vision artificielle à des personnes aveugles [21]. Concernant le sens cerveau vers machine, l'usage d'une méthode d'acquisition invasive permet d'atteindre une haute résolution spatio-temporelle, ce qui permet de proposer une réponse plus rapide et plus précise [22]. En contrepartie, il y a certains problèmes qu'il faut considérer. Outre le fait que chaque application requiert un emplacement d'électrodes spécifique, il est difficile de garantir la stabilité du signal sur des périodes de fonctionnement de l'ordre de mois ou années. Ceci explique que beaucoup d'expérimentations sont effectuées sur des animaux de laboratoire, notamment des singes et des rats. L'utilisation de méthodes invasives n'est donc actuellement pas une solution viable pour proposer des applications à base d'ICM au grand public.

Les méthodes non-invasives se distinguent surtout des méthodes invasives par le fait qu'elles comportent beaucoup moins de risques pour l'utilisateur. En effet, celles-ci utilisent l'imagerie cérébrale, ce qui ne demande pas d'opération chirurgicale. Cela permet d'envisager un seul système de mesure de l'activité cérébrale pour n'importe quelle application puisque l'ensemble de l'activité cérébrale peut être prise en compte. Elles sont également moins contraignantes à porter pour l'utilisateur. Néanmoins, elles souffrent d'un manque de résolution spatiale puisque le signal est bruité par le passage au travers de la boîte crânienne. Plusieurs moyens de mesures peuvent être utilisés comme l'Imagerie par Résonance Magnétique (IRM), l'Imagerie par Résonance Magnétique fonctionnelle (IRMf), la Magnéto-Encéphalographie (MEG) ou encore l'EEG. Si les méthodes basées sur l'étude des champs magnétiques proposent une très bonne résolution spatiale elles nécessitent cependant un équipement imposant. Par exemple, un appareil d'IRM peut peser plusieurs tonnes et coûter plus d'un million de dollars pour les versions basiques [23]. L'EEG est la méthode de mesure la plus étudiée pour l'ICM. En effet, cette méthode, utilisant un casque sur lequel sont placés des électrodes, est très simple à porter pour l'utilisateur, en plus d'avoir une très bonne résolution temporelle en comparaison avec les autres méthodes non-invasives. Cela permet d'envisager des applications fonctionnant en temps réel. Cette dernière méthode semble donc être la plus adaptée pour amener l'ICM dans le quotidien de tous.

Il existe plusieurs façons relativement simples pour exploiter les signaux EEG afin de proposer des applications utilisant une ICM, les méthodes des Potentiels Évoqués Visuels de Régime Permanent (PEVRP) ou celles des Désynchronisations et Synchronisations Liées à l'Événement (DSLE) peuvent être citées. Néanmoins, ces dernières offrent souvent des possibilités limitées, car l'EEG est une méthode de mesure souffrant d'une faible résolution spatiale. Une façon de résoudre ce souci de résolution est d'utiliser le problème inverse de l'EEG qui permet la distribution volumique des

sources électriques dans le cerveau à partir des signaux mesurés sur le cuir chevelu. Cette méthode est déjà utilisée dans des applications d'imagerie cérébrale, comme par exemple l'étude de l'épilepsie. Nous avons donc ici une piste intéressante pour atteindre notre but d'étendre au grand public l'utilisation de l'ICM. Cependant, la résolution du problème inverse de l'EEG demande de calculer en amont la matrice représentant le modèle de propagation entre les sources électriques et les électrodes de mesure. Actuellement, plusieurs heures de calcul sont nécessaires pour concevoir cette matrice à partir de la modélisation du cerveau de l'utilisateur. Ce temps de calcul ralentit donc fortement les expérimentations sur les ICM utilisant le problème inverse. Le défi ici consiste alors à réduire suffisament ce temps de calcul pour fluidifier l'exploration de cette solution novatrice. Le projet *Seizing Advances in BCI from high Resolution EEG imaging in runtime* (SABRE) cherche justement à répondre à ce verrou en jouant sur 2 leviers. Le premier est algorithmique et vise à réduire la complexité des calculs nécessaires à l'augmentation de la résolution spatiale. Le second est matériel. Le but est de développer un accélérateur matériel de la nouvelle version de l'algorithme, ce qui permet de réduire les temps de calculs par rapport à un processeur à usage universel. Plusieurs choix de cibles matérielles sont possibles, comme le processeur graphique, le FPGA ou encore l'ASIC. Laquelle d'entre elles est la plus adaptée pour permettre la diffusion de l'ICM au plus grand nombre ? Il faut ainsi définir des priorités dans les contraintes de coût, portabilité, consommation et performances.

Dans ce document, nous présentons les travaux effectués sur la partie accélération matérielle du projet SABRE. Notre objectif est d'analyser la partie la plus critique de l'algorithme afin de l'implanter sur cible matérielle. Ainsi nous pourrons réduire au mieux le temps de calcul afin de permettre l'exploration de nouvelles perspectives dans le domaine des ICM.

Dans le Chapitre I, plusieurs méthodes d'exploitation des signaux EEG sont présentées. Celles-ci sont utilisées pour des applications ICM simples et peuvent également utiliser une accélération matérielle. Les travaux algorithmiques du projet SABRE sur le problème inverse de l'EEG sont également présentés.

La partie d'algorithme considérée comme critique à la fin du Chapitre I est analysée dans le cadre du Chapitre II. Sa capacité à être accélérée est également discutée, en tenant compte de sa structure et des sous fonctions qui la composent. Différentes cibles possibles pour l'accélération matérielle sont également comparées afin de déterminer l'approche la plus profitable.

Une fois la plate-forme d'accélération choisie, le Chapitre III détaille plusieurs méthodes de mise en œuvre d'accélérateurs matériels. Cette analyse concerne 2 axes différents : les moyens de création d'opérateurs et les différentes possibilités d'intégration au sein d'un système. Cette analyse justifie le choix des travaux qui sont présentés par la suite.

Le Chapitre IV présente les travaux effectués sur 2 opérateurs non-linéaires : l'inverse et l'inverse de la racine carrée. Cette dernière est une opération clé en électromagnétisme car elle est utilisée pour calculer la valeur d'un potentiel électrostatique [24]. Le travail effectué pour cet opérateur est donc non seulement utile pour le sujet de l'ICM traité dans ce document mais aussi pour tout algorithme basé sur l'électrostatique. Cela explique une volonté de développer des opérateurs totalement génériques qui sont proposés en projet libre.

L'implantation de l'algorithme détaillé dans le Chapitre II est le sujet du Chapitre V. Les solutions retenues pour les autres opérateurs nécessaires sont tout d'abord présentées. Un retour d'expérience sur le développement d'architecture à haut débit avec des contraintes de tailles de données est également fourni. Enfin, les résultats d'accélération sur plate-forme FPGA et ASIC sont présentés et mis en regard avec la volonté d'accélérer la résolution du problème directe de l'EEG pour ainsi faciliter l'utilisation du problème inverse dans le domaine des ICM.

En conclusion, le résumé des contributions est dressé et des perspectives de travaux sont proposées. Ces perspectives permettent de se rapprocher encore plus de l'objectif de temps réel pour une ICM à base de signaux EEG, ouvrant un nouveau marché pour plusieurs domaines d'application.

Chapitre 1

Interface-cerveau machine par électro-encéphalographie

1.1 Introduction

Dans ce chapitre, le principe de l'EEG ainsi que son utilisation actuelle dans le cadre de l'ICM sont présentés. Dans un premier temps, les méthodes d'exploitation directe des signaux EEG sont abordées, tout en discutant de leurs points forts et points faibles, dans l'optique d'une expansion de l'ICM au grand public. Par la suite, nous présentons la thématique du problème inverse qui permet de passer d'une mesure surfacique à un modèle volumique, et ainsi augmenter la résolution spatiale de EEG. Cela permettra ainsi de situer et présenter le projet dans lequel s'inscrivent les travaux présentés dans ce document.

1.2 Principe de l'EEG et exemples d'ICM par exploitation directe

1.2.1 Principe de l'EEG

L'activité du cerveau génère des échanges électro-chimiques au niveau des neurones. Ces échanges créent des champs électromagnétiques qui peuvent donc être considérés comme des images de l'activité cérébrale. Le principe d'une ICM est de mesurer ces signaux afin d'en déduire une volonté du patient à transmettre à une machine. Ce domaine est actuellement en nette expansion. Des équipes de recherche proposent ainsi des plateformes dédiées à la conception et aux tests d'ICM [25]. Contrairement aux méthodes invasives, qui demandent d'implanter des électrodes intra-crâniennes, l'utilisation d'une ICM non invasive permet d'assurer l'intégrité physique de l'utilisateur, ce qui est le critère essentiel pour permettre le développement commercial de l'ICM à une grande échelle, voire pour un usage commun du grand public. Les domaines d'applications sont nombreux. L'aspect paramédical est évident, mais il y a aussi des possibilités d'utilisation d'ICM dans les domaines militaire ou vidéoludique. Plusieurs méthodes de mesures non invasives peuvent être utilisées comme l'IRM, l'IRMf, la MEG ou encore l'EEG. Les 3 premières méthodes reposent sur la mesure

(a)

(b)

FIGURE 1.1 – Exemples des appareils nécessaires pour de la mesure des champs magnétiques : (a) machine d'IRM [1], (b) machine de MEG [2].

des champs magnétiques qui nécessite un équipement imposant et coûteux. En effet, les appareils d'IRM, dont un exemple est visible en Figure 1.1a, pèsent plusieurs dizaines de tonnes et coûtent plus de 850 000 euros [23]. Cela s'explique notamment par le fait qu'ils ont besoin d'un puissant aimant afin de générer un champ magnétique stable. La version la plus portable connue actuellement n'est transportable qu'en semi-remorque [26]. Toute tentative de miniaturisation passe toutefois par la diminution de la puissance de l'aimant de l'appareil, ce qui diminue la résolution spatiale de la mesure. Concernant la MEG, l'appareil de mesure nécessaire est également relativement imposant, comme nous pouvons le voir en Figure 1.1b.

De plus, les champs magnétiques mesurés étant très faibles (de l'ordre du femtotesla dans le cas de la MEG), un blindage magnétique peut s'avérer nécessaire autour de l'appareil de mesure afin d'éviter les risques d'interférences avec le monde extérieur. Ces méthodes basées sur la mesure des champs magnétiques sont donc trop encombrantes, et la miniaturisation n'est pas possible dans l'immédiat. Par conséquent, elles ne répondent pas au critère de confort de l'utilisateur de l' ICM. Ces points expliquent en partie pourquoi l'EEG est la méthode d'acquisition la plus étudiée actuel-

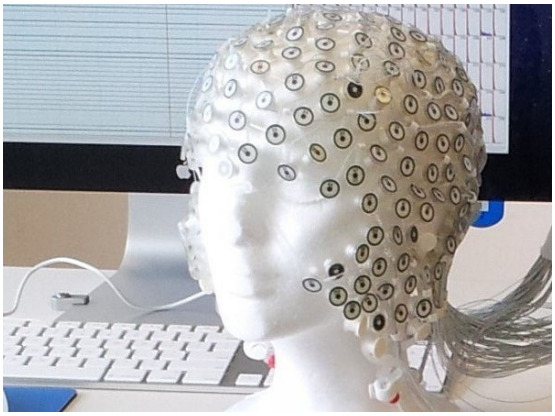


FIGURE 1.2 – Exemple d'un casque d'EEG [3].

lement.

L'équipement de mesure pour l'EEG consiste en un casque sur lequel sont placés plusieurs électrodes. Un exemple de casque est visible en Figure 1.2. Cette méthode de mesure a également une très bonne résolution temporelle en plus d'une bonne portabilité. Les électrodes d'un casque d'EEG sont placées selon le système 10-20. Il s'agit d'une méthode standard internationale qui décrit l'emplacement des électrodes en fonction du nombre de ces derniers [27].

Le prix des casques EEG varie en fonction du nombres d'électrodes. Cela peut varier de 90 euros pour un casque à une seule électrode jusqu'à plus de 20 000 euros pour un casque avec 256 électrodes, ce qui est représente la capacité de mesure la plus élevée actuellement [28]. Dans le cas d'une ICM, les signaux mesurés par les électrodes doivent être amplifiés avant d'être analysés et ainsi en déduire une commande à transmettre à une machine.

Les potentiels mesurés par EEG sont les images des potentiels postsynaptiques excitateurs ou inhibiteurs des neurones. Ces potentiels ont lieu respectivement à l'entrée d'un flux d'ions positifs et négatifs dans la cellule. Ces événements à l'échelle du neurone génèrent un potentiel très faible qui ne contribue pas de manière significative aux potentiels mesurés sur le cuir chevelu. De ce fait, les potentiels enregistrés par EEG représentent la somme des potentiels générés par des milliers voire des millions de neurones au même instant. L'amplitude des signaux mesurés sur le cuir chevelu par EEG, pouvant atteindre 300 μV, est donc l'image directe du nombre de neurones synchrones à un instant donné. Un exemple de signaux relevés par ces électrodes est visible en Figure 1.3. Une mesure EEG permet de visualiser différentes oscillations électromagnétiques, que l'on nomme rythmes cérébraux. Chaque rythme possède sa propre bande de fréquences et correspond à une fonction cérébrale spécifique. Les ondes EEG les plus intéressantes sont les suivantes [29] :

- l'onde *delta*, 0,4 à 4 Hz, caractérise le sommeil profond ;
- l'onde *theta*, 4 à 8 Hz, caractérise la somnolence ;
- l'onde *alpha*, 8 à 13 Hz, caractérise un état de conscience apaisée, contient la sous-onde *mu*, entre 10 et 12 Hz ;
- l'onde *beta*, 13 à 30 Hz, caractérise une période de concentration.

L'étude de ces ondes dans des situations spécifiques peut permettre d'en déduire de possibles commandes pour une ICM.

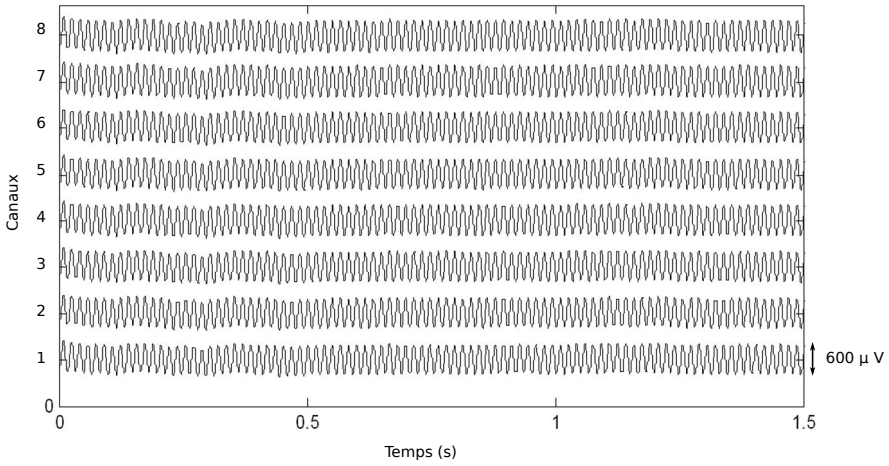


FIGURE 1.3 – Exemple de mesures EEG sur 8 canaux [4].

Dans la suite de cette section, plusieurs méthodes d'analyse des signaux EEG sont présentées.

1.2.2 Paradigmes d'ICM non invasive par exploitation directe des signaux EEG

1.2.2.1 Potentiels liés à l'événement

Une grande partie des paradigmes utilisés par les ICM proposées actuellement se repose sur l'analyse des Potentiels Liés à l'Événement (PLE). Il s'agit de réponses du cerveau qui apparaissent à un moment fixe après un événement externe, comme un stimulus sensoriel, ou un événement interne, comme le fait de penser à une action précise. On distingue ainsi 2 types de potentiels :

— le potentiel exogène est la conséquence d'un événement externe. Ce potentiel est une réponse physiologique au stimulus et ne dépend donc d'aucun contexte ;
— le potentiel endogène est la conséquence d'un événement interne. Ce potentiel peut être en plus dépendant du contexte. Par exemple, si un utilisateur tente de repérer la lettre *D* dans un mot, il va générer un potentiel si cette lettre lui est présentée, mais rien n'est mesurable avec cette même lettre si l'utilisateur cherche la lettre *T*.

L'utilisation de ces potentiels liés à l'événement est donc une méthode relativement simple pour mettre en place des ICM. La nature du potentiel détermine l'importance de la phase d'entrainement lors de la mise en place d'une ICM. Si un potentiel exogène est utilisé, presque aucun entrainement n'est requis, la réponse étant physiologique. A l'inverse, si un potentiel endogène est mis en œuvre, une phase d'entrainement via un retour de l'ICM est nécessaire. Par la suite, plusieurs méthodes d'utilisation de ces potentiels sont détaillées.

1.2.2.2 Potentiels évoqués visuels de régime permanent

Le paradigme des PEVRP utilise un stimulus visuel, le plus souvent un élément graphique clignotant. Cette méthode repose sur le fait que l'on observe sur les mesures EEG un pic d'amplitude à la même fréquence que celle de l'élément clignotant au centre du champ visuel. Les réponses sont plus facilement observables pour des fréquences de stimulation entre 5 et 27 Hz [30]. Il est donc possible de présenter différents stimuli à l'utilisateur sur un même écran. Lorsque l'utilisateur oriente son regard sur un stimulus, un signal EEG caractéristique est alors mesuré et détecté pour déclencher la commande souhaitée (Figure 1.4). Les stimuli doivent avoir une fréquence et/ou une phase différente afin de pouvoir les différencier par la suite [31, 16]. Il faut noter que ces stimuli doivent être suffisamment espacés afin d'assurer de n'en avoir qu'un seul au centre du champ de vision et ainsi éviter les interférences. L'interface décrite dans [31] concerne un système multimédia à 4 commandes possibles, présentées sur un seul écran. Ces commandes sont différenciées par des phases différentes.

Quand un système propose plus de 4 commandes, il est préférable d'utiliser des fréquences différentes ou un mélange des 2 techniques, soit l'utilisation de fréquences et de phases différentes. En effet, il y a trop de risques d'interférences avec des différences de phases inférieures à 90 degrés [31, 16]. C'est d'ailleurs la solution qui consiste à utiliser à la fois phases et fréquences différentes qui a été choisie pour une interface contrôlant un lit d'hôpital [16]. Il faut alors éviter au maximum de mettre 2 stimuli de même fréquence ou de même phase côte à côte afin de limiter au mieux le risque d'erreur.

Cette ICM de lit hospitalier comporte 8 commandes différentes, ce qui nécessite autant de stimuli visuels, distribués sur 2 fréquences et 4 phases. Les signaux EEG sont mesurés par un casque avec 3 électrodes puis la détection de fréquence et de phase s'effectue après amplification. Cette détection se fait à l'aide de filtres passe-bande pour les fréquences et une méthode des moindres carrés pour la phase. Une phase de test a été réalisée avec 15 sujets qui devaient observer une séquence de 8 commandes sur la disposition de stimuli. Les résultats montrent une précision moyenne de choix de commandes de 92,5% avec un temps moyen entre 2 commandes de 5,22 secondes.

Les travaux d'une équipe taïwanaise utilisent aussi la méthode des PEVRP mais en utilisant une fréquence par commande [32]. En effet, les 6 commandes utilisées pour le contrôle d'une interface de téléphone portable sont associées à des stimuli se partageant la bande de fréquence entre 9 et 14 Hz. L'analyse des signaux mesurés par le casque EEG à 4 électrodes se fait par analyse de la corrélation entre les signaux mesurés et des groupes d'échantillons pré-enregistrés lors d'une phase d'apprentissage. La phase de test sur 4 sujets sur une séquence de 6 commandes a donné une précision moyenne de 89%.

La méthode des PEVRP est donc une bonne solution pour réaliser une ICM de manière relativement simple et à coût raisonnable. En effet, il suffit pour l'application d'ICM de gérer l'affichage des stimuli sur l'écran observé par l'utilisateur. Les signaux EEG sont récupérés pour déterminer la fréquence et la phase du stimulus fixé, et donc la commande souhaitée, Figure 1.4. Le coût final de la solution s'élève à 110 euros [31].

Ce paradigme a toutefois ses limites. Le fait de devoir présenter des stimuli visuels pour chaque commande limite les possibilités pour une même taille d'écran. De plus, la nécessité de devoir fixer des éléments clignotants peut déclencher des crises d'épilepsie

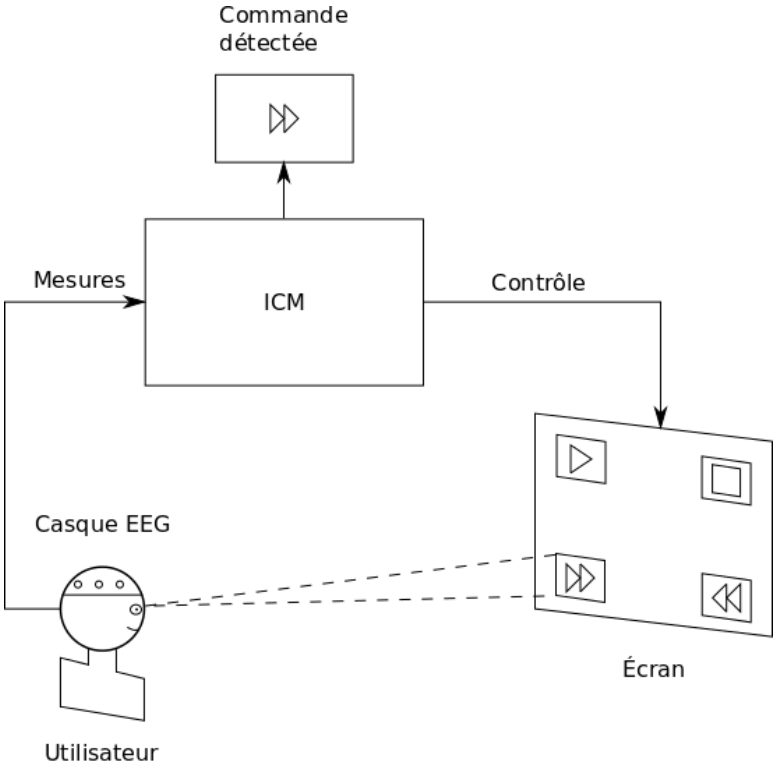


FIGURE 1.4 – Représentation d'une ICM utilisant le paradigme PEVRP.

et certaines fréquences de stimulation peuvent engendrer de la fatigue [33] car cela demande beaucoup de concentration. Il est donc difficile d'imaginer un usage fréquent et prolongé d'une ICM basée sur la méthode PEVRP.

En conclusion, la méthode des PEVRP est donc une méthode relativement simple et peu coûteuse pour mettre en place une ICM. Toutefois, elle demande à l'utilisateur de fixer un écran avec des éléments clignotants, ce qui n'est pas forcément recommandé pour de longues sessions d'utilisation. De plus, sa résolution en nombre de commandes machine est limitée d'une part par la taille de l'écran, et d'autre part par le nombre limité des couples fréquences/phases disponibles pour les stimuli. Cette méthode reste donc trop contraignante et limitée dans l'optique d'une ICM grand public.

1.2.2.3 Désynchronisations et synchronisations liées à l'événement

Le paradigme des DSLE est un autre cas particulier des PLE. Cela correspond à 2 phénomènes visibles dans le spectre de puissance des rythmes cérébraux qui peuvent être déclenchés par des événements internes ou externes. La Désynchronisation Liée à l'Événement (DLE) est une diminution de puissance de l'EEG dans une bande de fréquence donnée, le tout en relation avec un événement particulier. La DLE la plus connue est celle provoquée par l'ouverture des yeux et qui est visible dans la bande de fréquence de l'onde *alpha* [34]. Ces désynchronisations sont en effet très répandues dans la bande *alpha*, notamment lors de tâches impliquant la mémoire ou tout ce qui est lié à la perception [18]. A l'inverse, la Synchronisation Liée à l'Événement (SLE) est une augmentation de puissance de l'EEG dans une bande de fréquence donnée.

Le fait que les potentiels évoqués apparaissent à un moment fixe par rapport à l'événement est particulièrement visible pour la DSLE, comme le montrent les travaux de Pfurtscheller [35]. Dans cette étude, l'EEG d'un volontaire lors d'une tâche motrice imaginée ou réellement effectuée a révélé une DLE dans le rythme *mu* démarrant 2,5 secondes avant le début de l'événement étudié. De la même façon, le rythme *beta* se distingue par une DLE juste avant le début de l'événement et est directement suivie par une SLE juste après le début de l'événement déclencheur. Si la réponse globale des signaux EEG à un même événement est la même pour chaque utilisateur, les caractéristiques précises peuvent varier, comme l'amplitude des désynchronisations et des synchronisations ou encore les fréquences où elles sont le plus visibles ou l'instant où elles commencent. Les ICM se basant sur le paradigme des DSLE ont donc besoin d'une phase d'apprentissage afin de mémoriser la réponse cérébrale de l'utilisateur à l'événement étudié. Puis, les signaux EEG sont analysés durant l'utilisation de l'ICM afin de retrouver cette réponse et ainsi déterminer une commande à envoyer.

L'Imagerie Motrice (IM) est le nom donné au cas particulier où l'événement étudié est l'imagination d'un mouvement. C'est notamment la méthode utilisée pour la mise en œuvre de la plateforme *Embedded EEG-based BCI System* (3EGBCI) [36]. Ce prototype vise à détecter l'intention de bouger une des deux mains afin de contrôler des appareils domestiques. Le fonctionnement de l'ICM peut se résumer comme suit : un mouvement imaginé de la main droite permet de faire défiler les choix d'un menu alors qu'un mouvement de la main gauche sert à valider le choix et ainsi accéder au menu associé. Après avoir isolé les bandes de fréquences à étudier grâce à des filtres spécifiques, l'analyse effectuée utilise la technique nommée *Common Spatial Pattern* (CSP). Il s'agit d'une méthode de séparation aveugle de source qui permet d'extraire efficacement des DSLE [37]. Les résultats communiqués après des tests sur 12 sujets montrent une précision moyenne de 94,5%. La perspective de ces travaux est d'ajouter un troisième événement, comme l'imagination d'un mouvement de pied ou de la langue. Il est en effet assez complexe d'augmenter le nombre de commandes à prendre en compte par l'ICM. Cela implique d'adapter les blocs de filtrage voire d'en ajouter si les nouvelles DSLE à observer concernent d'autres bandes de fréquences.

Le principal avantage du paradigme des DSLE est le fait d'utiliser des événements internes. Ainsi une ICM utilisant cette méthode ne nécessite pas d'éléments supplémentaire pour générer les stimuli. Cependant, la difficulté d'augmenter le nombre de commandes détectables limite actuellement la résolution des systèmes proposés.

1.2.2.4 P300

L'onde P300 est un autre potentiel évoqué qui peut être utilisé. Son nom indique qu'il s'agit d'une onde positive qui apparait environ 300 ms après le début de l'événement. Elle peut aussi être appelée P3. Elle a été découverte en 1964 par Chapman et Bragdon [38]. Lors de tests d'EEG, ils ont remarqué qu'en cas d'exposition à 2 stimuli dont un beaucoup plus rare que l'autre, une onde d'amplitude positive apparaissait environ 300 ms après le début de l'événement le moins fréquent. L'onde P3 est divisée en 2 sous-ondes. La P3a est provoquée par un stimulus innatendu et est donc associée à l'effet de surprise. La P3b est quant à elle observable quand le sujet perçoit un stimulus imprévisible qui demande une réponse, comme par exemple compter le

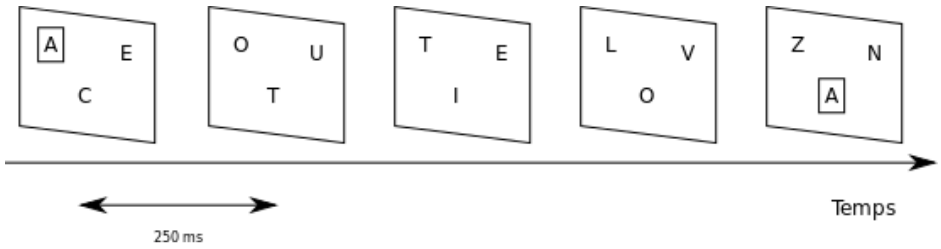


FIGURE 1.5 – Exemple de triple présentation visuelle rapide en série : si des potentiels P300 apparaissent pour le premier et le cinquième jeu de stimuli, la lettre "A" est transmise.

nombre d'occurrences ou chercher une lettre précise. Ce paradigme est souvent utilisé pour des ICM visant à écrire du texte lettre par lettre. Par exemple, une équipe chinoise a proposé un système liant l'utilisation de l'onde P300 à une triple présentation visuelle rapide en série [39]. Cela consiste à présenter à l'utilisateur un jeu de 3 lettres toutes les 250 ms comme présenté en Figure 1.5. Un potentiel P300 peut donc être détecté à chaque fois que la lettre recherchée est présente dans le champ visuel de l'utilisateur. La connaissance des jeux de lettres présentées permet donc de déterminer la lettre devant être écrite. Une autre équipe de Singapour propose quant à elle un système reposant sur un écran avec les 26 lettres de l'alphabet ainsi que les chiffres de 0 à 9 disposés en grille 6×6 [40]. Chaque ligne et chaque colonne est ainsi affichée selon une séquence pseudo-aléatoire avec un intervalle entre 2 stimuli de 187 ms. L'onde P300 permet ainsi de détecter la ligne et la colonne correspondant au caractère choisi et donc de déterminer ce dernier. L'ICM ainsi développée utilise un casque EEG disposant de 7 électrodes et le traitement est effectué par 3 processeurs, ayant chacun accès à une même mémoire. La premier processeur gère l'arrivée des données mesurées et horodatées par les électrodes et compare ces dates avec celles des affichages de lignes ou de colonnes afin de transférer uniquement les données pertinentes au deuxième processeur. Celui-ci transfert ces valeurs vers le cœur de calcul qui exécute le filtre. Le dernier processeur gère quant à lui l'affichage de la grille 6×6 sur un écran. La phase d'utilisation se décompose en cycles où un cycle correspond à une suite de stimuli où chaque ligne et chaque colonne sont affichées une fois. Les résultats d'expérimentations donne une précision de 65,37% en s'autorisant 2 cycles par détection de caractère. En passant à 5 cycles par caractère, la précision atteint la valeur de 93,66%. En reprenant l'intervalle de temps entre 2 stimuli, cette précision est donc atteinte en allouant 10,285 s par caractère. Le compromis à choisir entre précision et temps de réponse est fortement impactant pour cette méthode. De plus, cela nécessite d'une part un support visuel mais aussi un temps conséquent pour détecter une commande. Il est donc difficile d'adapter cette méthode pour des applications ICM à grande échelle.

1.2.3 Bilan sur les ICM utilisant l'EEG

L'EEG est une méthode de mesure non invasive qui se démarque dans le domaine des ICM. En effet, son coût réduit par rapport à d'autres moyens de mesure, ainsi que son faible encombrement en font une méthode de choix pour développer des ICM à

moindre coût. La majorité des systèmes proposés repose sur l'analyse des potentiels évoqués, qui correspondent à des modifications des rythmes cérébraux suite à l'exposition à des événements externes ou internes. Toutefois, ces systèmes souffrent du manque de résolution spatiale de la mesure par EEG, ce qui implique soit un nombre réduit de commandes réalisées par l'ICM, soit un temps de détection de commande allongé, ce qui éloigne la possibilité d'applications en temps réel et limite donc les perspectives actuelles du domaine de l'ICM. Pour lever ce verrou, de nouvelles méthodes utilisant la résolution du problème inverse de l'EEG peuvent mener à des ICM non invasives de haute résolution [41]. Cette méthode permet en effet de passer du modèle surfacique des signaux EEG à un modèle volumique des sources actives dans le cerveau. Cette méthode permet donc d'augmenter la résolution spatiale de l'EEG tout en conservant sa résolution temporelle, mais à un coût en complexité analysé dans la suite de ce chapitre.

1.3 Passage par le problème inverse de l'EEG

1.3.1 Définition du problème inverse de l'EEG

L'activité électrique du cerveau est un processus spatio-temporel, l'activité est en effet distribuée dans les 3 dimensions du cerveau et évolue dans le temps. La résolution des 2 domaines est donc primordiale pour analyser la précision de la méthode d'imagerie cérébrale utilisée. Si le point fort de l'EEG est sa résolution temporelle, de l'ordre de la milliseconde, la résolution spatiale est quant à elle bien plus limitée pour 2 raisons. La première de ces raisons est la limite de l'échantillonnage spatial. En effet, le système international 10-20 classique dispose les électrodes avec un écart moyen de 6 centimètres entre 2 d'entre elles [42]. Les évolutions récentes des systèmes d'EEG ont permis de proposer des systèmes de meilleures résolutions disposant de 64 à 256 électrodes. Par exemple, avec 124 électrodes, la distance entre 2 points de mesure est réduite à 2,5 centimètres [43]. La seconde raison du manque de résolution spatiale vient des différents effets de conduction. Pour atteindre les électrodes du casque EEG, les potentiels électriques générés par les neurones doivent traverser entre autres le tissu nerveux, le liquide cérébro-spinal, le crâne et le cuir chevelu. Ce trajet implique que le signal arrivant aux électrodes est atténué, distordu et bruité [44]. Il faut donc chercher à corriger ces défauts avec des techniques dédiées afin d'améliorer la résolution spatiale de l'EEG. C'est ici qu'entre en jeu des problèmes distincts mais étroitement liés : le problème direct et le problème inverse de l'EEG. En effet, leurs résolutions sont requises afin d'établir une cartographie haute résolution de l'activité électrique du cerveau à partir d'une mesure EEG.

A partir de la connaissance de la distribution des sources électriques dans le cerveau et de modèles des propriétés de conduction de la tête, le problème direct de l'EEG a pour but de déterminer le champ électrique généré par ces sources. La solution peut prendre la forme de potentiels électriques, comme les potentiels à la surface du cortex cérébral ou du cuir chevelu. Cette solution peut aussi être représentée par d'autres métriques comme par exemple une distribution de densité de courants. Le problème direct est bien défini et a une unique solution, définie par les équations de Maxwell [45]. La résolution de ce problème direct de l'EEG permet de mettre en place la relation entre

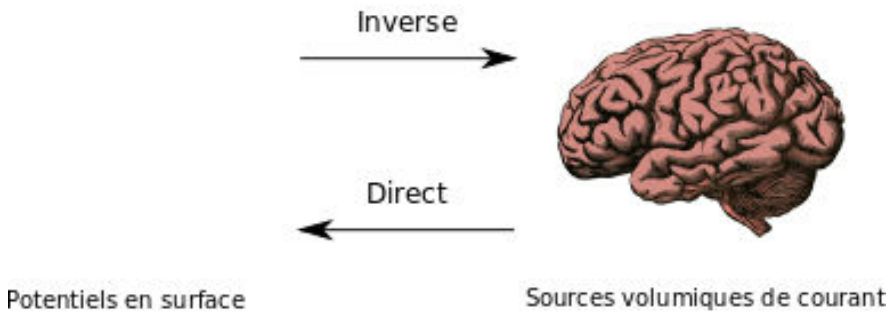


FIGURE 1.6 – Représentation du lien entre problème direct et problème inverse de l'EEG [5, 6].

les sources électriques dans le cerveau et ce qui est effectivement mesuré par les électrodes. Cette relation peut par la suite être représentée sous la forme d'une matrice de transfert dont les cœfficients sont dépendants de la géométrie et des caractéristiques de conduction de la tête du sujet étudié.

De l'autre côté, le problème inverse de l'EEG s'intéresse à déterminer l'emplacement et l'importance des sources électriques actives dans le cerveau à partir des potentiels mesurés par le casque EEG et des caractéristiques de conduction de la tête du sujet. Cette dualité entre problème direct et problème inverse est illustrée en Figure 1.6. Contrairement, au problème direct, le problème inverse n'admet pas de solution unique. En effet, Helmholtz a prouvé en 1853 qu'une infinité de configurations de sources dans un volume peut engendrer un jeu donné de potentiels mesurés à sa surface. Il faut donc ajouter d'autres contraintes afin d'obtenir une solution unique. Ces contraintes peuvent être anatomiques, physiologiques, spatio-temporelles ou encore fonctionnelles et sont issues des connaissances de la communauté sur le fonctionnement du cerveau. Lors de la résolution du problème inverse de l'EEG, plusieurs modèles de sources peuvent être considérés. Le modèle de sources isolées s'intéresse à déterminer l'activité électrique du cerveau à l'échelle du neurone, alors que le modèle de sources distribuées considère des groupements de sources situés dans les 3 dimensions du cerveau. Le choix du modèle de sources dépend de l'application visée, bien que l'objectif premier de la résolution du problème inverse de l'EEG reste le même : trouver une représentation des sources électriques dans le cerveau qui explique les potentiels mesurés par le casque EEG. La neuro-imagerie basée sur le problème inverse permet donc d'établir une cartographie haute-résolution du cerveau de manière non invasive et peu coûteuse.

Cette méthode de neuro-imagerie est déjà utilisée dans d'autres domaines comme l'étude de l'épilepsie. En effet, dans les cas où il s'agit d'une épilepsie pharmacorésistante, il est intéressant de rechercher l'emplacement du foyer épileptogène afin de déterminer si une opération chirurgicale est possible ou non. Les travaux de Michel [46] donnent un exemple de la procédure d'imagerie. Le patient est tout d'abord équipé d'un casque d'EEG à 128 électrodes pour une phase d'enregistrement d'environ une heure. Les signaux sont mesurés à une fréquence de 500 Hz et enregistrés. Par la suite, une observation visuelle exclut les signaux d'électrodes présentant des artefacts. Ces signaux sont remplacés par des signaux interpolés. Une fois le jeu de signaux à analyser défini, le but est de repérer l'instant où le plus de signaux possibles présentent

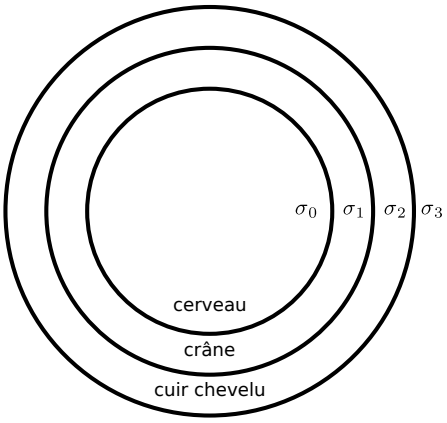


FIGURE 1.7 – Représentation d'un modèle inhomogène à 3 sphères.

une impulsion à la morphologie similaire (taux de corrélation de plus de 80%). Sur les 44 patients observés et présentés dans [46], le nombre moyen d'impulsions similaires est de 27 sur les 125 électrodes utilisées. Cet instant présentant le plus d'impulsions est par la suite utilisé pour déterminer l'origine des sources. La méthode de résolution de problème inverse de l'EEG est nommé EPIFOCUS [47]. Il s'agit d'une méthode de résolution linéaire qui ne considère qu'une seule source, même si celle-ci peut être étendue. Ces caractéristiques ne sont pas gênantes pour le cas de l'étude de l'épilepsie car le foyer épileptogène peut s'étendre sur plusieurs centimètres carrés. Si cette méthode permet de déterminer précisément l'emplacement des neurones concernés par l'épilepsie de ces patients, elle reste difficilement utilisable pour des applications d'ICM. En effet, elle est basée sur l'évolution temporelle des signaux, et non sur leur valeur à un instant donné. Cela nécessite donc de faire les traitements mathématiques une fois la séance de mesures finie.

Il faut noter que la précision de la localisation de sources par la résolution du problème inverse dépend directement de la modélisation des caractéristiques de conduction de la tête du sujet. Plusieurs types de modèles peuvent être utilisés, chacun ayant un niveau de précision différent. Par exemple, les modèles sphériques, homogènes dans le cas d'une seule sphère et inhomogènes pour les modèles à 3 ou 4 sphères, sont très bien connus et permettent donc d'avoir rapidement des solutions analytiques, qui conviennent parfaitement à des études de validation. La Figure 1.7 représente un modèle à 3 sphères concentriques où les σ_i représentent les conductivités des différentes zones. D'un autre côté, les modèles à géométrie réaliste, homogène ou non, décrivent plus précisément le modèle de conduction de la tête du patient, grâce à une imagerie IRM, mais nécessite l'usage de méthodes numériques comme la méthode des éléments finis, la méthode des différences finies ou encore la méthode des éléments frontière [18]. Pour illustrer l'importance de la précision des modèles utilisés, une équipe néerlandaise a montré qu'une légère différence d'épaisseur ou de forme du crâne pouvait mener à une erreur de localisation d'un centimètre, ce qui l'ordre de grandeur de l'erreur résultant de la multiplication ou division du facteur de conduction du crâne par 2 [48].

La neuro-imagerie par résolution du problème inverse de l'EEG est une voie intéressante pour proposer à terme une ICM non invasive et de haute résolution. Cette

méthode est déjà utilisée dans des domaines scientifiques qui n'ont pas de contraintes temporelles fortes. Dans le cas de l'étude de l'épilepsie, les calculs sont donc effectués après la séance de mesures. Il faut donc s'intéresser à une méthode où le traitement des données peut se faire en même temps que la mesure des signaux EEG.

1.3.2 Travaux de réduction de la complexité algorithmique de la résolution du problème inverse

Le premier objectif du projet dans lequel s'inscrit cette thèse est de réduire la complexité algorithmique de la résolution du problème inverse de l'EEG qui consiste à déterminer les sources de courant dans le cerveau responsables des potentiels mesurés sur le cuir chevelu du patient. Il est important de savoir que la résolution du problème inverse nécessite plusieurs solutions du problème direct [49]. En effet, chaque problème direct, donc toutes les combinaisons source électrique/électrode possibles, doit être préalablement résolu. Cela signifie que la moindre avancée dans les méthodes de résolution du problème direct a une grande importance pour les méthodes d'imagerie utilisant le problème inverse.

Dans la suite de ce Chapitre, un résumé des travaux d'une thèse antérieure est présenté. Cette thèse s'intéresse à la réduction de la complexité algorithmique du problème direct de l'EEG. Comme cela traite d'aspects mathématiques qui ne sont pas nécessaires à la compréhension de la suite de ce manuscrit, les résultats les plus importants sont mis en évidence.

Quand un modèle de tête réaliste est utilisé, la solution du problème direct de l'EEG ne peut être obtenue qu'en utilisant une méthode de résolution numérique, comme par exemple la méthode des éléments finis ou celle des éléments de frontière. La méthode aux éléments finis repose sur une discrétisation du volume du cerveau, tout en prenant en compte les inhomogénéités de conduction, ce qui implique au final un besoin conséquent de calculs. Il a été montré que le temps de calcul nécessaire peut être diminué afin de s'approcher de celui que demande la méthode des éléments de frontière, pour une précision équivalente [50]. Cette dernière méthode est effectivement moins gourmande en temps de calcul car elle demande de discrétiser uniquement les interfaces entre 2 régions ayant une conductivité différente. Ceci explique le fait que de nombreuses recherches ont été réalisées autour de cette méthode. La discrétisation de ces surfaces est réalisée en les approximant par des assemblages de triangles.

L'approche proposant une formulation symétrique [51] s'est vite imposée car elle permet un gain de précision tout en diminuant le temps de calcul nécessaire. Cependant, cette méthode a néanmoins un impact négatif sur le conditionnement de la matrice générée, qui se calcule en divisant la plus grande valeur singulière par la plus petite. En analyse numérique, le contionnement d'une matrice inversible sur laquelle est basée un système matriciel indique l'erreur relative de la solution lorsque les entrées sont perturbées. **Le conditionnement est donc un indicateur de la stabilité de la solution.** La formulation symétrique fait augmenter ce conditionnement quand la densité de discrétisation augmente. Il en est de même avec l'augmentation du contraste de conductivité. Dans le cadre de la modélisation de la tête, ce contraste correspond au ratio de la conductivité du cerveau, qui a la même valeur que la conductivité du cuir chevelu, par rapport à celle du crâne, notée σ. En utilisant des valeurs normalisées, le

contraste est donné par $\frac{1}{\sigma}$ avec σ variant de 1 à $\frac{1}{200}$ selon les modèles utilisés [52].

La valeur de conditionnement est importante pour déterminer la vitesse de résolution. En effet, la solution du problème direct de l'EEG est obtenue de manière itérative [53] et le nombre d'itérations nécessaires croit avec la valeur du conditionnement [54]. **De plus, le conditionnement joue également sur l'amplification des erreurs initiales. Il est donc très important de réduire au mieux cette valeur de conditionnement afin de réduire le temps de résolution et d'augmenter la précision.**

1.3.2.1 Proposition d'un préconditionneur de Calderon

L'apport de [55, 7] par rapport à cet état de l'art est de traiter les problèmes de conditionnement posés par la formulation symétrique, pour en optimiser les résultats pour des modélisations mieux discrétisées. La contrainte majeure est de proposer une solution conservatrice. En effet, la formulation symétrique a eu un tel impact qu'elle est notamment implémentée dans des outils de neuro-imagerie. L'amélioration apportée doit donc seulement provenir de calculs supplémentaires sans directement modifier les implantations déjà existantes. Les travaux effectués ont permis de proposer une méthode qui demande seulement de multiplier la matrice issue de la formulation symétrique à un préconditionneur basé sur les formules de Calderon [55]. Il a été montré que l'utilisation de ce préconditionneur conserve la précision apportée par la formulation symétrique, comme présenté en Figure 1.8. Cette figure compare l'erreur relative pour plusieurs formulations de la méthode aux éléments de frontière, dont la méthode proposée, nommée formulation Calderon symétrique. L'erreur relative est donnée en fonction du paramètre de discrétisation, d. Plus cette valeur est grande et plus le nombre de triangles approximant une même surface est conséquent. **Ces résultats montrent bien que la nouvelle formulation proposée n'affecte en rien la précision de la formulation symétrique.**

Il reste donc à étudier l'influence de la formulation Calderon symétrique sur le conditionnement de la matrice par rapport à la formulation symétrique classique. La comparaison peut être observée en Figure 1.9. Comme expliqué précédemment, **le conditionnement de la matrice issue de la formulation symétrique augmente considérablement avec la finesse de discrétisation alors que le conditionnement de la formulation Calderon symétrique reste stable.**

L'autre problème de la formulation symétrique classique réside dans sa stabilité face au contraste de conductivité entre 2 régions de la tête. Il faut donc aussi comparer à la fois l'erreur relative et le conditionnement de la formulation Calderon symétrique vis-à-vis de ce paramètre, Figures 1.10 et 1.11.

La formulation Calderon symétrique de la méthode aux éléments de frontière est donc plus stable à la variation du contraste de conductivité que la formulation symétrique classique, tout en conservant la précision atteinte. Tous ces éléments, notamment la stabilité de la valeur de conditionnement selon les 2 paramètres étudiés, **génèrent un impact important sur le nombre d'itérations nécessaires à la résolution du problème direct, Figue 1.12**. Cela offre des perspectives intéressantes dans le cas applicatif de la résolution du problème direct de l'EEG. **En effet, la réduction du nombre d'itérations nécessaires pour atteindre une précision souhaitée permet**

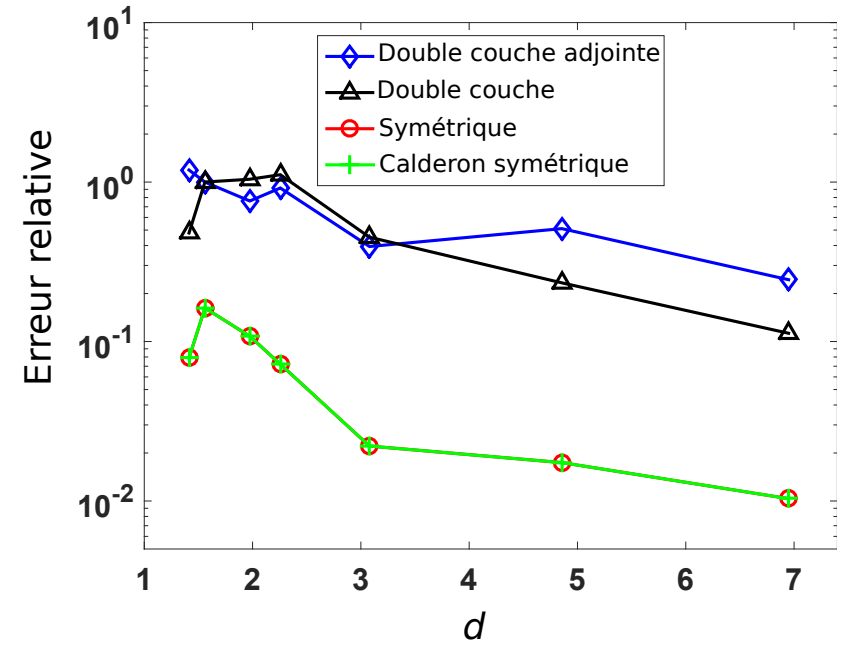


FIGURE 1.8 – Erreur relative selon le paramètre de discrétisation pour plusieurs formulation de la méthode aux éléments de frontière [7].

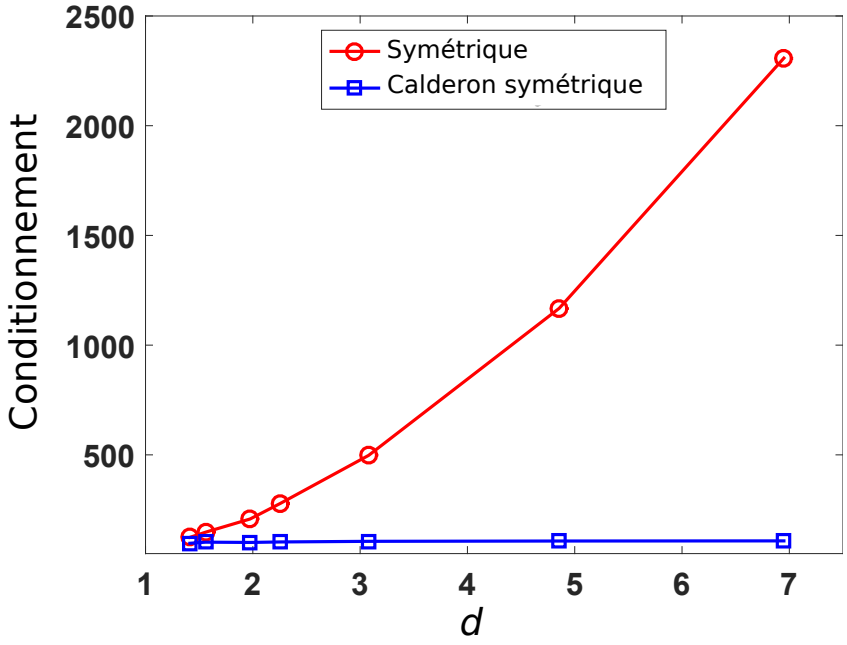


FIGURE 1.9 – Valeur de conditionnement selon le paramètre de discrétisation [7].

soit de diminuer le temps de calcul pour un résultat identique, soit de profiter du temps gagné pour exécuter des itérations supplémentaires et ainsi améliorer la précision du résultat obtenu.

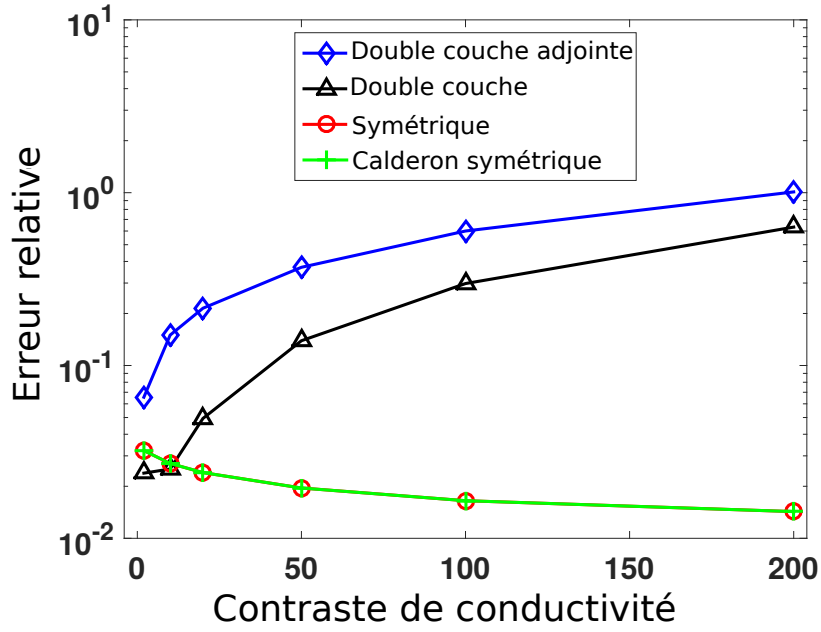


FIGURE 1.10 – Erreur relative selon le contraste de conductivité pour plusieurs formulation de la méthode aux éléments de frontière [7].

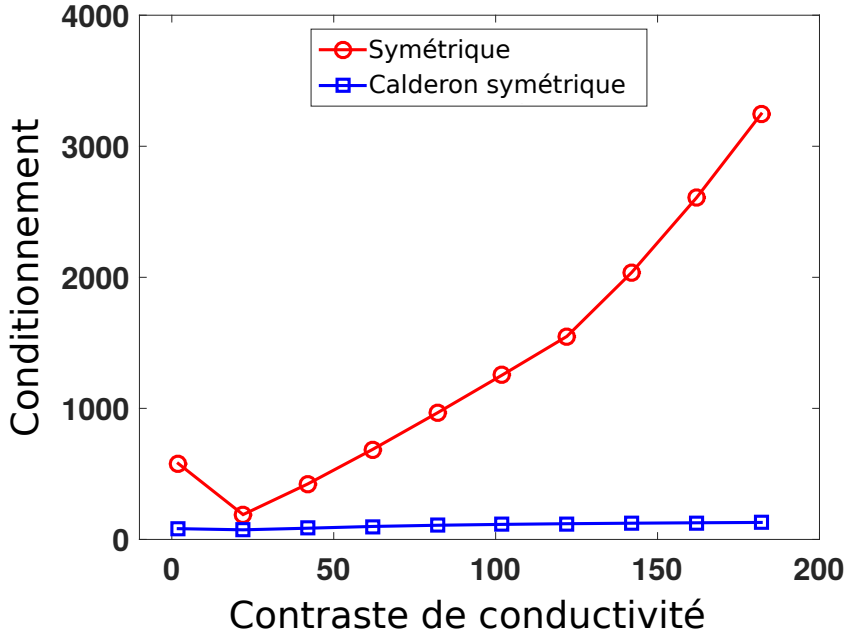


FIGURE 1.11 – Valeur de conditionnement selon le contraste de conductivité [7].

1.3.2.2 Amélioration de la résolution du système matriciel

L'utilisation de la formulation symétrique (ou Calderon symétrique) pour la résolution du problème direct de l'EEG aboutit à un système de la forme $\mathcal{Z}\vec{y} = \vec{b}$. Dans ce système Z est la matrice générée par la formulation symétrique, également appelé opérateur, b est un vecteur comprenant les conditions de dérivées partielles et $\vec{y}$ le vec-

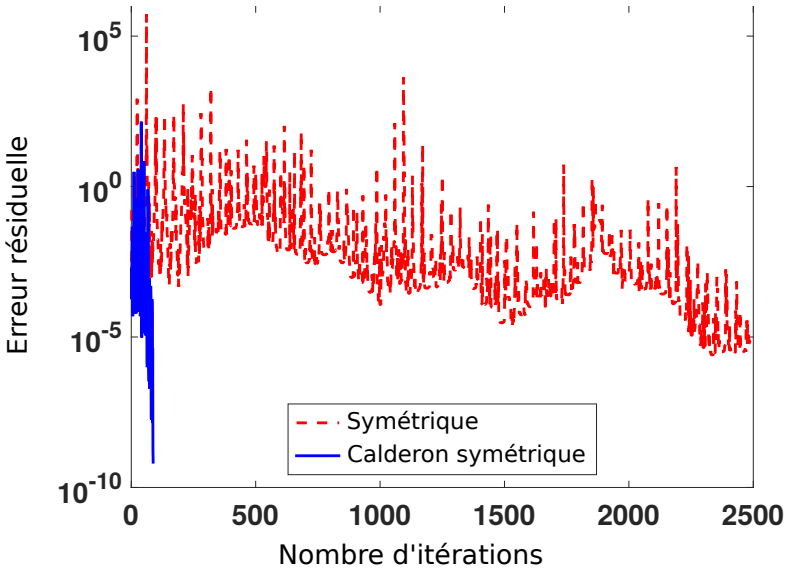


FIGURE 1.12 – Convergence de la méthode de résolution itérative pour la formulation symétrique préconditionnée ou non [7].

teur contenant les inconnues à retrouver. La solution la plus directe pour résoudre ce système consiste à inverser naïvement la matrice $\mathcal{Z}$ puis de la multiplier par le vecteur $\vec{b}$. Cette solution n'est cependant pas envisageable pour des systèmes de haute résolution car cette opération a une complexité algorithmique en $\mathcal{O}(N^3)$ avec N le nombre d'inconnues. Il est alors préférable d'utiliser des méthodes de résolution rapides, algorithmes qui accélèrent la résolution d'un système matriciel. Ces méthodes peuvent être itératives. **Elles demandent alors de réaliser le PMV $\mathcal{Z}\vec{x_j}$ où la valeur $\vec{x_j}$ représente l'approximation de la solution à la j^e itération. Sans technique particulière, ce PMV a une complexité algorithmique en $\mathcal{O}(N^2)$. Il est donc important de réduire cette complexité pour contrer cet aspect itératif.**

La méthode employée ici est l'approximation adaptative croisée [56, 57]. En effet, cette méthode permet, dans le cas de la formulation symétrique du problème direct de l'EEG, de passer d'une complexité en $\mathcal{O}(N^2)$ à une complexité en $\mathcal{O}(Nlog(N))$. **Cet algorithme permet de compresser la matrice opérateur, ce qui permet de réduire le nombre de valeurs à garder en mémoire et ainsi réduire le temps pour effectuer un PMV.** Le Tableau 1.1 montre l'effet de cette compression sur le temps de calcul d'un PMV par rapport au nombre d'inconnues. Si les ordres de grandeur sont les mêmes pour un nombre limité d'inconnues, la différence de complexité permet un gain de temps de calcul non négligeable quand le nombre d'inconnues croît. Ainsi, pour environ 7 500 inconnues, le temps de calcul d'un PMV d'une matrice dense est sensiblement identique à celui d'une matrice compressée mais en multipliant le nombre d'inconnues par environ 6,5, le ratio du temps de calcul atteint la valeur de 8.

Après s'être assuré que l'usage de la compression de l'opérateur permettait d'éviter l'explosion des temps de calculs en voulant augmenter la précision du système global, il faut vérifier qu'il en est de même avec le temps de génération de l'opérateur lui-même. La Figure 1.13 présente la comparaison du temps de calcul des opérateurs symétrique dense, symétrique compressé et Calderon symétrique

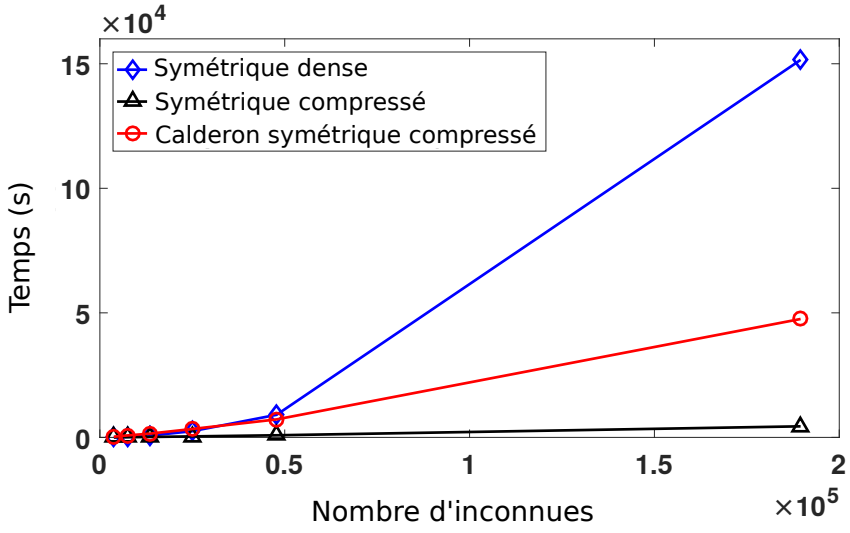


FIGURE 1.13 – Temps de calcul des opérateurs selon le nombre d'inconnues [7].

compressé.

L'usage de la compression d'opérateur a donc le même impact sur la génération de ce même opérateur que sur le temps de calcul du PMV. En effet, pour un nombre d'inconnues peu élevé, les temps de calcul des 3 opérateurs sont équivalents, mais dès que la résolution du système augmente, la formulation dense va demander beaucoup plus d'efforts de calcul que les formulations compressées, ce qui valide donc le choix d'utiliser de la compression de matrice afin de réduire la complexité de résolution du problème direct de EEG.

Cependant, une différence assez conséquente de temps de calcul peut être observée entre les opérateurs symétrique compressé et Calderon symétrique compressé. **Il en faut pas oublier que la version préconditionnée se base sur la version symétrique classique, il est donc normal d'avoir un surplus de calcul.** Pour donner une idée de l'importance de ce surcoût, voici la liste des opérations à effectuer pour passer d'une résolution du problème direct de l'EEG via la formulation symétrique classique, qui demande de résoudre le système matriciel $\mathcal{Z}\vec{x} = \vec{b}$, à une résolution utilisant la formulation Calderon symétrique :

— génération de la matrice de la formulation symétrique standard, nommée $\mathcal{Z}$;
— génération du préconditionneur de Calderon, nommé $\mathcal{C}$;

Tableau 1.1 – Temps d'exécution en seconde sur une station de travail d'un PMV en fonction du nombre d'inconnues N. [7]

N	Matrice dense	Matrice compressée
7 430	0,024	0,017
13 608	0,073	0,033
24 921	0,236	0,078
47 790	0,787	0,100

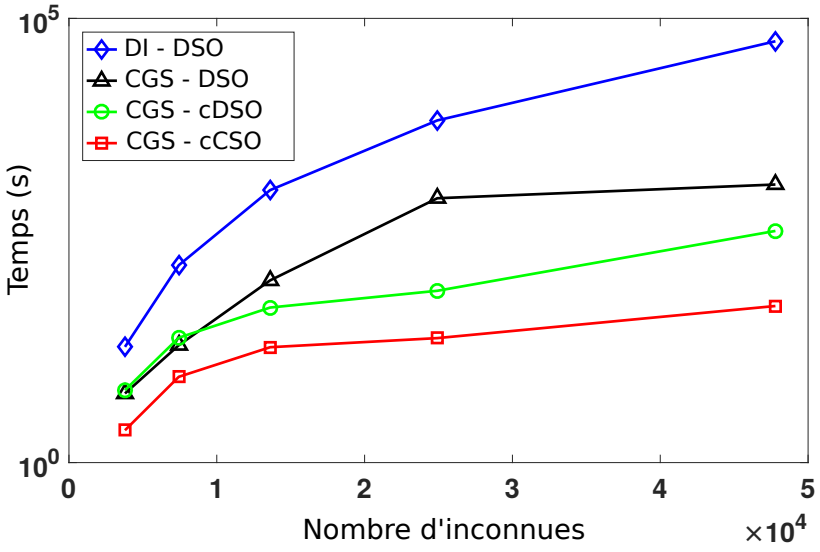


FIGURE 1.14 – Temps de résolution du problème direct selon plusieurs méthodes : *Direct Inversion (DI), Dense Symmetric Operator (DSO), Compressed Symmetric Operator (cSO), Compressed Calderon Symmetric Operator (cCSO)* [7].

— génération de la matrice de Gramm liée à la discrétisation standard, nommée $\mathcal{G}$;
— normalisation de $\mathcal{C}$ et $\mathcal{Z}$ avec la matrice de régularisation nommée $\mathcal{Q}$;
— exécution de la multiplication matricielle suivante : $\mathcal{Z}_c = \mathcal{Q}\mathcal{C}\mathcal{Q}\mathcal{G}^{-1}\mathcal{Q}\mathcal{Z}\mathcal{Q}$;
— modification de la partie droite du système b de manière semblable : $\vec{b_c} = \mathcal{Q}\mathcal{C}\mathcal{Q}\mathcal{G}^{-1}\mathcal{Q}\vec{b}$;
— résolution du système $\mathcal{Z}_c\vec{y} = \vec{b_c}$;
— obtention de la solution en utilisant $\vec{x} = \mathcal{Q}\vec{y}$.

Le nombre d'opérations nécessaires est donc conséquent. Cela explique donc la différence de temps de calcul entre la formulation symétrique classique et la formulation Calderon symétrique. [7] a par la suite étudié le temps de calcul d'une solution de problème direct, Figure 1.14.

La première observation pouvant être faite est que la résolution par l'inversion directe de l'opérateur est la plus mauvaise au point de vue du temps de calcul et de la stabilité quand le nombre d'inconnues augmentent. Pour les autres solutions, la méthode itérative du gradient conjugué carré est utilisée pour résoudre le système matriciel, ce qui permet d'améliorer considérablement le temps de calcul par rapport à une inversion directe. **Globalement, même si le temps de génération de l'opérateur Calderon symétrique compressé est plus long que celui de l'opérateur symétrique compressé classique, la stabilité du conditionnement permet de réduire considérablement le nombre d'itérations, et donc le temps total de calcul d'une solution du problème direct de l'EEG.** Enfin, la méthode d'utilisation du préconditionneur de Calderon avec compression de l'opérateur, a été testé sur un cas réel de modèle de tête obtenu par IRM. Le but de ce cas est de générer ce qu'on appelle la *Lead-Field Matrix* (LFM), qui est une matrice qui représente le modèle de propagation entre les sources électriques dans le cerveau et les électrodes du casque EEG. De plus, cette LFM est

Tableau 1.2 – Temps de calcul pour fournir un opérateur, une solution de problème direct et un LFM selon plusieurs méthodes [7].

Opérateur	Méthode	Opérateur (s)	Problème direct (s)	LFM (h)
symétrique dense	inversion directe	10845,67	54609,99	321,57
symétrique dense	CGS	10845,67	7294,61	45,56
symétrique compressé	CGS	1436,63	2322,89	13,95
Calderon compressé	CGS	7888,86	62,40	2,56

l'élément clé permettant de résoudre par la suite le problème inverse de l'EEG [49]. Le Tableau 1.2 présente les résultats de la comparaison de différentes méthodes en considérant un système avec 1500 sources électriques et un casque EEG de 21 électrodes.

Même si la formulation Caldéron symétrique compressée demande environ 5,5 fois plus de temps que la version non préconditionnée pour générer l'opérateur, cette méthode permet d'aller 5,5 fois plus rapidement en ce qui concerne la génération de la LFM. Ce gain est notamment expliqué par la meilleur convergence de la méthode proposée, et montre donc que le surplus initial de calculs est bénéfique pour la suite. De même, le gain sur la résolution du problème direct de l'EEG est considérable, passant de presque 40 minutes à une seule minute. L'utilisation de la compression des opérateurs apporte également un gain en terme d'occupation de la mémoire. En effet, l'utilisation de matrices denses demandent plus de 16 GB d'occupation, alors que les versions compressées en demandent moins de 3 GB [7]. **Comme vu dans le Tableau 1.2, les changements algorithmiques proposés permettent de diviser par environ 5,5 le temps de génération de la LFM par rapport à l'état de l'art.** L'autre point intéressant est la distribution de ce temps de calcul. En effet, ce temps de génération comporte tout d'abord le temps de création de la matrice opérateur, puis le temps de résolution du problème direct pour chaque électrode. Ce qui donne :

$$T_{LFM} = T_{Op} + N_e \times T_{Pd} \tag{1.1}$$

où T_{LFM} est le temps de génération de la LFM, T_{Op} celui de la matrice opérateur, T_{Pd} le temps de la résolution d'un problème direct et N_e le nombre d'électrodes. **La formulation Calderon compressé permet ainsi de concentrer 85,6% du temps de calcul du LFM dans le temps de calcul de l'opérateur. A titre de comparaison, le temps de calcul de l'opérateur symétrique compressé représente seulement 2,8% du temps de calcul de la LFM. Optimiser le temps de génération de l'opérateur Calderon compressé réduit donc également de façon conséquente le temps de génération de la LFM. Il faut rappeler que ces résultats concernent un système qui considère 1500 sources et 21 électrodes, ce qui dimensionne directement la taille des matrices utilisées. Augmenter le nombre d'électrodes augmente donc à la fois le nombre de cœfficients à calculer pour la matrice opérateur, le nombre de problèmes directs à résoudre mais aussi le temps pour résoudre l'un d'entre eux. Il est donc pertinent d'étudier une approche d'accélération matérielle pour réduire au mieux ces temps de calcul et ainsi accélérer la mise en œuvre de systèmes ICM.**

1.3.3 Perspective de l'accélération matérielle

Les calculs nécessaires pour obtenir chaque cœfficient de la matrice opérateur sont indépendants les uns des autres. Il est donc intéressant d'envisager l'usage d'un accélérateur dédié qui effectue ces calculs en parallèle. A la suite des travaux de [55, 7], un profilage de l'algorithme générant la matrice opérateur a été réalisé. Une fonction du code a ainsi été identifiée comme étant susceptible de profiter grandement d'une accélération matérielle. En effet, les appels à cette fonction sont nombreux et indépendants et elle contient des opérations non-linéaires très gourmandes en temps de calcul comme la division, la racine carrée, le logarithme népérien ou l'arctangente. L'analyse de cette fonction est faite au cours du Chapitre 2.

Réussir à accélérer le temps de calcul de cette fonction est donc le prochain verrou à lever afin de réduire de nouveau le temps de calcul de la LFM. Cela a plusieurs intérêts. A court terme, cela permet d'accélérer les différents tests lors de la phase de recherche algorithmique. En effet, dans l'hypothèse d'une division par 100 du temps de construction de l'opérateur Calderon compressé, le temps de calcul de la LFM passe de 2 heures et 33 minutes à seulement 23 minutes, ce qui permet d'être beaucoup plus flexible pour tester différentes pistes, en plus de simplifier et accélérer les protocoles de tests cliniques, à la fois pour les volontaires et pour l'équipe de recherche. En effet, réduire le temps d'attente entre la modélisation de la tête du patient et le test d'une ICM permet également de faire passer beaucoup plus de volontaires sur une journée de tests, sans pour autant devoir multiplier les machines de calcul afin de paralléliser. A plus long terme, dans le cas d'une application ICM, la réduction du temps de calcul permet une mise en fonctionnement plus rapide du dispositif, cela permet aussi d'envisager la possibilité de réaliser des recalibrages réguliers pour maintenir les performances de l'ICM sur la durée.

Les différents verrous à lever à court terme lors de cette intégration matérielle sont :

— accessibilité : le domaine de l'ICM est relativement récent et aucun consensus de méthode n'existe aujourd'hui. La partie accélérée doit donc répondre aux besoins de la majorité de la communauté ;
— précision : l'accélération matérielle peut entrainer une perte de précision. Il faut donc veiller à ce que le temps de calcul des itérations supplémentaires pour compenser cette perte ne dépasse pas le gain fourni par l'accélération ;
— haut débit : l'accélération passe par une parallélisation des calculs afin d'augmenter le débit de calcul ;
— faible latence : il faut que l'appel à l'accélérateur matériel soit le moins impactant possible pour le déroulement du reste de l'algorithme. Il faut donc que le système central soit le moins longtemps possible en famine.

Pour les perspectives à moyen et long terme, deux points sont à étudier :

— évolutivité : le domaine de l'ICM comme les technologies d'accélération possibles évoluent rapidement. La solution proposée doit donc être facilement adaptable à ces évolutions. L'évolutivité peut concerner la précision, le format des données en entrée et en sortie de l'accélérateur ou encore la généricité des modules utilisés ;
— encombrement : dans l'optique d'une limitation de coûts, il faut penser à limiter la quantité de ressources de calcul et de mémorisation, ainsi que de réfléchir à la technologie idéale pour une ICM portable et grand public.

1.4 Conclusion

L'EEG est la méthode de prise de mesure non invasive la plus étudiée dans le cadre de l'ICM. En effet, elle est la plus confortable et la moins encombrante pour l'utilisateur, tout en proposant une très bonne résolution temporelle. Ces points forts ont permis le développement de plusieurs ICM, notamment basées sur l'étude des potentiels liés à l'événement. Ces potentiels sont directement repérables sur les signaux mesurés sur le cuir chevelu et ont besoin d'un stimuli, interne ou externe. Cependant, le fonctionnement de certains potentiels évoqués limite les performances des ICM du point de vue du temps de réponse ou du confort de l'utilisateur, limitant ainsi le temps d'utilisation. De plus, le point noir de l'utilisation de l'EEG réside dans sa résolution spatiale, qui est insuffisante. Cela implique globalement des ICM capables de détecter un nombre limité de commandes utilisateur.

Un axe de recherche visant à gommer ce défaut concerne la résolution du problème inverse de l'EEG. Cela consiste de passer du modèle surfacique mesuré par les électrodes du casque EEG a un modèle volumique de sources électriques dans le cerveau de l'utilisateur. Cela permet notamment d'obtenir la résolution spatiale que l'on souhaite, en choisissant le nombre de sources électriques étudiées. Cette méthode est déjà utilisée dans d'autres domaines d'application, comme l'étude de l'épilepsie.

La résolution du problème inverse passe par la conception de la LFM, matrice construite à partir de la résolution de tous les problèmes directs du système. Il est donc logique de chercher à réduire le temps de calcul du problème direct. Des apports de l'équipe partenaire travaillant sur l'aspect algorithmique ont permis de réduire la complexité des calculs nécessaires, ainsi que d'accélérer la convergence des algorithmes déjà utilisés par la communauté scientifique. L'accélération des calculs n'est toutefois pas suffisante, et le temps nécessaire pour générer l'élément clé de la résolution du problème inverse de l'EEG est encore trop conséquent. Cela retarde donc les recherches actuelles et exclut actuellement la possibilité d'un recalibrage rapide d'une ICM. Il peut donc être intéressant d'envisager de déporter une partie des calculs nécessaires vers un accélérateur matériel, ce qui offrirait la possibilité d'obtenir un parallélisme bien supérieur à ce que peut offrir un processeur et ainsi atteindre un niveau d'accélération de temps de calcul supplémentaire.

L'objectif des travaux présentés dans la suite de ce document est de proposer une solution pour l'accélération matérielle du calcul de la matrice opérateur nécessaire à la résolution du problème inverse. Si la rapidité des calculs est évidemment la priorité, la précision des résultats n'est pas à négliger. Il est aussi intéressant d'assurer que la solution proposée soit facilement adaptable aux futures évolutions technologiques. Le fait de pouvoir ainsi traiter un grand nombre de données en un temps réduit, permettra à court terme de simplifier et accélérer les différents protocoles de test de la communauté travaillant sur l'utilisation du problème inverse pour des applications d'ICM. Sur un plus long terme, cela aide à la mise en place d'une ICM commerciale basée sur la résolution du problème inverse de l'EEG.

Chapitre 2

Analyse de l'algorithme ciblé et de son potentiel d'accélération

2.1 Introduction

Ca chapitre aborde la perspective d'accélération d'une partie de l'algorithme générant la matrice opérateur. Dans un premier temps, la fonction suite à un profilage est présentée, en mettant en valeur pourquoi elle se prête bien à une accélération matérielle mais aussi les difficultés que cela implique. Ensuite, une comparaison entre plusieurs moyens d'accélération sera proposée.

2.2 Présentation de la partie de code à accélérer

2.2.1 Fonctionnalité

Il a été montré dans le Chapitre 1 que les travaux des auteurs de [55, 7] ont permis de réduire considérablement le temps de conception de la LFM et que la majorité du temps de calcul est désormais dédié à la création de la matrice opérateur. Cette matrice est issue de la formulation symétrique classique à laquelle est appliquée un préconditionnement. Un profilage de la génération de la matrice opérateur Calderon symétrique a mis en évidence une fonction qui requiert la majorité du temps de calcul. Cette fonction est utilisée pour le calcul de chaque cœfficient de la matrice opérateur de la formulation symétrique, qui est la formulation la plus étudiée dans le cadre de la méthode des éléments de frontière. Le fait d'accélérer cette fonction profite donc à toute la communauté scientifique travaillant sur la résolution du problème direct de l'EEG. Il s'agit d'une fonction d'intégration en 3 dimensions proposée par Roberto Graglia [8]. Il s'agit d'une intégration sur un triangle par rapport à un point d'observation, ce qui est cohérent avec le fait que, dans le cas d'application considéré, les surfaces sont approchées par des triangles.

La Figure 2.1 peut être utilisée comme support pour visualiser comment se déroule cette intégration. Pour chaque triangle, une base locale ($\hat{u}$,$\hat{v}$,$\hat{w}$) est définie de sorte à ce que le triangle soit contenu dans le plan ($\hat{u}$,$\hat{v}$). Les nœuds du triangles sont numérotés de 1 à 3 dans le sens anti-horaire avec le nœud 1 sur l'origine de la base et le nœud 2 sur le vecteur $\hat{u}$. Les côtés du triangle à l'opposé des nœuds i sont notés δT_i, et ont

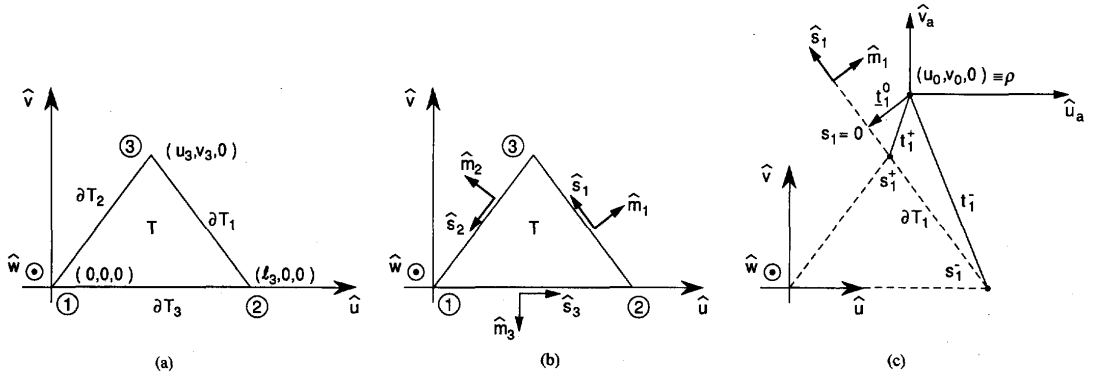


FIGURE 2.1 – (a) Définition de la base locale selon le triangle considéré. (b) Définition des vecteurs tangents $\hat{s}$ et normaux $\hat{m}$ sur le contour du triangle. (c) Exemple de la distance entre le côté 1 et ρ, la projection du point d'observation [8].

pour longueur l_i. Ce qui donne dans le plan ($\hat{u}$, $\hat{v}$) les coordonnées de points suivantes, comme indiqué en Figure 2.1 (a) :

$$\text{nœud } 1 = (0,0), \tag{2.1}$$

$$\text{nœud } 2 = (l_3,0), \tag{2.2}$$

$$\text{nœud } 3 = (u_3,v_3). \tag{2.3}$$

Ensuite, les vecteurs unitaires $\hat{s}_i$ sont définis, colinéaires aux côtés δT_i, et orientés dans le sens trigonométrique, des vecteurs normaux $\hat{m}_i$ y sont associés et dirigés vers l'extérieur du triangle comme le montre la Figure 2.1 (b). Le point d'observation pour l'intégration, nommé **r**, a pour coordonnées (u_0, v_0, w_0), ce qui permet de définir le point $\boldsymbol{\rho}$, de coordonnées triangle. Pour effectuer le calcul de l'intégration, il faut d'abord déterminer la distance entre $\boldsymbol{\rho}$ et le premier et dernier nœud de chaque côté, respectivement ${t_i}^-$ et ${t_i}^+$. Il faut aussi connaitre la distance entre $\boldsymbol{\rho}$ et les différents côtés, ${t_1}^0$ ainsi que les longueurs entre la projection de $\boldsymbol{\rho}$ et chaque nœud d'un côté, notées ${s_i}^-$ et ${s_i}^+$. Un exemple de ces notations est visible en Figure 2.1 (c) pour le côté 1 du triangle.

Ces différentes valeurs peuvent être trouvées grâce aux formules suivantes :

$${s_1}^- = -\frac{(l_3-u_0)(l_3-u_3)+v_0v_3}{l_1}, \tag{2.4}$$

$${s_1}^+ = \frac{(u_3-u_0)(u_3-l_3)+v_3(v_3-v_0)}{l_1}, \tag{2.5}$$

$${s_2}^- = -\frac{u_3(u_3-u_0)+v_3(v_3-v_0)}{l_2}, \tag{2.6}$$

$${s_2}^+ = \frac{u_0u_3+v_0v_3}{l_2}, \tag{2.7}$$

$${s_3}^- = -u_0, \tag{2.8}$$

$$s_3{}^{+} = l_3 - u_0, \tag{2.9}$$

$$t_1{}^{0} = \frac{v_0(u_3 - l_3) + v_3(l_3 - u_0)}{l_1}, \tag{2.10}$$

$$t_2{}^{0} = \frac{u_0 v_3 - v_0 u_3}{l_2}, \tag{2.11}$$

$$t_3{}^{0} = v_0, \tag{2.12}$$

$$t_3{}^{+} = t_1{}^{-}, \tag{2.13}$$

$$t_2{}^{-} = t_1{}^{+}, \tag{2.14}$$

$$t_3{}^{-} = t_2{}^{+}, \tag{2.15}$$

$$t_1{}^{-} = \sqrt{(l_3 - u_0)^2 + v_0{}^2}, \tag{2.16}$$

$$t_1{}^{+} = \sqrt{(u_3 - u_0)^2 + (v_3 - v_0)^2}, \tag{2.17}$$

$$t_2{}^{+} = \sqrt{u_0{}^2 + v_0{}^2}. \tag{2.18}$$

Il reste à définir les valeurs $R_i{}^0$, $R_i{}^-$ et $R_i{}^+$, respectivement les distances séparant le point d'observation des côtés des triangles et des différents nœuds. Ces valeurs sont facilement trouvables en utilisant le théorème de Pythagore. La valeur de l'intégrale finale qui est recherchée ici est donnée par la formule suivante :

$$I = -|w_0| \sum_{i=1}^{3} \beta_i + \sum_{i=1}^{3} t_i^0 f_{2i}, \tag{2.19}$$

avec

$$\beta_i = \arctan \frac{t_i{}^0 s_i{}^+}{(R_i{}^0)^2 + |w_0| R_i{}^+} - \arctan \frac{t_i{}^0 s_i{}^-}{(R_i{}^0)^2 + |w_0| R_i{}^-}, \tag{2.20}$$

$$f_{2i} = \ln \left(\frac{R_i{}^+ + s_i{}^+}{R_i{}^- + s_i{}^-}\right). \tag{2.21}$$

L'intégration est donc divisée en deux grandes étapes. La première consiste à calculer les différentes valeurs de projection à partir des coordonnées du triangle. La seconde utilise ces différentes valeurs pour calculer le résultat final, avec toutefois des disctinctions de cas nécessaires selon la valeur des dénominateurs dans les formules de β_i et f_{21}.

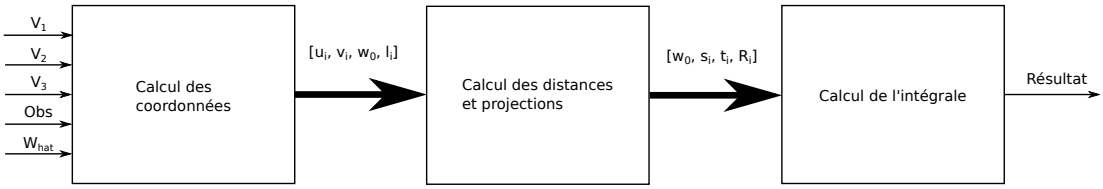


FIGURE 2.2 – Représentation graphique de la structure de la fonction d'intégration.

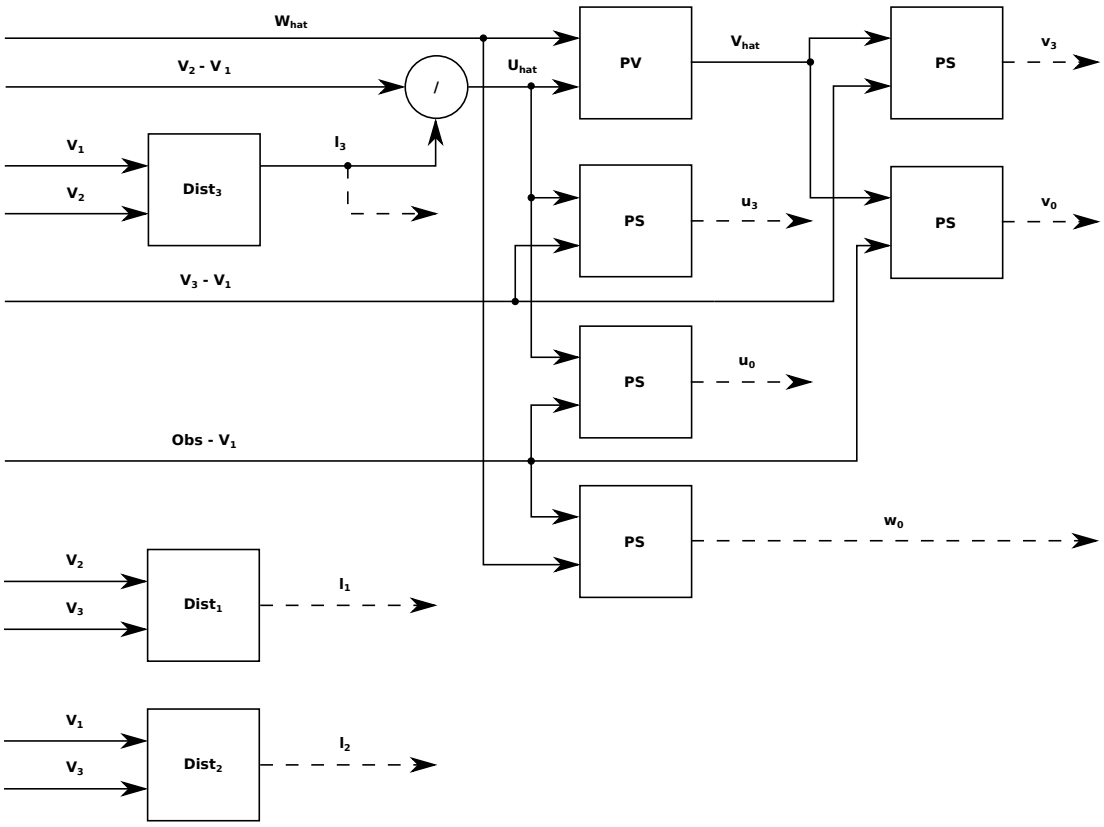


FIGURE 2.3 – Représentation graphique du calcul des coordonnées.

2.2.2 Analyse de la fonction

Il est intéressant d'étudier plus en profondeur l'implantation logicielle de cette fonction d'intégration pour en déterminer les points clés en vue de son accélération. Trois grandes parties peuvent être distinguées : le calcul des coordonnées, celui des distances et projections et enfin celui du résultat final 2.2. Dans cette sous-section, les prochaines figures permettent d'observer plus en détail l'enchainement des calculs pour chacun des blocs. La Figure 2.3 correspond à la partie dédiée au calcul des coordonnées. Les entrées sont les coordonnées dans le repère global des points du triangle V_i, celle du point d'observation Obs ainsi que le vecteur W_{hat} qui représente le vecteur $\hat{w}$ de la base locale, calculé en amont de cette fonction. Les blocs nommés $Dist_i$ calculent la distance l_i entre les deux points concernés. Cette distance est donnée par la formule suivante, correspondante ici à l'exemple de l_1 :

$$l_1 = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2 + (z_3 - z_2)^2}, \tag{2.22}$$

où (x_i,y_i,z_i) représentent les coordonnées du point i dans le repère global.

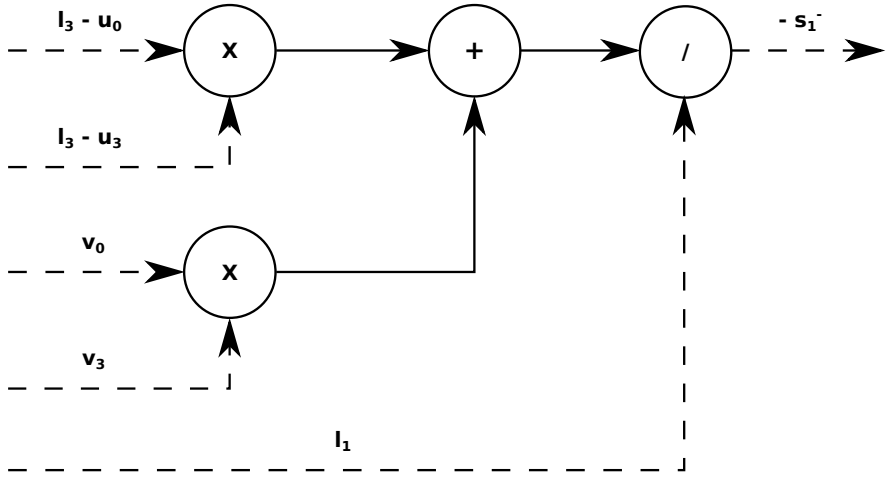


FIGURE 2.4 – Représentation graphique du calcul d'une projection

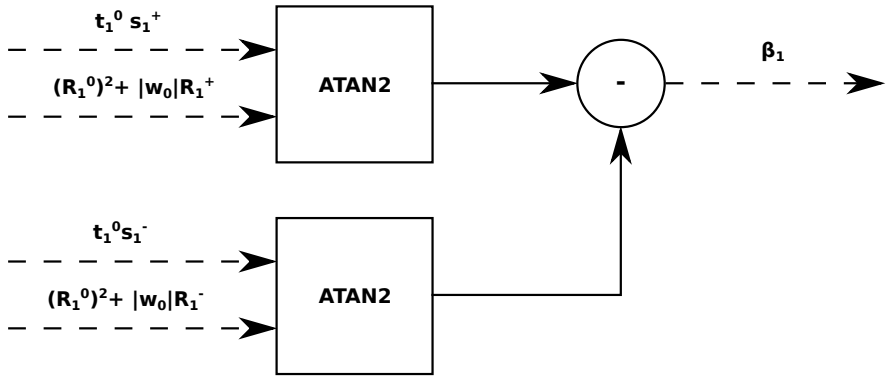


FIGURE 2.5 – Représentation graphique du calcul de β_1.

Les blocs PV et PS représentent respectivement des blocs de produit vectoriel et scalaire entre deux vecteurs. La première étape du calcul consiste à passer du repère global au repère local, tout en calculant les distances l_i. Tout d'abord, l'origine du repère local est placé sur V_1 en soustrayant ses coordonnées aux autres points donnés en entrée de la fonction. Il faut ensuite déterminer les autres vecteurs formant la base locale, $\hat{u}$ et $\hat{v}$ pour enfin déterminer les différentes coordonnées dans cette nouvelle base. Toutes ces données sont utilisées par la suite pour réaliser les formules présentées en section 2.2.1.

La Figure 2.4 présente le calcul de la projection s_1^-. Il peut être noté que la construction de cette valeur suit un d'arbre de calcul où les branches se rejoignent progressivement au fil des opérations. La dernière de ces opérations est une division par l_1, valeur calculée dès le début de la fonction. La partie qui gère le calcul des distances et projections contient plusieurs structures similaires en parallèle. Ce bloc présente aussi de la dépendance de données puisque les valeurs t_i sont nécessaires pour calculer les valeurs R_i.

Pour le calcul de l'intégrale finale, deux sous-blocs peuvent être définis : un bloc de calcul des β_i et un bloc de calcul des f_{2i}. Le calcul de β_1 est représenté en Figure 2.5. Par rapport à l'equation 2.20, il s'agit de la fonction $atan2$ qui est utilisée. Cette fonction est une variante à deux arguments de la fonction arctangente classique.

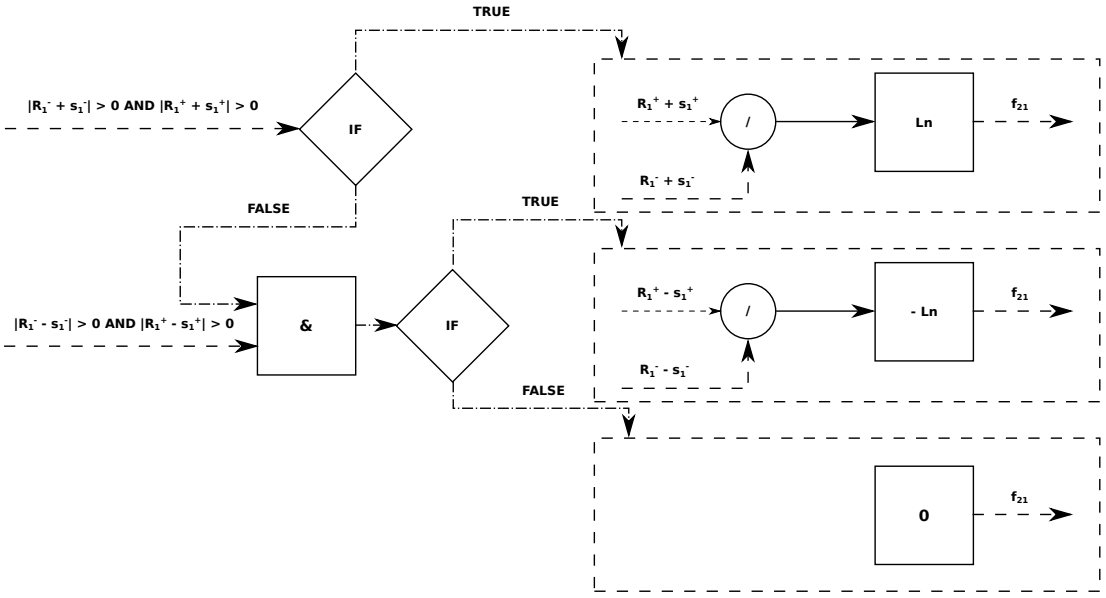


FIGURE 2.6 – Représentation graphique du calcul de f_{21}.

Pour $y \neq 0$, cette fonction suit la définition suivante [58] :

$$atan2(y,x) = \begin{cases} \varphi sgn(y) & \text{si} \quad x > 0, \\ \frac{\pi}{2} sgn(y) & \text{si} \quad x = 0, \\ (\pi - \varphi) sgn(y) & \text{si} \quad x < 0, \end{cases} \tag{2.23}$$

où φ est l'angle dans l'intervalle $[0, \frac{\pi}{2}[$ avec $\tan \varphi = \left|\frac{y}{x}\right|$ et sgn est la fonction signe. Pour $y = 0$, la définition devient :

$$atan2(0,x) = \begin{cases} 0 & \text{si} \quad x > 0, \\ \text{Non défini} & \text{si} \quad x = 0, \\ \pi & \text{si} \quad x < 0. \end{cases} \tag{2.24}$$

Cette méthode évite donc la division entre les deux opérandes, ce qui peut être intéressant selon la différence de complexité entre les implantations des fonctions $arctan$ et $atan2$. Le bloc de génération de la valeur f_{21} est représenté en Figure 2.6, les autres valeurs f_{2i} suivent un schéma similaire. Il s'agit de la seule étape de la fonction qui fait appel à de la logique conditionnelle. En effet, il faut prévoir les cas où le dénominateur de l'argument du logarithme népérien est nul, ce qui donne donc trois résultats possibles pour les valeurs f_{2i}.

Plusieurs observations concernant le potentiel d'accélération peuvent être faites sur cette fonction. Tout d'abord, tous les appels de cette fonction sont indépendants, ce qui fait qu'il est possible de réaliser plusieurs appels simultanément si les entrées son disponibles. La fonction étudiée est majoritairement un flot de données. Le seul besoin en logique se situe au niveau du calcul des f_{2i}. Il n'y a pas d'instructions itératives et rien ne justifie la présence d'une machine d'état, ce qui permet facilement d'effectuer un traitement en pipeline. De plus, à l'intérieur d'un même appel de fonction, la majorité des calculs demandés peut être parallélisé. Il y a donc deux grands axes possibles pour accélérer les calculs :

— accélération d'un appel de fonction : l'exécution d'un unique appel de fonction peut être optimisé en profitant au maximum des différents parallélismes internes possibles, ce qui demande néanmoins d'augmenter les ressources de calcul ;
— augmenter le nombre d'appels simultanés : ceci peut-être réalisé en dupliquant les ressources de calcul ou avec la mise en place d'un pipeline, ce qui n'est pas non plus sans impact pour les ressources nécessaires.

Il est également tout à fait possible d'envisager une stratégie d'accélération combinant ces deux axes, sachant que les possibilités dépendent aussi de la plate-forme d'accélération choisie.

Il faut également noter la présence des fonctions non-linéaires suivantes :
— division ;
— racine carrée ;
— logarithme népérien ;
— arctangente.

L'exécution de ces fonctions est loin d'être aussi immédiate qu'une addition ou une multiplication et nécessite des méthodes dédiées. Par exemple, les calculatrices usuelles peuvent utiliser la méthode *CORDIC* pour effectuer ces fonctions [59]. Il existe notamment un symposium international, *ARITH*, qui se concentre sur ces problématiques de calculs arithmétiques. [60].

2.2.3 Bilan de la présentation de la fonction à accélérer

La fonction étudiée est celle qui demande le plus de temps de calcul lors de la conception de la matrice opérateur Calderon symétrique présentée dans le Chapitre 1. Elle est en effet appelée pour le calcul de chaque cœfficient de cette matrice et son impact dépend donc directement de la taille de cette dernière, et donc de la résolution souhaitée. Cette fonction effectue une intégration en 3 dimensions d'une surface triangulaire par rapport à un point d'observation dans l'espace et est composée de deux grandes étapes : une première phase de calculs de distances et de projections puis le calcul effectif de l'intégrale. Cette fonction a un potentiel d'accélération évident. En effet, beaucoup de calculs à l'intérieur de celle-ci peuvent être parallélisés. De plus, les différents appels à la fonction sont indépendants les uns des autres, ce qui permet la parallélisation de différents appels afin de maximiser le nombre de calculs effectués en un temps donné. En ce qui concerne les difficultés identifiées, les fonctions non-linéaires présentes dans l'algorithme font depuis longtemps l'objet de recherches pour diminuer leur impact sur le temps d'exécution. Il faut donc porter une attention particulière à l'implantation de ces fonctions afin de ne pas trop perturber les performances finales de la solution d'accélération retenue.

Un autre paramètre à prendre en compte et le compromis possible entre le facteur d'accélération et la précision. En effet, selon l'équipe partenaire travaillant sur l'aspect algorithmique, une précision relative de 10^{-2} est nécessaire afin d'assurer la bonne convergence de la résolution du problème direct. Cela permet donc de réaliser les calculs sur des données représentées en virgule fixe, ce qui est plus rapide que de les réaliser sur des flottants.

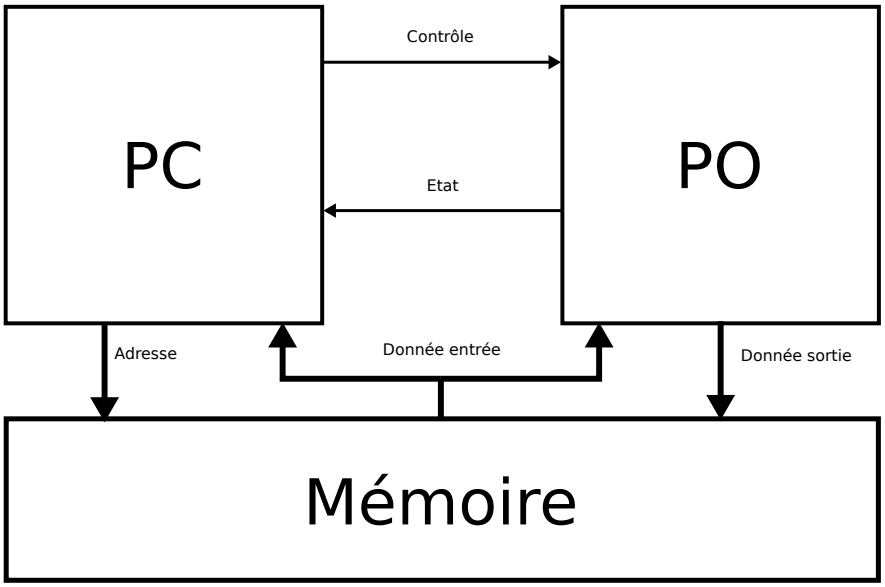


FIGURE 2.7 – Représentation de l'organisation d'un processeur généraliste.

2.3 Perspectives d'accélération

Les performances d'un algorithme selon les critères de précision, débit ou encore latence dépendent en partie du matériel sur lequel l'implantation a été faite. Pour optimiser ces résultats il faut adapter l'implantation à la plate-forme choisie. Dans cette section, différentes cibles matérielles pour un accélérateur sont présentées puis comparées entre elles afin de réaliser un choix pertinent pour accélérer l'algorithme présenté en Section 2.2. Les résultats d'un premier test d'accélération sont par la suite indiqués afin de donner une idée de l'accélération possible pour l'algorithme visé.

2.3.1 Choix de la cible matérielle pour accélération

La manière la plus simple de tester un algorithme est de l'exécuter sur le processeur généraliste, composant central de tout ordinateur. Si cela permet de vérifier facilement la fonctionnalité du code à tester, les performances, notamment le temps d'exécution, sont loin d'être optimales au premier essai. Dans la suite de cette Section, le fonctionnement d'un processeur généraliste est décrit pour mettre en avant les éléments mis en œuvre pour accélérer l'exécution d'un code. La même chose sera faite par la suite pour d'autres plate-formes matérielles : le processeur graphique, le FPGA ou encore l'ASIC.

2.3.1.1 Processeur généraliste

Le processeur généraliste est l'élément clé d'un ordinateur, cadencé par une horloge pouvant atteindre plusieurs gigahertz. La Figure 2.7 permet d'illustrer de manière simple son fonctionnement. Un processeur est composé de trois composants principaux une Partie Contrôle (PC), une Partie Opérative (PO) et une mémoire. La PC récupère dans la mémoire les instructions à exécuter puis contrôle la PO en conséquence. Cette PO, qui peut aussi être appelée Unité Arithmétique et Logique (UAL), récupère

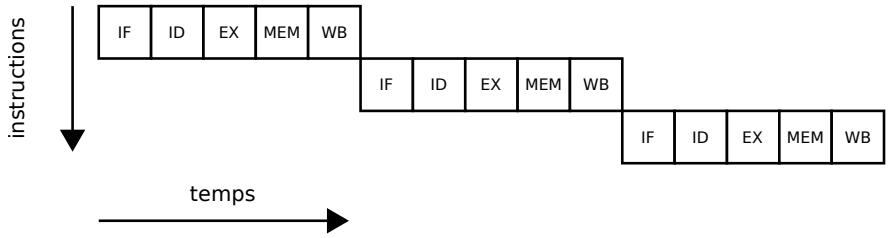


FIGURE 2.8 – Séquençage des instructions dans un processeur sans pipeline.

dans les mémoires les données à traiter puis y mémorise le résultat de l'instruction. L'architecture de l'UAL dépend donc du jeu d'instructions pour lequel le processeur a été conçu. Le fonctionnement peut donc être découpé en tâches distinctes. L'exemple le plus connu est celui du processeur *Reduced Instruction Set Computer* (RISC) qui considère 5 étapes, comme présenté en Figure 2.8 :

- lecture de l'instruction (ou *Instruction Fetch* (IF)) : la prochaine instruction à exécuter est chargée par la PC depuis la mémoire ;
- décodage de l'instruction (ou *Instruction Decode* (ID)) : l'instruction est analysée par la PC pour déterminer ce qui est demandé ;
- exécution de l'instruction (ou *Execute* (Ex)) : l'UAL exécute l'instruction ;
- Accès à la mémoire (ou *Memory* (Mem)) : la PC gère un transfert entre la mémoire et un registre ;
- Rangement du résultat (ou *Write Back* (WB)) : Le résultat issu de la PO est rangé dans le registre cible.

Dans l'hypothèse où chacune de ces étapes s'effectue en un seul cycle d'horloge, il peut donc être considéré que le processeur effectue 0,2 instruction par cycle puisqu'une instruction est totalement réalisée tous les 5 cycles. Pour réduire le temps d'exécution des instructions, la piste évidente est de réduire la durée d'un cycle d'horloge, et donc d'augmenter la fréquence du processeur. Cette stratégie a été utilisée pour passer des 740 kHz du processeur Intel 4004, datant de 1971 [61], aux fréquences actuelles. Pourtant la croissance de cette fréquence de fonctionnement a considérablement ralenti ces dernières années, comme présenté en Figure 2.9. Même si la finesse de gravure des puces continue de s'améliorer au fil des années, la fréquence s'est en effet stabilisée depuis le début des années 2000. Cela s'explique notamment par des problèmes de dissipation de chaleur au sein de la puce [62]. Il est donc normal d'observer la même tendance pour la puissance consommée. En contrepartie, le nombre de transistors a continué sa progression avec l'augmentation du nombre de cœurs.

La solution d'augmenter la fréquence de fonctionnement afin d'augmenter le nombre d'instructions réalisées en un temps donné n'est donc pas suffisante pour maintenir un rythme d'amélioration des performances constant. D'autres solutions permettent d'améliorer le temps d'exécution des instructions au sein d'un processeur. La première est d'utiliser une architecture pipelinée. Cela permet d'avoir plusieurs instructions simultanément dans le processeur, comme illustré en Figure 2.10. En comparaison avec l'exemple de la Figure 2.8, les trois instructions représentées sont exécutées en 7 cycles au lieu de 15. L'augmentation des performances ne passe donc pas par une réduction du temps de traitement d'une instruction mais par une parallélisation. Le nombre d'instructions traitées par le processeur en un temps donné pourrait donc théoriquement

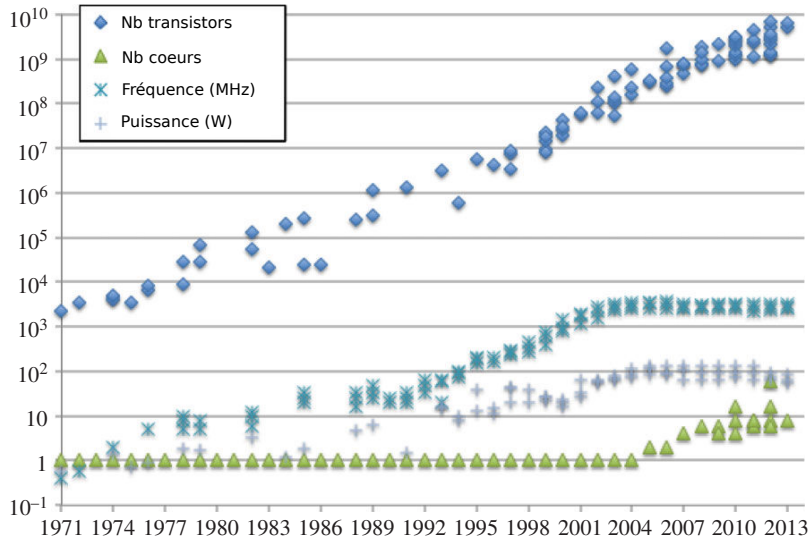


FIGURE 2.9 – Évolution du nombre de transistors, de cœurs, de la fréquence et de la puissance des processeurs de 1971 à 2013. [9]

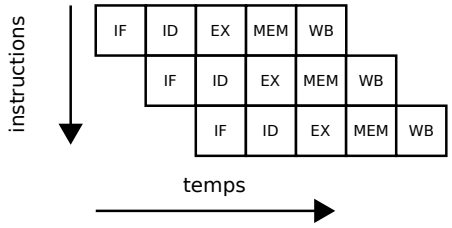


FIGURE 2.10 – Séquençage des instructions dans un processeur avec pipeline.

être multiplié par la valeur de la profondeur du pipeline, qui correspond au nombre d'instructions simultanément dans le processeur. Cependant, ce facteur d'accélération est rarement atteint. En effet, le fonctionnement du pipeline peut être perturbé par des aléas qui peuvent être de plusieurs natures :

— aléa structurel, quand deux instructions à des niveaux différents du pipeline ont besoin de la même ressource ;
— aléa de données, quand une instruction a besoin d'une donnée qui n'est pas encore calculée par une autre instruction en cours ;
— aléa de contrôle, quand un branchement est effectué. Les instructions qui suivent l'instruction de branchement dans le pipeline ne doivent pas être exécutées.

La gestion des deux premiers cas d'aléas s'effectue via la création de bulles. Cela correspond à mettre en attente l'instruction en défaut le temps que le problème se résolve de lui-même. Un exemple de bulle est montré en Figure 2.11, où l'exécution de la troisième instruction nécessite le *Write Back* de la deuxième et interrompt donc la suite du pipeline.

L'utilisation d'un pipeline pour un processeur permet donc de réaliser plusieurs instructions en même temps et ainsi améliorer le temps d'exécution d'un algorithme donné. Si augmenter la profondeur du pipeline permet théoriquement d'améliorer encore plus les résultats, cela augmente également la probabilité d'apparition d'aléas,

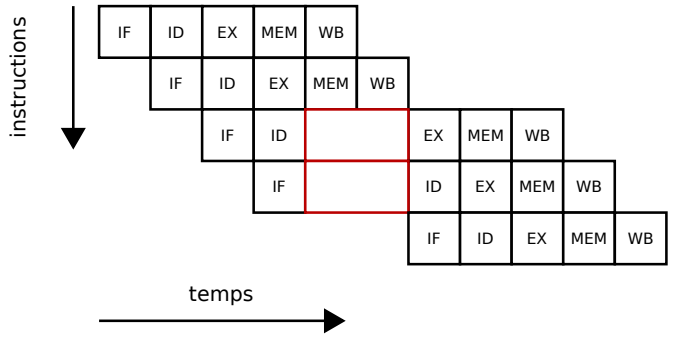


FIGURE 2.11 – Exemple de bulle dans un pipeline.

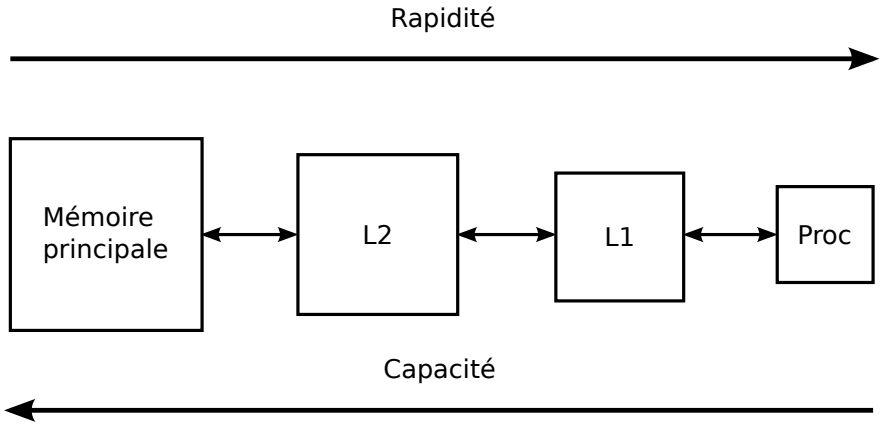


FIGURE 2.12 – Représentation de la hiérarchisation de la mémoire.

qui nécessitent l'interruption du flux des instructions. L'une des tâches d'un compilateur est donc d'ordonner judicieusement les instructions afin de limiter au maximum l'apparition d'aléas.

Dans les faits, les différentes étapes de l'exécution d'une instruction se déroulent rarement en un seul cycle. Le nombre de cycles nécessaires peut même être variable. C'est notamment le cas pour les accès mémoire. En effet, avec la complexification des systèmes, l'organisation de la mémoire a dû évoluer afin que les accès mémoire ne soient pas trop pénalisants pour les performances du processeur. Il existe donc une hiérarchisation de la mémoire en plusieurs niveaux de cache. La Figure 2.12 en illustre un exemple. Pour passer de la mémoire principale à un registre du processeur, les données passent par des mémoires caches nommées L2 et L1. Plus les mémoires sont proches du processeur et plus elles sont rapides d'accès et comme les mémoires les plus rapides sont aussi les plus chères, la capacité de chaque bloc augmente au fur et à mesure que l'on s'éloigne du processeur. Quand le processeur a besoin d'une donnée, il envoie donc une requête au cache L1. Si la donnée ne s'y trouve pas, il y a un échec d'accès et il faut alors faire une requête au niveau suivant et cette logique continue jusqu'à la mémoire principale. Cela explique l'aspect variable du nombre de cycles d'un accès mémoire. Il est donc important, à la génération du micro-code exécuté par le processeur, de bien ranger les données dans la mémoire afin d'éviter le plus d'échecs d'accès possible. Dans le cas de la fonction d'intégration considérée, la forte dépendance des données empêche l'utilisation optimisée du pipeline du processeur. En effet, il est for-

tement probable, au vu de l'algorithme, d'avoir des opérations nécessitant le résultat de l'opération qui précède dans le pipeline, ce qui cause donc une bulle. Le processeur n'est donc pas la structure la plus optimisée pour exécutée ce code.

En plus des temps d'accès mémoire variable, la latence en nombre de cycles varie selon les instructions [63]. Le pipeline est par conséquent complexe à gérer si l'on souhaite optimiser le ratio d'utilisation de l'UAL. Il faut noter que l'UAL exécute des opérations sur des données de type entiers. Quand des calculs sur des nombres flottants sont nécessaires, le processeur utilise son Unité de calcul en Virgule Flottante (UVF), aussi connue sous le nom de *Floating-Point Unit* (FPU). Il s'agit d'une unité spécialement conçu pour les calculs flottants, connus pour être plus complexes que les calculs sur des entiers. Tous ces éléments font qu'au final, il est possible que l'UAL ne soit pas utilisée durant toute la durée de fonctionnement du processeur. Le *multi-threading* est une technique qui permet d'optimiser ce temps d'utilisation [64]. Le principe est d'exécuter plusieurs fils d'instructions (ou *thread*) sur les ressources d'une seule unité de calcul. Chaque *thread* dispose toutefois de son propre compteur de programme et de ses propres registres. Le principe est d'entrelacer les instructions des différents *threads* afin d'optimiser l'utilisation du processeur. Par exemple, si une instruction est confrontée à un défaut de cache et doit donc attendre un accès mémoire, l'instruction d'un autre fil d'instruction peut profiter de ce temps pour accéder à l'UAL. Il est donc important que tous les *threads* soient indépendants les uns des autres afin de ne pas avoir de conflits de données. Cela demande donc une structure de programme particulière avec une bonne gestion de la jonction de *threads*. Comme la mémoire est partagée, il y a aussi un risque de multiplier les défauts de cache si les données sont mal gérées.

Un autre moyen d'augmenter la capacité de calcul est d'utiliser une architecture multi-cœurs. Comme son nom l'indique cela consiste à totalement dupliquer le cœur de calcul. Chaque cœur a donc un fonctionnement pipeliné et peut supporter le *multi-threading*. Deux stratégies d'agencement de la mémoire sont alors envisageables. Il est en effet possible d'utiliser des caches dédiés à chaque cœur ou des caches partagés, chacun ayant ses avantages et inconvénients [65].

Les caches dédiés permettent d'assurer que l'exécution d'un programme sur un cœur de calcul ne pollue pas celle se déroulant sur un autre cœur. De plus, les temps d'accès d'un tel cache est généralement plus faible qu'un cache partagé. Cela s'explique surtout par le fait qu'il ne répond aux besoins que d'un seul cœur de calcul. Toutefois, cette solution ralentit logiquement les échanges de données entre deux cœurs puisqu'il faut monter plus haut dans la hiérarchie mémoire pour trouver une ressource commune. Cela implique donc une duplication de la donnée concernée dans le cache dédié de chaque cœur ayant besoin de celle-ci.

L'utilisation de caches partagés permettent donc de fluidifier le transfert de données entre les cœurs de calcul tout en limitant la duplication des données. Cela nécessite néanmoins une taille de cache et une bande passante plus élevées, ainsi que de permettre à plusieurs cœurs d'accéder de manière simultanée au cache. Ceci augmente en contrepartie la latence de la réponse et la demande énergétique.

Il faut donc disposer les mémoires caches de manière réfléchie afin d'optimiser les performances. Ainsi, les caches de type L1 sont généralement dédiés alors que les caches L2 sont partagés. La structure multi-cœurs est notamment bénéfique pour l'exécution de codes dotés d'un haut degré de parallélisme. Le facteur d'accélération

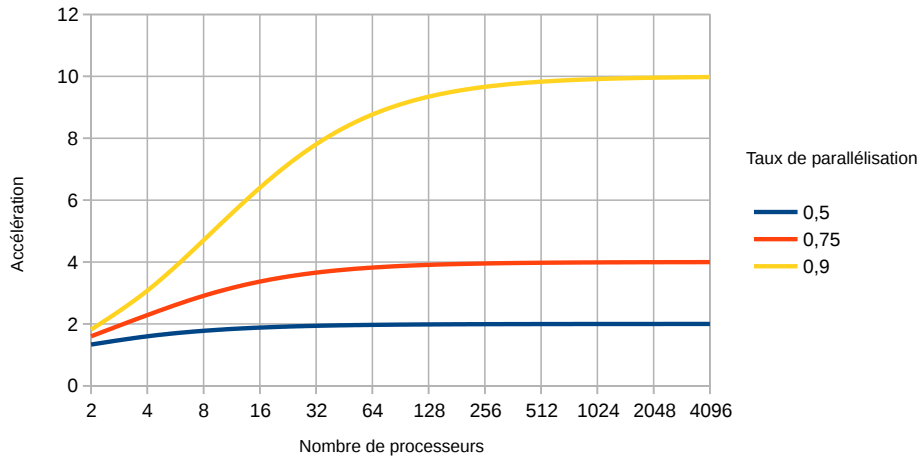


FIGURE 2.13 – Évolution de l'accérération d'exécution en fonction du nombre de processeurs pour plusieurs taux de parallélisation.

atteignable d'une architecture multi-cœurs par rappport à une architecture simple cœur a été théorisée par Amdahl en 1967 [66]. Le facteur théorique dépend du nombre de cœurs, ainsi que du taux de parallélisation du code à exécuter selon la formule suivante :

$$F_a = \frac{1}{1 - t_p + \frac{t_p}{N_c}}, \tag{2.25}$$

avec F_a le facteur d'accélération atteignable, t_p le taux de parallélisation et N_c le nombre de cœurs présents dans l'architecture.

L'accélération atteignable de l'exécution d'un code est donc limité par la partie qui ne peut pas être parallélisée. Ainsi, dans l'exemple d'un code parallélisable à 75%, le temps d'exécution ne pourra pas être divisé par plus de 4, peu importe le nombre de cœurs dans l'architecture, Figure 2.13.

Dans le cas de la fonction à accélérer présentée en Section 2.2, il faut donc exécuter un appel complet par cœur afin de profiter au mieux de la parallélisation. Chaque cœur garde cependant l'impossibilité d'utiliser le pipeline de manière optimale. Pour un grand nombre d'appels de la fonction successifs, l'utilisation d'une architecture multi-cœurs peut donc diviser le temps d'exécution au mieux par le nombre de cœurs du processeur.

2.3.1.2 Processeur graphique

Le processeur graphique, plus connu sous le nom de *Graphics Processing Unit* (GPU), est un composant électronique qui pousse encore plus loin la notion d'architecture multi-cœurs. En effet, il peut comporter plusieurs milliers de cœurs, même si ceux-ci sont plus simples qu'un cœur de processeur généraliste et fonctionnent à une fréquence moins élevée. A l'origine, cet élément a été conçu pour exécuter des fonctions de calculs d'images pour de l'affichage vidéo. Ces fonctions doivent donc être exécutées en grand nombre de fois en un temps réduit et les besoins en puissance de calcul ont augmenté au fil des améliorations de la qualité d'affichage [67]. A l'origine,

les GPU étaient spécialisés au point d'être composés d'unités de calcul non-flexibles et dédiées à un seul calcul. Par la suite, les cœurs de calcul se sont rapprochés de la structure des processeurs généralistes afin d'offrir plus de flexibilité, Les données sont également passées du format entier au format flottant pour répondre au besoin de précision.

Cette évolution profite à la communauté scientifique qui y trouve une solution intéressante pour accélérer le temps d'exécution d'algorithmes massivement parallèles. De plus, le prix de ce type de matériel reste abordable, notamment grâce à sa large diffusion au grand public par le biais du domaine du jeu vidéo. Il n'est d'ailleurs pas rare que la communauté scientifique examine les nouvelles consoles afin de voir s'il est intéressant d'exploiter leurs performances de calculs [68, 69]. Les GPU ont ainsi été détournés de leur vocation première afin de réaliser des calculs à la place d'un processeur générique. Cet usage peut se retrouver sous le nom de *General-Purpose computings on Graphics Processing Unit* (GPGPU) et est utilisé dans de nombreux domaines comme les sciences computationnelles, la finance, l'analyse d'images, ou plus récemment le minage de crypto-monnaie. Cela a donc changé le cheminement des données. Quand le rôle du GPU était uniquement de concevoir les images à afficher sur un écran, les données étaient reçues sous la commande du processeur central et le résultat était directement affiché sur l'écran via l'interface vidéo du GPU. Dans le cas du GPGPU, les résultats sont renvoyés en mémoire pour être analysés ou utilisés par la suite par exemple par le processeur central. Il faut donc s'assurer que le gain en temps d'exécution obtenu par l'usage de la parallélisation est supérieur au temps nécessaire pour transférer les données.

Les constructeurs de GPU ont également pris conscience de cette nouvelle utilisation et propose des composants spécialement dédiés au calcul parallèle pour supercalculateurs. Par exemple, la Nvidia Titan V dispose de 5 120 unités de calcul fonctionnant à une fréquence de base de 1,2 GHz, mais elle propose également des cœurs de calcul spécifiques pour les applications d'intelligence artificielle [70]. Des nouvelles interfaces de programmation ont également été développées afin de créer des codes dédiés à être exécutés sur GPU. Ces solutions se basent sur les langages de programmation connus (C, C++, Fortran...) auxquels il faut ajouter des mots clés pour profiter de la parallélisation [71]. La mise en place du calcul parallèle sur GPU ne demande donc que peu de connaissances supplémentaires par rapport à une solution fonctionnant uniquement sur un processeur.

Cette solution d'accélération matérielle sur GPU est donc intéressante pour exécuter du code totalement parallèle, dans la logique de la loi d'Amdhal. Tous les cœurs de calcul effectuent alors la même suite d'instructions sur des données d'entrées différentes. Le calcul s'effectue en 3 phases. Toutes les données nécessaires sont tout d'abord envoyées de la mémoire de l'ordinateur vers le GPU. Tous les cœurs concernés exécutent ensuite la suite d'instructions à accélérer avant que tous les résultats soient renvoyés à la mémoire. Cette méthode implique donc une famine du processeur central plus longue qu'une solution où les données sont traitées dès leur transfert et non pas quand tous les jeux de données sont transmis. Pour accélérer au mieux la fonction d'intégration en trois dimensions présentée en Section 2.2, il faut exécuter un appel de fonction par cœur. Le facteur d'accélération dépend alors directement du nombre de cœurs utilisés, même si ceux-ci ne sont pas optimisés pour l'application visée. Une

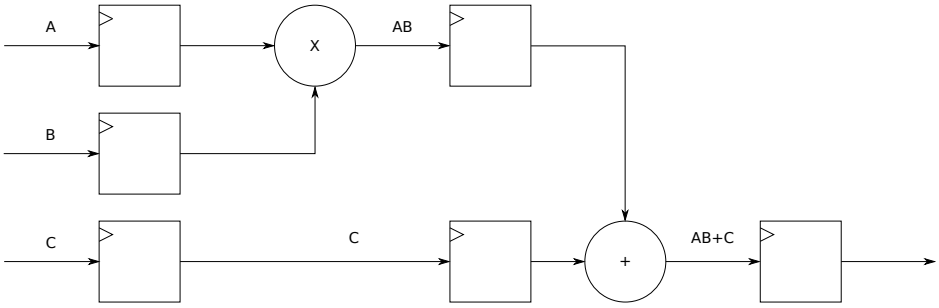


FIGURE 2.14 – Exemple d'un pipeline dans un circuit spécialisé.

fonction d'intégration différente de celle analysée dans ce Chapitre a déjà été accéléré par GPU [72]. Le facteur d'accélération de 2,5 mesuré entre un processeur Intel Xeon E5-2698 v3, composé de 32 cœurs, et un GPU Nvidia Tesla K40 qui contient 2888 cœurs de calcul. Cependant, les GPU les plus puissants actuellement souffrent de la problématique de dissipation de chaleur et nécessitent la présence de radiateurs et ventilateurs. Cela élimine donc cette possibilité pour la perspective d'un système de calcul d'ICM portable.

2.3.1.3 Circuit spécialisé

Un autre solution pour accélérer l'exécution d'une fonction est de concevoir une architecture dédiée, autrement appelé ASIC. Cela est plus complexe que d'utiliser des composants génériques comme un processeur ou un GPU, mais un plus haut niveau d'optimisation est atteignable. Cela permet de mettre en place l'architecture idéale pour tout type de contraintes. Il est par exemple possible de concevoir une partie logique permettant de réutiliser un bloc fonctionnel en cas de contraintes de ressources. Au contraire, si un débit élevé est recherché, il est plus intéressant d'établir un pipeline optimisé. Cette dernière approche est particulièrement intéressante pour l'accélération matérielle de la fonction présentée en Section 2.2. En effet, une architecture pipelinée dédiée à l'application permet d'éviter tous les risques d'aléas de pipeline dont souffre un processeur généraliste. En effet, les données ne sont pas renvoyées dans une mémoire globale dans ce type d'architecture mais des points de mémorisation au plus près des parties calculatoires sont utilisés. Dans l'exemple en Figure 2.14, les entrées A, B et C sont acheminées au même instant dans la portion de pipeline chargée d'exécuter l'opération $AB+C$ et stockées dans les registres dédiés. Sur le cycle d'horloge suivant un registre va récupérer le résultat de la multiplication des deux premières opérandes alors que la donnée C va seulement être transmise à un autre registre. Un dernier registre permet de stocker au cycle suivant le résultat final, qui peut être par la suite utilisé pour d'autres opérations. Cette structure permet donc de prendre en compte un nouveau jeu de données à chaque cycle d'horloge sans risque d'aléas. L'architecture repose donc d'une part sur les registres pour assurer la bonne séquencialité des opérations ainsi que la montée en fréquence de fonctionnement, et d'autre part les éléments exécutant les opérations logiques et arithmétiques.

Si la conception d'un ASIC permet d'avoir un contrôle total sur l'architecture, et donc sur la performance, cela demande un coût de développement important. La production en petite série est également onéreuse, par exemple, pour la technologie 65

nm CMOS de STMicroelectronics, il faut compter 6000€/mm^2 [73]. Il est donc préférable de faire toutes les vérifications nécessaires sur le circuit créé avant de l'envoyer en fonderie. Pour la conception de circuits numériques, un langage de description de matériel (ou *Hardware Description Language* (HDL)) est souvent utilisé [74]. Ce type de langage est utilisé pour réaliser une description du circuit au niveau des registres (ou description *Register-Transfer Level* (RTL) et est également utile pour programmer des circuits configurables, comme un FPGA, qui permettent de faire des prototypes plus facilement. En première approche, un FPGA peut être considéré comme une matrice d'éléments configurables qui contiennent des registres comme points de mémorisation, ainsi que des tables de correspondance (ou *Look-Up Table* (LUT)) pour assurer les opérations logiques et arithmétiques. Ces éléments sont reliés entre eux par un routage lui-même configurable. Il est donc possible de mettre en place n'importe quelle fonction séquentielle ou combinatoire sur un FPGA, et de changer la configuration de cet élément à volonté. Tout ceci fait de cette plate-forme une solution de choix pour prototyper un circuit numérique avant d'en envisager une conception ASIC. Cette qualité a même été dépassée puisque le FPGA est maintenant utilisé pour implanter des systèmes complets, notamment grâce à des puces liant directement un processeur générique à un FPGA [75, 76]. Cette technologie mélangeant performance et flexibilité est désormais un élément clé pour les services d'entreprises comme Amazon ou Microsoft [77]. De plus la capacité de calcul des FPGA en font des outils permettant de traiter des flux de données de plusieurs terabits [78]. Une grande gamme de cartes de développement munies de FPGA est proposée sur le marché couvrant ainsi de nombreux domaines d'application (communications, traitement du signal, intelligence artificielle...) [79].

Concevoir un circuit dédié pour la fonction d'intégration présentée en Section 2.2 est donc une solution intéressante. Cela permet d'avoir un pipeline d'exécution optimisé qui peut prendre en compte un nouveau jeu d'entrée à chaque cycle d'horloge. Le nombre d'appels de la fonction exécutés en parallèle correspond alors à la profondeur du pipeline. Les sorties de l'algorithme implanté sont également délivrées cycle par cycle. L'utilisation d'un circuit spécialemnt conçu s'intègre donc parfaitement dans un système global où il faut limiter le temps d'inactivité des différents composants. Les architectures proposées peuvent être mises en place sur FPGA, ce qui permet de tester la fonctionnalité tout en ayant des performances intéressantes. Si ces dernières ne suffisent pas, l'architecture validée sur FPGA peut être conçue en ASIC pour augmenter la fréquence de fonctionnement. En effet, la fréquence atteignable sur un FPGA est majoritairement limitée par l'interconnexion entre les différents éléments configurables. Fondre un circuit ASIC permet donc de rapprocher au mieux les différents éléments de l'architecture et ainsi de diminuer le chemin critique du circuit.

2.3.1.4 Comparaison des plate-formes possibles

Il est évident qu'un processeur généraliste seul est insuffisant pour traiter un grand nombre de données en un temps réduit. Même si son architecture s'est complexifiée au fil des années avec l'évolution de son pipeline et la multiplication des cœurs, sa généricité reste un frein à la montée en performance. Ceci explique l'essort du calcul parallèle que ce soit sur un GPU, un FPGA ou même un ASIC. Ces solutions né-

cessitent que le processeur central gère le transfert des données entre la mémoire du système et l'accélérateur matériel. Plusieurs paramètres entrent en jeu pour comparer les différentes solutions. En ce qui concerne le temps de conception de la solution, il est plus facile de modifier le code d'un algorithme pour le rendre compatible avec une exécution sur un GPU que d'en réaliser une description RTL. Il faut donc être certain que les performances atteignables justifient l'engagement dans un développement plus long et coûteux. Cela explique que la communauté scientifique a déjà réalisé des comparaisons entre les différentes plate-formes. Ces comparaisons concernent plusieurs domaines comme le traitement d'images [80] ou les réseaux de neurones [81]. Ces études ont conclu que les solutions FPGA et ASIC offrent de meilleures capacités de calcul que sur GPU et pour un coût énergétique moins élevé.

Dans le cadre des travaux d'accélération de calcul de la matrice opérateur dans l'optique d'une interface cerveau-machine, il a été choisi de travailler sur une implantation FPGA, en gardant la possibilité de réaliser un ASIC à terme. Il est vrai que cette solution demande un temps de dévelopement conséquent, mais il est possible de produire un code de description matérielle testable sur plusieurs plate-formes, peu importe leurs constructeurs. Pour les solutions sur GPU, passer d'un constructeur à un autre, peut demander de changer le code de l'algorithme à accélérer afin de l'adapter à la nouvelle plate-forme. De plus, un architecture pipelinée peut être définie sur FPGA ou ASIC de façon à limiter le temps d'inactivité du processeur central et ainsi entamer plus tôt les calculs qui ont besoin du résultat d'intégration. Cette solution matérielle permet aussi de faire disparaitre tous les problèmes de tranferts de données entre la mémoire et l'unité de calcul, ce qui lui permet d'être plus performante sur le plan énergétique.

2.4 Conclusion

La fonction visée pour une accélération matérielle est la fonction clé pour la conception de la matrice opérateur qui sert à résoudre le problème direct de l'EEG. Une accélération matérielle de cette fonction est donc pertinente pour tout domaine de recherche s'intéressant à l'imagerie cérébrale. Le rôle de cette fonction est d'effectuer une intégration sur un triangle à partir d'un point d'observation. La fonction génère une sortie scalaire à partir de 5 entrées sur 3 dimensions et est composée de 2 grandes parties : une première étape de calculs de distances et de projections puis une dernière étape où l'intégrale est effectivement calculée. Il s'agit donc d'un enchaînement de calculs qui présente des fonctions non-linéaires très gourmandes en temps d'exécution comme la division, la racine carrée, le logarithme népérien ou l'arctangente. Chaque appel de la fonction est indépendant, ce qui offre une perspective d'accélération intéressante.

Un processeur généraliste permet aujourd'hui d'exécuter un grand nombre d'instructions différentes. Cette diversité d'actions va néanmoins à l'encontre de la performance. Le calcul parallèle offre une solution pertinente quand une grande capacité de calcul est demandée. Le moyen le plus évident de mettre en place une telle solution est d'utiliser un GPU, élément qui contient des milliers de cœurs de calculs et qui est de plus en plus utilisé par la communauté scientifique, notamment grâce à sa facilité d'utilisation. Une autre piste à exploiter est de concevoir une architecture électronique dédiée à l'algorithme à accélérer. Cette solution est certes plus longue à mettre en

place, mais l'émergence du FPGA, élément reconfigurable, a fluidifié le prototypage de ce genre de circuits et offre aujourd'hui des performances défiant celles d'un GPU tout en consommant moins d'énergie. La solution peut être encore plus spécialisée en concevant une puce ASIC.

La solution de concevoir un circuit dédié a été choisie pour mettre en place l'accélération de la fonction d'intégration étudiée. En effet, cela permet une parallélisation optimale des opérations nécessaires et le pipeline possible permet de lancer une nouvelle exécution à chaque cycle d'horloge. Une fois le code de description matérielle conçu et validé, il est simple de l'implanter sur différentes plate-forme afin par exemple de choisir le FPGA optimal pour l'architecture tout en réduisant au mieux le coût de mise en place de la solution. De plus, plusieurs critères peuvent être améliorés en passant d'une solution FPGA à une solution ASIC : rapidité d'exécution, efficacité énergétique, encombrement.

Cela correspond également à une volonté d'exploration. En effet, la communauté de l'électromagnétique computationelle a d'ores et déjà la capacité d'utiliser un GPU en tant qu'accélérateur assez facilement grâce aux outils mis en place par les constructeurs. Concevoir un circuit dédié à la fonction d'intégration étudiée demande un tout autre domaine de compétences qui pourrait donc s'avérer totalement complémentaire à l'étude mathématique des problèmes électromagnétiques. Les travaux de cette thèse permettent donc d'explorer la piste de l'accélération matérielle pour des communautés qui ne le considèrent pas encore comme un atout de productivité.

Chapitre 3

Méthodes de mise en œuvre d'opérateurs

3.1 Introduction

Dans le Chapitre 2 la fonction d'intégration en 3 dimensions qui doit être accélérée matériellement a été décrite. La possibilité de paralléliser une partie des calculs et de créer un pipeline de traitement a permis de conclure que la réalisation d'un circuit dédié est la meilleure solution pour mettre en place un accélérateur matériel.

Notre objectif à court terme est de prototyper une solution d'accélération pour le problème direct de l'EEG sur plate-forme FPGA. Dans un plus long terme, l'accélérateur conçu doit être intégré dans un circuit embarqué, où un ASIC peut être considéré. Il est alors important de choisir une méthode de conception adaptée aux 2 plate-formes et de définir comment l'accélérateur s'intègre dans un système complet.

Dans ce Chapitre, différentes méthodes de conception d'un circuit dédié sont tout d'abord passées en revue. Une fois la méthode de conception choisie, la problématique d'intégration d'un accélérateur dans un système est abordée.

3.2 Création d'opérateurs

La conception d'un circuit dédié consiste à spécifier les actions réaliser sur les entrées pour générer les sorties souhaitées. Il faut alors décrire les différents calculs nécessaires, les éventuelles opérations logiques mais également la synchronicité entre tous les signaux gérés. Ce niveau d'abstraction correspond à la description RTL. Cette description peut être écrite en langage de description matériel, ou HDL, comme le VHDL ou le Verilog. Le code décrivant l'architecture est analysé par un outil d'inférence qui va établir la correspondance entre la description matérielle et des composants de base présents dans des librairies. Cela permet de générer lors de l'étape de synthèse un ensemble de composants et leur interconnexion, appelé *netlist*. Cet élément permet de réaliser ensuite l'étape de placement et routage qui consite à disposer les composants les uns par rapport aux autres de sorte à limiter au mieux les distances d'interconnexion et ainsi obtenir le chemin critique le moins élevé possible. Ce processus est valide à la fois pour une plate-forme FPGA mais aussi pour un ASIC. Il est donc

important de pourvoir générer un code HDL adapté à chaque flot de conception afin de porter facilement un prototype réalisé sur FPGA en technologie ASIC.

La description matérielle par du code HDL d'une architecture simple est réalisable très rapidement mais la complexité de la tâche peut rapidement croître quand un grand nombre de signaux sont à gérer, ou quand plusieurs machines d'états doivent communiquer entre elles. La manière dont l'architecture est décrite joue directement sur les performances finales : ressources utilisées, latence, fréquence atteignable, précision... Cela explique l'apparition de solutions permettant d'élever le niveau d'abstraction nécessaire pour concevoir une architecture numérique. Ces outils permettent de générer rapidement des blocs fonctionnels qui peuvent être utilisés dans des architectures plus complexes. Ces différents moyens sont présentés plus en détail pour étudier leur impact par rapport à une description au niveau RTL d'un code HDL.

3.2.1 Synthèse de haut niveau

Le but affiché des outils de synthèse de haut niveau (ou HLS) est de diminuer le temps de conception d'architectures constituées de plusieurs opérateurs. Cela permet d'avoir un très bon ratio temps de développement/performances. Le principe est de proposer un flot de création d'une description RTL à partir d'un langage de haut niveau d'abstraction, c'est à dire un code logiciel, le plus souvent écrit en C/C++ ou SystemC [82]. Ce flot de conception est représenté en Figure 3.1. Le code de haut niveau doit être modifié en intégrant les librairies propres à l'outil de HLS utilisé. Ces librairies permettent par exemple de spécifier le format de représentation des données ou encore de donner des directives spécifiques au compilateur, comme le fait de dérouler certaines boucles pour profiter d'un parallélisme matériel. Les librairies utilisées diffèrent selon les outils HLS. La compilation de ce code permet de générer une description RTL dans un langage cible comme par exemple le VHDL ou le Verilog. Ce code peut être simulé, ce qui permet de vérifier le bon comportement de l'architecture générée et de réitérer l'opération sur le code de haut niveau si besoin. En effet, la latence et l'utilisation de ressources d'un opérateur généré par HLS varient grandement selon la manière dont les directives ont été utilisées [83]. Cela demande donc tout de même une connaissance en implantation matérielle pour savoir là où il faut agir. Au final, un bloc RTL fonctionnel est obtenu, appelé module IP. Celui-ci peut être utilisé comme architecture finale ou être intégré dans une architecture plus complexe. Toutefois, lorsque les fonctions visées sont complexes, les performances sont moins bonnes que celles d'un code décrit au niveau RTL [84, 85].

A ce jour, il existe plusieurs outils de synthèse de haut niveau. Les plus connus sont Catapult-C, Vivado HLS et Intel HLS Compiler. Ces 2 derniers ont la particularité d'être les outils proposés par les constructeurs FPGA les plus connus : respectivement Xilinx et Intel FPGA. Le code de description RTL fourni par ces outils est spécialement optimisé pour les FPGA du même constructeur.

Dans le cadre de la thèse présentée dans ce document, il a donc été décidé de ne pas utiliser d'outil HLS pour répondre au mieux à la problématique de la recherche de performance. En effet, concernant l'accélération de la fonction d'intégration sur un triangle, le circuit dédié doit être pipeliné afin de maximiser les appels de la fonction exécutés en même temps. Il est important d'optimiser l'architecture car le moindre

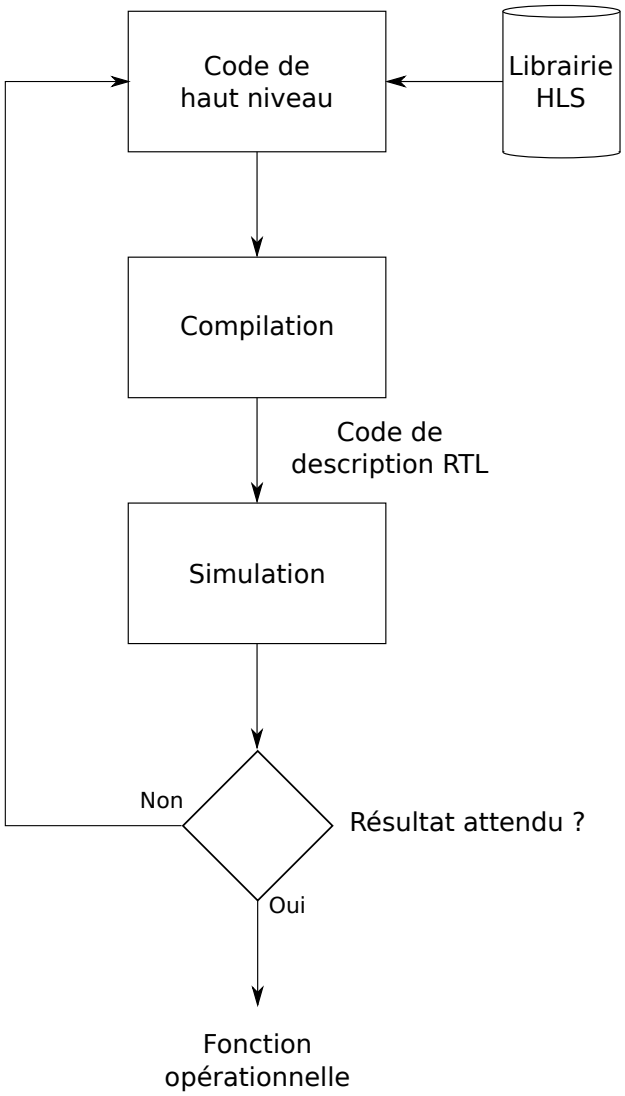


FIGURE 3.1 – Représentation du flot de conception HLS.

gain réalisé en terme de temps de calcul se répercute sur chaque appel de la fonction. L'importance de cette optimisation augmente donc avec le nombre d'exécutions nécessaires de la fonction qui dépend de la résolution de l'imagerie cérébrale étudiée. De plus, il faut garder une maitrise suffisante sur l'architecture afin de pouvoir optimiser l'accélérateur selon la cible finale (FPGA ou ASIC).

3.2.2 Utilisation d'IP constructeur

Un autre moyen d'accélerer la conception de description RTL est d'utiliser des blocs déjà considérés comme fonctionnels, ou IP. Ces blocs répondent généralement à une problématique de base et courante pour la communauté de conception matérielle comme par exemple la multiplication complexe, le filtre à réponse impulsionnelle finie, ou la transformée de Fourier. Ces blocs sont souvent paramétrables afin de répondre à des besoins variés.

Le meilleur moyen d'avoir une IP adaptée à l'architecture du FPGA utilisé est d'en

choisir une issue de la bibliothèque fournie par le constructeur. Une grande partie de ces bibliothèques est fournie avec la licence de la suite logicielle de conception mais certaines IPs peuvent avoir une licence spécifique. C'est notamment le cas pour l'IP de cœur ethernet de Xilinx [86]. Ces blocs sont spécifiques aux plates-formes des concstructeurs concernés. En effet, ils sont considérés comme des boites noires et seul le compilateur du constructeur correspondant peut correctement l'interpréter. Leur utilisation au sein d'une description RTL empêche donc de pouvoir tester cette dernière sur des plates-formes de constructeurs différents, en plus de nullifier la possibilité de conception ASIC du circuit ainsi décrit. Cette option est finalement à rejeter dans le cadre des travaux de thèse présentés afin de garder les perspectives d'exploration souhaitées.

3.2.3 Utilisation d'IP généraliste

Le principe de décrire rapidement une architecture via l'utilisation d'IP a également pris de l'ampleur avec l'émergence de sites internet proposant des blocs fonctionnels libre d'utilisation. L'un des exemples les plus connus est SPIRAL qui génère automatiquement la description matérielle d'algorithmes de traitement du signal comme par exemple la transformation de Fourier rapide ou le filtre à réponse impulsionnelle finie [87]. La génération du code se fait en prenant en compte les différents paramètres renseignés par l'utilisateur. Même si les opérateurs générés sont reconnus pour leur niveau d'optimisation [88], ils ne sont d'aucune aide pour l'accélération matérielle de la fonction d'intégration présentée dans le Chapitre 2 car aucun ne répond à la problématique de l'implantation de fonctions non-linéaires comme la division, la racine carrée, l'arctangente ou le logarithme népérien.

Un autre exemple de bibliothèque généraliste d'IP est OpenCores [89]. Il s'agit d'une plate-forme communautaire de partage de cœurs IP libres de droit. A ce jour, plus de 450 IP sont proposées mais elles peuvent être à différents niveaux de développement. Ainsi, certains projets sont en version alpha alors que d'autres sont stables. Le projet *Floating-Point Cores* (FloPoCo) est une autre initiative qui permet de générer automatiquement une description RTL [90]. Il s'agit plus précisément d'un générateur d'opérateurs qui produit un code HDL à partir de paramètres fournis en entrée. Comme son nom l'indique, les opérateurs proposés concernent majoritairement les opérations en virgule flottante mais on y trouve également des solutions d'approximation de fonctions en virgule fixe. Ces opérateurs utilisent des approximations polynomiales pour approcher des fonctions non-linéaires [91]. Un opérateur dédié au calcul de l'arctangente est même proposé [92]. Cela offre donc une perspective intéressante pour le développement de l'accélérateur de la fonction d'intégration qui nous intéresse.

3.2.4 Bilan sur la conception d'opérateur

La conception d'une description RTL pour FPGA ou ASIC demande un grand investissement de temps et de connaissances afin d'aboutir à un opérateur ayant le comportement souhaité. Pour accélérer la production de solution d'accélération matérielle, plusieurs solutions sont apparues sur le marché. Celle permettant d'accélérer au mieux la production d'opérateur repose sur la synthèse de haut niveau. Cette méthode

permet d'avoir un très bon rapport temps de développement/performances en générant une description matérielle à partir d'un algorithme écrit en langage de haut niveau, par exemple le C/C++. Cependant, cette méthode ne permet pas d'obtenir d'architectures totalement optimisées, ce qui ne correspond pas à notre objectif de proposer la meilleure accélération possible.

Un autre moyen d'accélérer la conception d'un accélérateur matériel est d'utiliser des IPs qui sont des blocs fonctionnels fournis soit par les constructeurs et fournisseurs d'outils logiciels, soit via des initiatives communautaires. Les opérateurs proposés par un constructeur de FPGA ont l'avantage d'être spécialement optimisés pour les FPGA de ce dernier. Ils sont aussi exclusifs au construteur, ce qui limite le nombre de plate-formes possibles pour les tests et empêche toute perspective de conception ASIC.

Les IPs proposées à but communautaire ont l'avantage de fournir un code RTL pouvant être testé sur tout type de plate-forme. Il s'agit donc d'une opportunité intéressante si les blocs proposés répondent aux contraintes de l'accélérateur matériel à concevoir.

Pour la suite des travaux présentés dans ce manuscrit, il a donc était choisi de concevoir l'architecture de l'accélérateur en écrivant le code HDL « à la main », tout en tenant compte des possibilités offertes par les blocs libres de droit proposés.

3.3 Intégration d'un accélérateur dans un système

L'étape d'intégration d'un accélérateur en tant que coprocesseur dans un système plus global est tout aussi importante que la phase de conception du circuit permettant l'accélération. En effet, la transmission des données à traiter ainsi que le contrôle de l'accélérateur doit être le plus transparent possible afin de ne pas détériorer les performances du circuit conçu. Il est donc important d'avoir un lien entre le processeur central et le coprocesseur le plus performant possible. Plusieurs types de lien peuvent être mis en œuvre selon le système. La perspective à court terme des travaux présentés est de s'intégrer dans des stations de calcul pour accélérer les tests cliniques d'ICM. L'utilisation du bus PCIe est alors la piste la plus pertinente. Dans cette Section, l'utilisation de ce bus pour intégrer un accélérateur est tout d'abord détaillée. Ensuite, la problématique d'intégration dans un système embarqué est abordée dans l'optique de la perspective à plus long terme ICM portable.

3.3.1 Intégration via un bus PCIe

Dans un ordinateur, le processeur central est intégré à la carte mère qui permet d'assurer la bonne connexion entre tous les éléments du système. Dans le but d'assurer une certaine évolutivité des systèmes, les cartes mères sont munies de différents ports permettants de connecter des cartes d'extension. Cela permet de pouvoir remplacer certains composants ou tout simplement ajouter des fonctionnalités. Pour ce type de connexion, le standard *Peripheral Component Interconnect* (PCI), créé par Intel au début des années 1990, s'est aujourd'hui imposé. Au fil des années, cette spécification s'est diversifiée en proposant notamment un type de connexion dédié aux applications nécessitant un débit élevé. C'est ainsi qu'est apparu le standard *Accelerated Graphics Port* (AGP) exclusivement dédié aux cartes graphiques. En 2004, une autre évolution

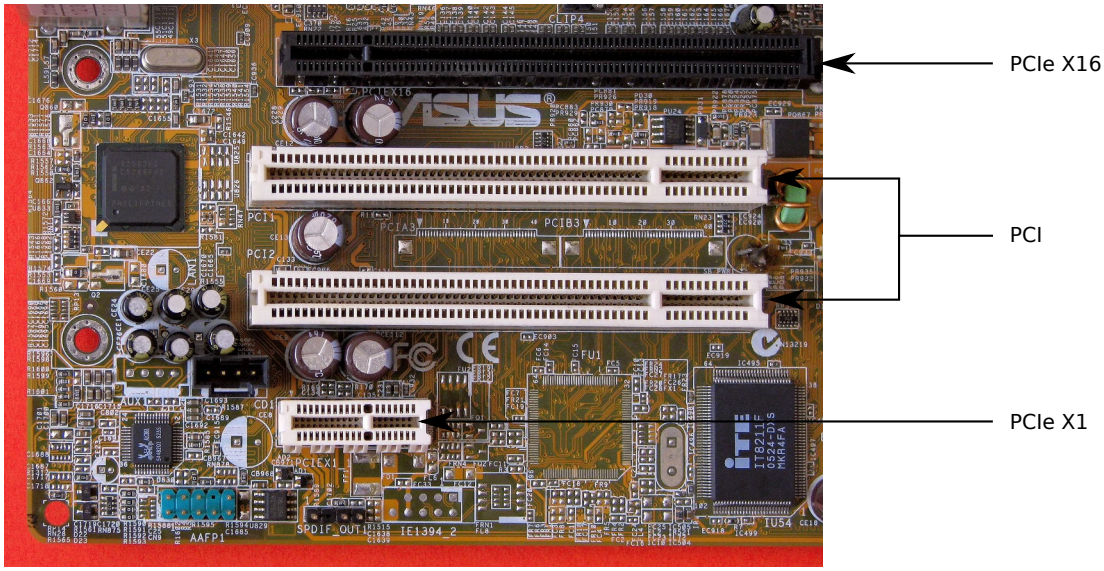


FIGURE 3.2 – Exemple d'un carte mère comprenant des ports PCI et PCIe [10].

du standard PCI, nommée PCIe est apparue dans le but de remplacer les ports PCI et AGP. L'évolution de la spécification est encadrée par le *PCI Special Interest Group* qui regroupe plus de 700 industriels [93].

Un connecteur PCIe peut avoir différentes tailles physiques. Celles-ci sont communiquées de la manière suivante : PCIe x1, PCIe x2, PCIe x4, PCIe x8 ou PCIe x16. Des ports de différentes tailles peuvent se trouver sur une même carte (Figure 3.2). Il est tout à fait possible de connecter une carte ayant un connecteur PCIe d'une certaine dimension sur un port plus grand. L'inverse est cependant physiquement impossible. La valeur qui indique la dimension d'un port correspond au nombre maximal de lignes PCIe pouvant être connectées sur ce port. Une ligne est une communication série point à point composée de 2 paires différentielles, ce qui permet de communiquer dans les 2 sens simultanément. Un port PCIe x16 a donc une bande passante potentielle 2 fois plus élevée qu'un port PCIe x8 de même génération.

Les générations PCIe marquent les différentes évolutions majeures du standard et définissent le débit atteignable par une ligne de communication. Actuellement, la dernière génération spéficiée est la quatrième. La spécification de la cinquième génération est annoncée pour 2019 [94]. Les sauts technologiques s'expliquent surtout par l'augmentation du débit binaire de la communication série. Le passage de la génération 2 à la génération 3 est aussi marquée par l'utilisation d'un codage plus efficace, diminuant ainsi l'impact de la couche protocolaire. Le Tableau 3.1 résume l'évolution du débit au fil des générations [95]. Le passage à une nouvelle génération double donc le débit utile. Par conséquent, un lien PCIe x8 de génération 3 a le même débit qu'un lien x16 de génération 2.

Comme le but premier du standard PCIe était de remplacer, avec de meilleurs performances, le standard AGP, ce type de communication a tout d'abord été utilisé pour les cartes graphiques. Les constructeurs de ces cartes ont donc naturellement assuré la bonne utilisation de ce standard pour leurs produits. L'intégration logicielle des GPU est maintenant quasi-transparente grâce aux interfaces de programmation dédiées.

L'utilisation du bus PCIe n'est pas encore aussi standardisé pour les plates-formes

Tableau 3.1 – Evolution du débit utile d'une ligne PCIe.

Génération PCIe	Efficacité du codage	Débit binaire (Gb/s)	Débit utile sur une ligne (GO/s)
1	5/8	2,5	0,250
2	5/8	5	0,500
3	118/120	8	0,985
4	118/120	16	1,969
5	118/120	32	3,938

FPGA. A titre d'exemple, Xilinx a annoncé en 2016 la première IP industrielle offrant la possibilité d'utiliser un lien PCIe x16 de génération 3, dont la spécification est apparue en 2010 [96]. Cette solution ne propose qu'un outil d'intégration matérielle et ne traite pas des problématiques d'intégration logicielle. Cela explique l'apparition de différents projets indépendants des contructeurs qui cherchent à répondre à ce besoin en proposant à la fois la partie matérielle et les outils d'intégration logiciel. Ces solutions peuvent être propriétaires et donner naissance à des entreprises, comme Xillybus et Northwest Logic. D'autres solutions entrent dans le cadre de projets de recherche et sont proposées de manière libre : EPEE [97], JetStream [98], ou RIFFA [99].

Cette dernière solution a notamment été utilisée sur la plate-forme NetFPGA SUME. Cette plate-forme est issue du projet NetFPGA, qui propose des plates-formes matérielles et logicielles destinées au prototypage de systèmes réseaux profitant d'accélération matérielle par FPGA [100]. La carte NetFPGA SUME contient également un FPGA Xilinx Virtex-7 690T. La caractéristique principale de ce composant est la présence de 3600 cellules DSP, qui correspondent à des multiplieurs accessibles dans la partie configurable. La multiplication étant une opération importante de l'algorithme présenté dans le Chapitre 2, cette carte est donc un choix pertinent pour les travaux de cette thèse afin de profiter à la fois des capacités de calcul et du lien PCIe déjà mis en place.

La solution RIFFA agit à la fois sur les aspects logicel et matériel (Figure 3.3). Du côté logiciel, la librairie RIFFA fournit des fonctions permettant la transmission de données entre le FPGA et l'ordinateur. Ces fonctions peuvent être utilisées en langage C/C++, Python et Java. Les fonctions d'envoi et de réception, qui renvoient le nombre de mots de 32 bits déplacés, sont définies ainsi :

— *fpga_send*(*fpga, chnl, data, len, destoff, last, timeout*) ;
— *fpga_recv*(*fpga, chnl, data, len, timeout*).

Les arguments suivant servent donc à correctement paramétrer la communication :

— *fpga* : désigne le FPGA concerné par la transaction, la solution RIFFA peut gérer plusieurs liens en même temps ;
— *chnl* : désigne l'accélérateur concerné, si plusieurs architectures de calcul sont présentes sur le FPGA ;
— *data* : pointeur vers l'adresse qui stocke les données à envoyer pour la fonction d'envoi, ou l'adresse où les données reçues sont mémorisées pour la réception ;
— *len* : nombre de mots de 32 bits qui doivent transiter ;

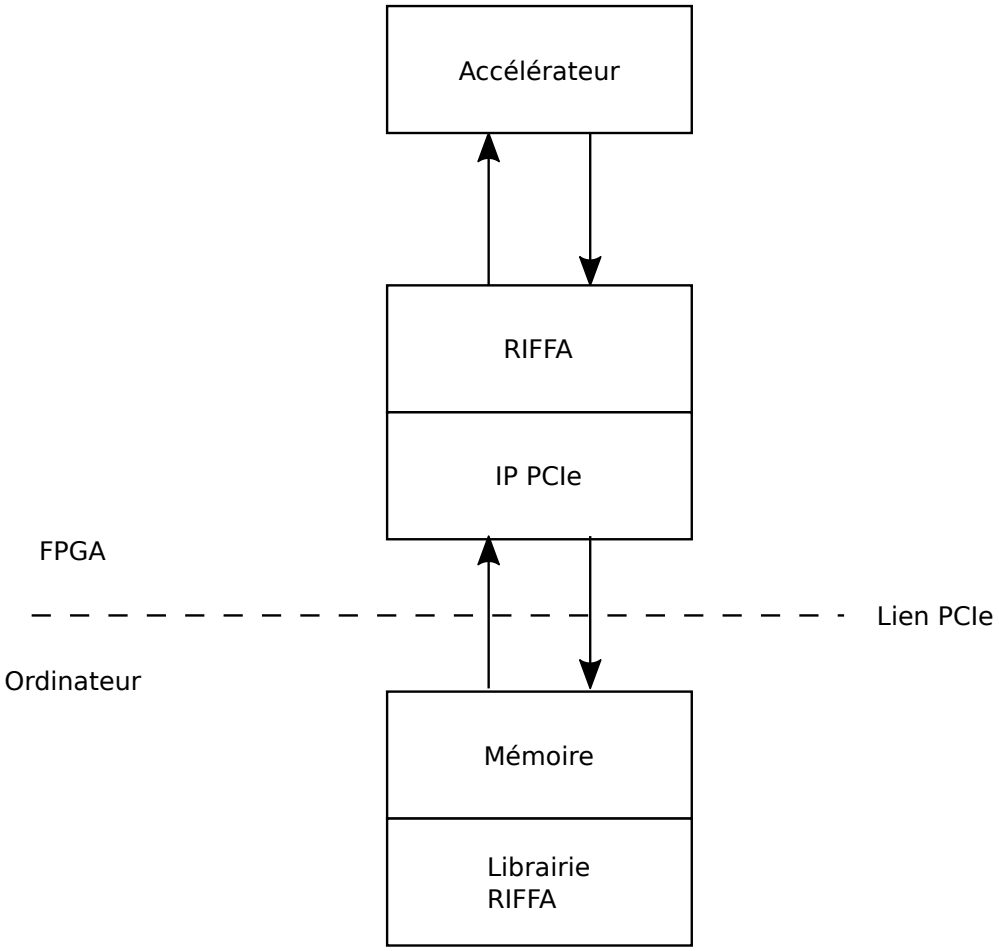


FIGURE 3.3 – Représentation de l'intégration d'un accélérateur à l'aide de la solution RIFFA.

— *destoff* : permet de gérer l'ordonnancement des données si plusieurs fonctions d'envoi sont utilisées en même temps ;
— *last* : donnée binaire qui indique si un autre envoi va suivre directement ou non. Cela permet donc d'optimiser l'envoi d'un grand nombre de mots ;
— *timeout* : temps d'attente en ms pour recevoir un acquittement du FPGA.

Les données peuvent donc facilement être transférées entre la mémoire et le FPGA en manipulant leur adresse de stockage dans la mémoire.

Pour la partie matérielle, la solution RIFFA se repose sur l'IP constructeur qui gère l'interface PCIe. C'est cet élément qu'il faut configurer afin de définir quelle est la génération du lien PCIe mis en place. Cette IP collecte les données issues du lien série pour les délivrer ensuite de manière synchrone. La partie RIFFA transfère ensuite les données vers l'accélérateur avec 2 *First in First Out* (FIFO) qui fonctionnent avec un protocole d'échange de type *handshake*. Cela permet de facilement intégrer un accélérateur pipeliné. Dans sa dernière version, le projet RIFFA gère les liens PCIe de génération 1 et 2 sur 8 lignes mais seulement 4 lignes pour la génération 3. L'usage du lien PCIe de génération 3 avec RIFFA n'apporte donc aucune augmentation de débit par rapport au lien de génération 2. Les communications se font de manière synchrone. Pour les liens PCIe x8 de génération 1 et 2, 128 bits de données peuvent être transmis

à chaque cycle. La différence de débit vient donc de la fréquence de fonctionnement. En effet, les données sont transmises à une fréquence de 125 MHz si la génération 1 est utilisée et la génération 2 fonctionne à une fréquence de 250 MHz.

La solution RIFFA est donc un outil pour rapidement mettre en place un lien PCIe entre un ordinateur et un accélérateur de calcul conçu sur FPGA. Ce projet a notamment fait ses preuves dans le cadre des travaux du groupe NetFPGA. Depuis, le site internet officiel de RIFFA a fermé et le projet n'est donc plus maintenu. Des solutions industrielles commencent également à fournir les outils nécessaires à l'intégration d'une carte FPGA sur un bus PCIe. C'est notamment le cas de BittWare, qui propose une carte contenant un FPGA Xilinx Virtex Ultrascale+ et qui fournit dans les sources un projet utilisant une communication PCIe x16 de génération 3 [101].

3.3.2 Intégration dans un SoC

Un système sur puce, ou SoC, contient au moins tous les éléments de base d'un ordinateur (processeur, mémoire, ports d'entrées/sorties) sur une seule et même puce. La conception de ces systèmes est donc souvent relative à un flot de conception ASIC mais il est également possible de mettre en œuvre un SoC sur un FPGA. En effet, il est possible d'implanter sur ces plate-formes des processeurs dits *softcore*. Cela signifie qu'une partie du FPGA est configurée de sorte à reproduire le comportement d'un processeur qui peut interagir avec le reste des éléments implantés.

Les constructeurs de FPGA permettent également la mise en œuvre de systèmes embarqués nécessitant des processeurs plus performants que les processeurs *softcore* en proposant des SoC FPGA. Il s'agit de puces disposant de processeurs classiques, alors dits *hardcore*, avec des liens de communications dédiés avec une partie de logique configurable. Par exemple, le SoC Zynq-7000 de Xilinx propose un contient un processeur ARM Cortex-A9 à 2 cœurs lié à une partie configurable de type Artix-7 [102].

Différentes stratégies peuvent être utilisées pour intégrer un coprocesseur dans un SoC : intégrer directement au processeur ou par l'intermédiaire du bus système (Figure 3.4). Intégrer directement un coprocesseur au processeur est uniquement possible en cas de conception basée sur un processeur *softcore*. En effet, cela demande de pouvoir modifier le processeur, au niveau de son jeu d'instruction mais aussi au niveau de ses entrées/sorties [103]. L'intégration via le bus système n'a pas d'impact sur le processeur. Cette méthode peut donc être réalisée pour tout type de SoC. Le coprocesseur est généralement connecté au bus en tant qu'esclave, c'est-à-dire qu'il ne peut émettre de requêtes à d'autres éléments du système. Le processeur doit alors gérer la transmission des données de la mémoire au coprocesseur.

L'intégration d'un coprocesseur dans un SoC est donc une tâche fastidieuse qui demande des connaissances concernant le fonctionnement d'un bus système ainsi que les interactions entre le processeur et les autres composants. Divers projets proposent des méthodes d'intégration plus flexibles et plus simple à mettre en œuvre. C'est le cas du processeur polymorphique Molen [104]. Ce processeur est basé sur un jeu d'instructions restreint, qui permet de contrôler n'importe quel accélérateur. Le jeu d'instructions minimal possible contient 4 instructions :

— *set* : permet de configurer l'accélérateur attaché au processeur Molen ;
— *execute* : permet de contrôler l'exécution des opérations de l'accélérateur ;

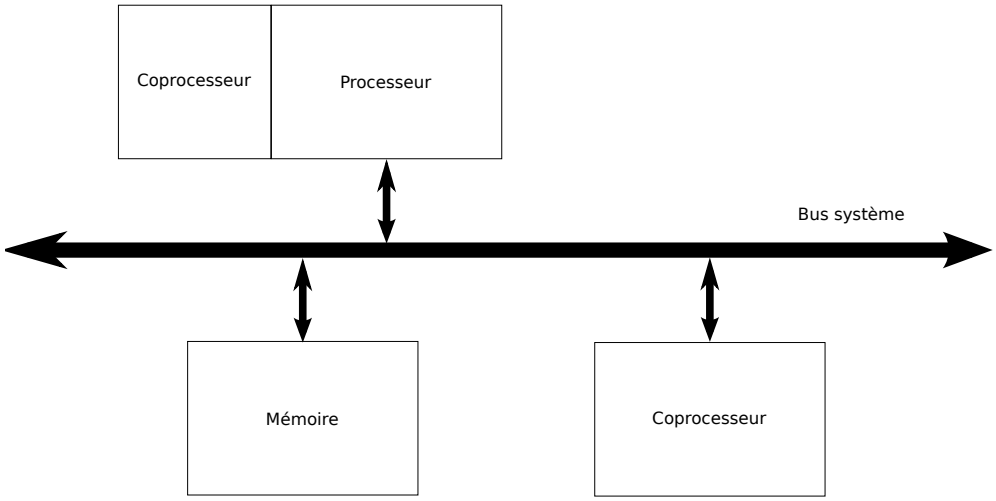


FIGURE 3.4 – 2 possibilités d'intégration de coprocesseurs dans un SoC : interfaçage direct sur le processeur ou sur le bus.

— *movtx* : permet de déplacer une donnée d'un registre du processeur central vers un registre d'échange le processeur central et le processeur Molen ;
— *movfx* : permet de déplacer une donnée du registre d'échange vers un registre du processeur central.

La machinerie du processeur Molen s'insère entre le bus système et le processeur (Figure 3.5). Cette solution analyse toutes les instructions envoyées au processeur central. Si elles correspondent au jeu d'instructions du processeur Molen, ce dernier les gère directement. Il est donc difficile d'assurer une bonne parallélisation entre l'accélérateur et le processeur central. Une parfaite connaissance du jeu d'instructions du processeur central est de plus nécessaire afin d'être certain que le processeur Molen n'utilise pas d'identifiants d'instructions déjà employés. La solution n'est donc pas générique. Cela demande aussi d'avoir l'accès à l'interface entre le processeur et le bus système, donc cette solution n'est pas possible pour tout type de système embarqué. De plus, dans le cas d'architectures multi-processeurs, cela demande d'intégrer une machinerie Molen par processeur, ce qui augmente le surplus matériel.

Pour répondre à ces défauts, nous avons conçu l'architecture nommée Ouessant qui s'inspire du jeu d'instructions minimal du processeur Molen [105]. Son objectif premier est de proposer une manière simple d'intégrer un ou plusieurs accélérateurs matériels dans n'importe quel système embarqué, peu importe le processeur central utilisé et le protocole du bus système. Cette architecture ne s'intègre pas entre le processeur principal et le bus mais directement sur ce dernier et de manière indépendante (Figure 3.6). L'architecture Ouessant est composée de 3 parties (Figure 3.7). En bout de chaine se trouvent les accélérateurs intégrés au sein de l'architecture Ouessant. Les liens avec le contrôleur du coprocesseur sont réalisés à l'aide de files de données (ou FIFO). Ces FIFO sont génériques et ont des fonctionnalités de mise en série ou de parallélisation. Elles permettent donc d'intégrer une grande variété d'accélérateurs. Le contrôleur a le fonctionnement classique d'un processeur non-pipeliné. Il demande les accès à la mémoire pour recevoir ses instructions, les décode puis les exécute. Il est totalement générique et compatible avec n'importe quelle architecture de système em-

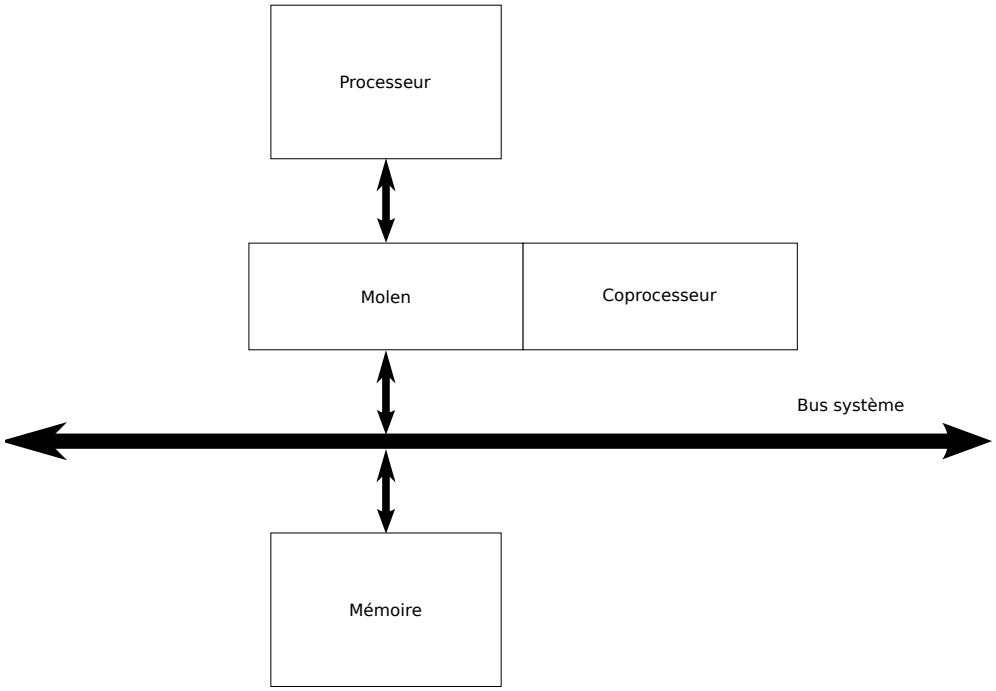


FIGURE 3.5 – Intégration d'un coprocesseur à l'aide du processeur polymorphique Molen : l'architecture Molen se situe en coupure entre le bus système et le processeur.

barqué. Il ne dispose donc d'aucune information sur le processeur central ou de son emplacement en terme d'adressage sur le bus système.

Le coprocesseur Ouessant a un accès maître et un accès esclave sur le bus. La partie esclave est chargée de recevoir des paramètres de configuration, envoyés par le processeur central. Ces paramètres sont des adresses correspondant à la première adresse de zones de mémoire spécifiques (ou *bank*). Le système indique ainsi au coprocesseur Ouessant où se trouve sa première instruction, l'emplacement des données à traiter et celui dans lequel mémoriser les résultats. L'interface esclave contient également un registre d'un signal de démarrage, qui permet de lancer l'exécution du contrôleur qui va alors commencer par faire la demande d'accès mémoire pour lire sa première instruction dont l'adresse a été configurée. C'est l'interface maître du coprocesseur Ouessant qui se charge des requêtes à la mémoire et transmet le résultat de ces dernières au contrôleur. Le déroulement d'une opération via le coprocesseur proposé est résumé en Figure 3.8. Une fois que le processeur central a transmis tous les paramètres nécessaires et a lancé le calcul sur le coprocesseur, il peut exécuter sa liste d'instructions de manière classique, en attendant que le coprocesseur le prévienne que l'opération demandée est finie, via un signal qui prend la forme d'une interruption. Cela permet de simplifier la parallélisation entre le processeur central et l'accélérateur matériel.

Les interfaces maître et esclave de la solution Ouessant sont composées de deux parties. d'un côté, une première partie totalement générique qui gère tous les mécanismes propres au coprocesseur et une seconde partie qui sert à adapter les signaux génériques de la première au protocole de bus utilisé par le système. Pour rendre le coprocesseur compatible avec un nouveau protocole de bus, il faut donc implanter le bloc d'adaptation correspondant. C'est toutefois la seule partie de la solution propo-

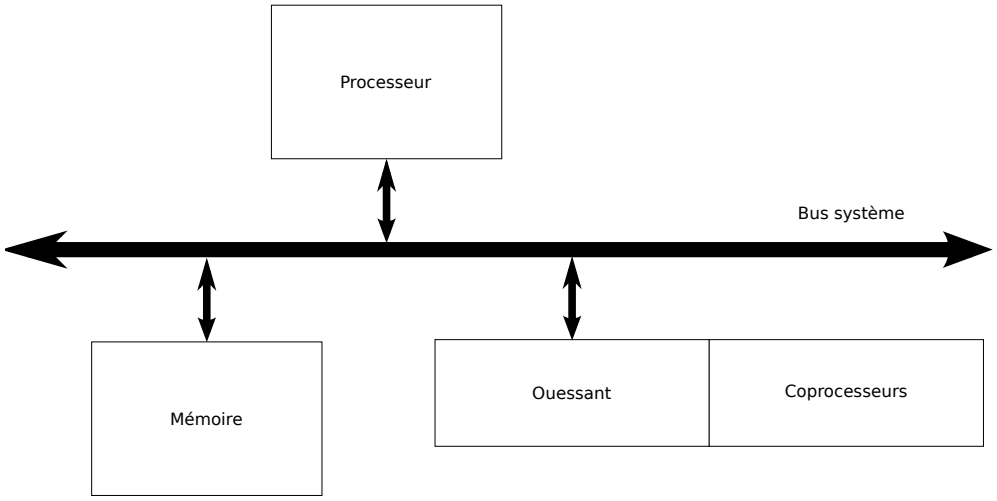


FIGURE 3.6 – Intégration d'un coprocesseur à l'aide du coprocesseur Ousessant : l'architecture Ouessant est connectée au bus système.

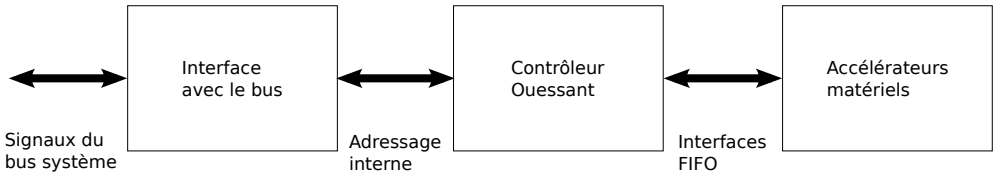


FIGURE 3.7 – Architecture du coprocesseur Ouessant.

sée qui nécessite d'être modifiée pour adapter le coprocesseur à un système embarqué n'utilisant pas un bus déjà supporté.

La liste des instructions du coprocesseur Ouessant est la suivante :

— *Move To Coprocessor* (mvtc) : déclenche un transfert de données de la mémoire vers le coprocesseur ;
— *Move From Coprocessor* (mvfc) : déclenche un transfert de données du coprocesseur vers la mémoire ;
— *Execution* (exec) : déclenche les calculs de l'accélérateur et attend la fin de ceux-ci ;
— *End of Operation* (eop) : indique la fin du programme et avertit le processeur principal de la fin des opérations.

Ces instructions sont suffisantes pour assurer le bon fonctionnement du coprocesseur. Dans le cas d'une architecture 32 bits, une instruction du coprocesseur Ouessant est organisé de la façon suivante (Figure 3.9) :

— code d'instruction (5 bits) : c'est la partie décodée par le contrôleur afin de déterminer l'instruction à exécuter. Avec les 5 bits disponibles, cela laisse une possibilité d'avoir jusqu'à 32 instructions pour les futures amélioration du coprocesseur ;
— *bank* (3 bits) : identifiant de la zone mémoire concernée par l'instruction. Ne concerne que les instructions mvtc et mvfc. Par exemple, si le champ précédent indique l'instruction mvtc et que le champ *bank* contient la valeur 2, il s'agit d'un transfert de données vers le coprocesseur depuis la mémoire, à partir de l'adresse de la *bank* 2, conservée dans l'interface lors de la configuration du

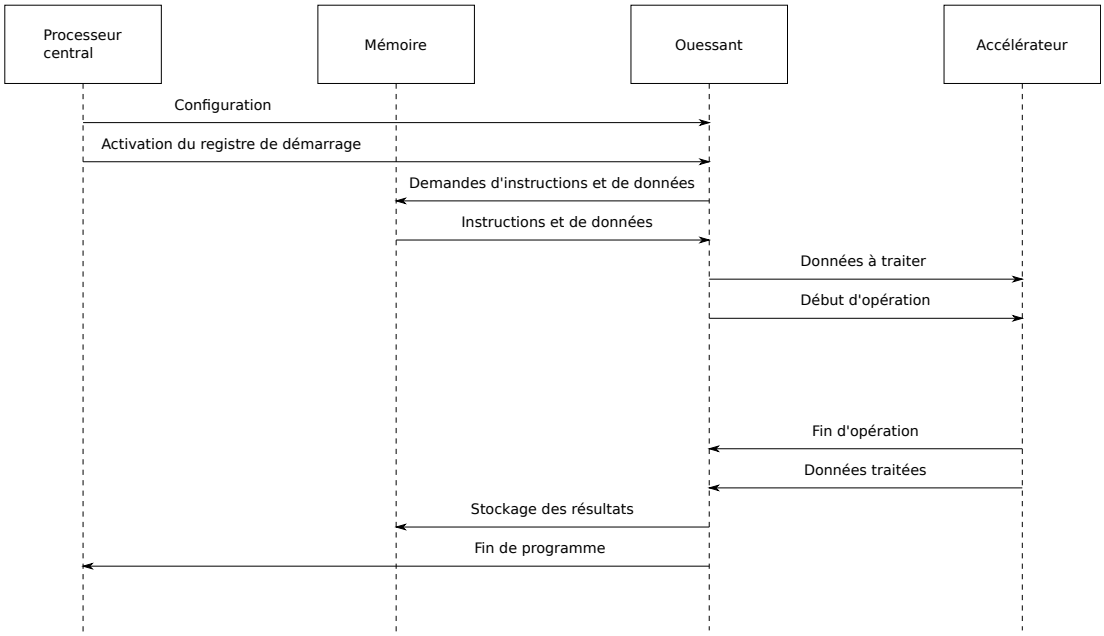


FIGURE 3.8 – Déroulement de l'utilisation du coprocesseur Ouessant.

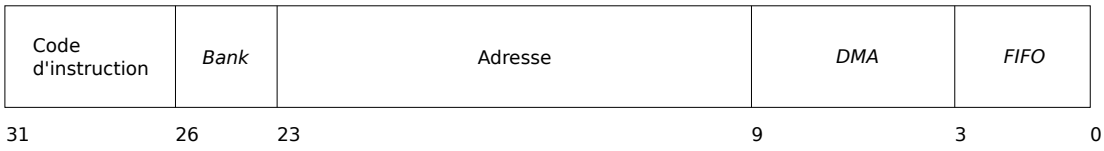


FIGURE 3.9 – Découpe d'une instruction du coprocesseur Ouessant.

coprocesseur ;

- adresse (14 bits) : sert à l'adressage interne au coprocesseur. De part la volonté de généricité par rapport au système sur lequel il est intégré, le coprocesseur utilise des adresses génériques qui seront par la suite additionnées aux adresses des différentes *banks* stockées dans le bloc d'interface ;
- *Direct Memory Access* (DMA) (6 bits) : permet de déterminer la taille du paquet de données à transmettre si l'utilisation du DMA est souhaitée ;
- FIFO (4 bits) : permet de désigner la FIFO concernée par l'instruction quand plusieurs accélérateurs sont connectés.

L'influence du coprocesseur Ouessant sur les ressources utilisées et l'accélération possible, celui-ci a été implanté et intégré dans un système basé sur le processeur Leon3 [106]. Il s'agit d'un processeur *softcore* libre de droit basé sur l'architecture SPARCv8 dont le code VHDL est mis à disposition. Le système repose sur un bus Amba2 et un système Linux a correctement été intégré au processeur. L'architecture a été testée avec 2 accélérateurs matériels. Le premier est dédié à une opération de transformée en cosinus discrète inverse, plus connue sous le terme anglais *Inverse Discrete Cosine Transform* (IDCT). C'est une opération notamment utilisée pour le décodage du standard JPEG. Le second est une transformée de Fourier discrète, ou *Discrete Fourier Transform* (DFT), sur 256 points et provenant de l'outil SPIRAL, présenté précédemment dans ce Chapitre. Les tests ont été effectués sur la carte Nexys4 de Digilent, équipée d'un FPGA Artix7 LX100T. Tout le système est cadencé à 50 MHz.

Tableau 3.2 – Temps de calcul en nombre de cycles et facteur d'accélération par utilisation du coprocesseur Ouessant.

Opération	Logiciel	Accéléré (Accélérateur seul)	Accélération
IDCT	5000	3000 (18)	1,67
DFT	600000	7000 (2485)	85,71

Concernant la machinerie du coprocesseur, moins de 1000 LUT et 750 *Flip-Flop* (FF) sont utilisés. Cela concerne l'interface avec le bus, le contrôleur du coprocesseur et le contrôle des FIFO. Ces FIFO sont inférées en tant que BRAM dont le nombre va dépendre de l'accélérateur intégré au sein du coprocesseur. Un test d'accélération a été réalisé en exécutant les 2 fonctions de manière logicielle et via l'accélérateur intégré au coprocesseur Ouessant. Le Table 3.2 montre le nombre de cycles nécessaires pour effectuer chaque opération et le facteur d'accélération associé. Si l'utilisation du coprocesseur apporte un gain dans les 2 cas, ces mesures montrent que le transfert des données a un impact conséquent sur l'accélération. En effet, dans le cas de l'accélération matérielle de l'IDCT, seulement 18 cycles sur les 3000 nécessaires ne concernent que l'accélérateur. La différence provient du temps de transfert des données, du fonctionnement du système d'exploitation et de la partie contrôle du coprocesseur. L'opérateur de DFT est plus complexe et nécessite plus de cycles de fonctionnement, ce qui permet d'atténuer l'impact des autres délais et ainsi augmenter le facteur d'accélération.

Il est donc intéressant d'estimer le surcoût causé par la partie du contrôle du coprocesseur. En effet, le transfert de données est dans tous les cas inévitable et il est difficile de jouer sur le fonctionnement du système d'exploitation. L'accélération matérielle de l'opération de DFT a donc été lancée sans passer par un système d'exploitation. Le nouveau temps d'exécution est de 4000 cycles, ce qui donne un surcoût de 3000 cycles venant de Linux. En retirant les 2485 cycles nécessaires à l'accélérateur, il en reste 1515 pour gérer le transfert de 1024 mots de 32 bits. Au final, il faut en moyenne 1,5 cycle au coprocesseur Ouessant pour déplacer une donnée entre la mémoire et l'accélérateur, ce qui est un surcoût raisonnable. Ces résultats ont fait l'objet d'une publication à DATE 2016 [105].

Le coprocesseur Ouessant a par la suite été amélioré [107]. La compatibilité avec le bus AXI4 a été ajoutée, ce qui permet par exemple d'intégrer le coprocesseur au sein des SoC FPGA Zynq de Xilinx. Le jeu d'instructions a été étendu afin d'offrir d'autres fonctionnalités. La première est l'exécution non-bloquante des calculs. Cela signifie que le contrôleur du coprocesseur peut continuer à recevoir et exécuter des instructions pendant qu'un accélérateur traite des données. Cela permet de préparer sans temps mort une autre exécution pour un autre accélérateur intégré sur le coprocesseur. La possibilité de relier dynamiquement 2 accélérateurs par une FIFO a également été implantée, ce qui permet d'effectuer 2 opérations successives sans avoir besoin de faire transiter les données par la mémoire. Cela rend aussi possible la réutilisation d'un accélérateur si l'opération qu'il réalise est nécessaire à plusieurs reprises.

Cette fonctionnalité a été testée sur une application de génération de canal de communication (Figure 3.10). Le processeur central Leon3 génère des échantillons gaus-

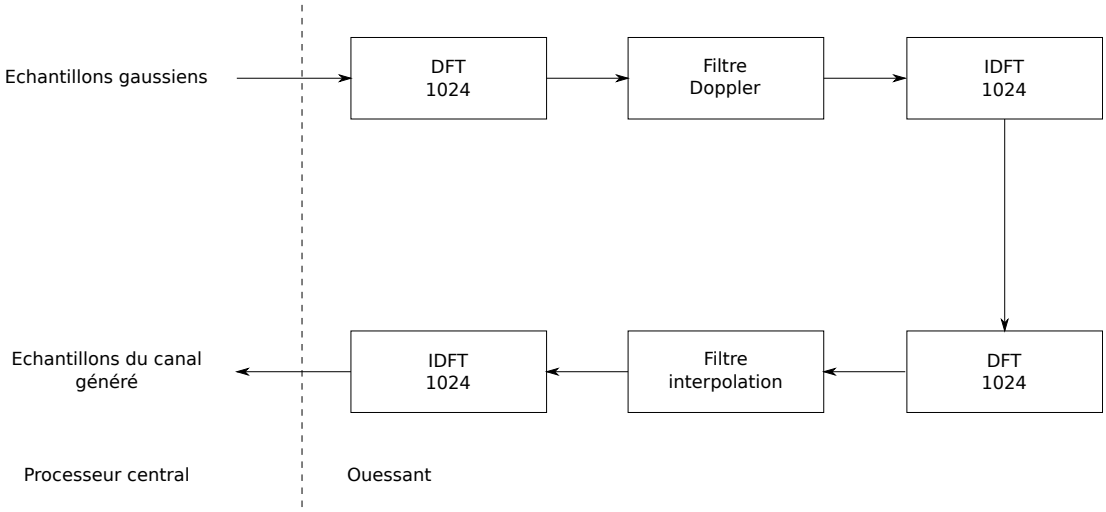


FIGURE 3.10 – Architecture pour le test de connexion entre accélérateurs du coprocesseur Ouessant.

siens qui sont envoyés au coprocesseur Ouessant dans lequel sont intégrés 4 accélérateurs : un bloc de DFT sur 1024 points, un bloc de l'opération inverse, l'*Inverse Discrete Fourier Transform* (IDFT) et 2 filtres. L'agencement de ces opérations permet bien de tester la fonctionnalité de connexion dynamique entre 2 accélérateurs et la réutilisation des blocs de DFT et d'IDFT. Cette solution offre un facteur d'accélération de plus de 650 par rapport à une version totalement logicielle.

Via le projet Ouessant, nous avons voulu contribuer à la communauté du système embarqué en proposant un outil pour faciliter l'intégration d'accélérateurs métériels dans des systèmes complexes, afin de réduire de temps de production de nouvelles solutions. La solution proposée est totalement indépendante du système auquel on l'intègre et son impact en temps de calcul par rapport à un accélérateur intégré directement sur le bus est négligeable. Le projet est libre de droit est ainsi utilisable par la communauté [108].

Depuis, les contructeurs de SoC FPGA ont également proposé leur propre solution pour faciliter l'intégration d'accélérateurs. C'est notamment le cas d'Intel FPGA avec l'*Acceleration Stack*, suite logicielle qui permet de gérer tous les aspects de l'intégration, que ce soit du côté matériel ou logiciel [109]. La librairie logicielle proposée permet de contrôler le FPGA, par exemple en le configurant ou en employant des fonctions de débogage [110]. Du côté matériel, la partie configurable est reliée au processeur Xeon par 2 liens PCIe x8 de génération 3 ainsi qu'un lien *Ultra Path Interconnect* (UPI), qui est un protocole propriétaire d'interconnexion entre un processeur Intel et d'autres modules [111]. Ces liens sont pris en charge dans la partie configurable par une partie statique nommée Unité d'Interface FPGA. C'est cet élément qui se charge de la communication avec l'accélérateur configuré sur le FPGA via un lien *Core Cache Interface* (CCI) qui est une interface abstraite utilisée pour faciliter l'intégration (Figure 3.11).

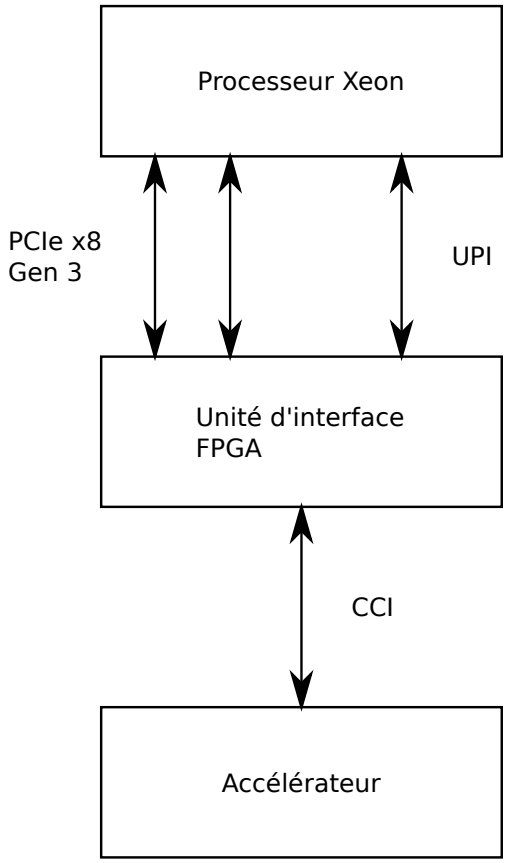


FIGURE 3.11 – Représentation de la solution d'intégration Intel.

3.4 Conclusion

La mise en œuvre de l'accélération matérielle d'une fonction logicielle par un circuit dédié comporte 2 étapes : la réalisation du circuit et son intégration dans un système complet. La conception d'un circuit numérique dédié à un algorithme demande des compétences spécifiques et un temps de conception conséquent. Les outils de synthèse de haut niveau permettent de réduire ce temps de conception et de rendre cette tâche plus accessible. Cette méthode offre un bon compromis entre temps de travail humain et performance. Cependant, dès que les architectures deviennent complexes, le résultat est beaucoup moins optimisé qu'une description matérielle au niveau RTL.

Un autre moyen d'accélérer la conception d'une description matérielle est l'utilisation d'IPs. Il s'agit de blocs de description d'éléments usuels fournis par les constructeurs ou par des projets communautaires. Les IPs proposées par les constructeurs de FPGA ont l'avantage d'être optimisées pour les FPGA du même constructeur mais ne peuvent pas être prises en compte par des flots de conception concurrents. Cela empêche de porter l'architecture sur ASIC. L'utilisation d'IPs libres de droit et non spéficiques à une plate-forme est intéressante, si les blocs proposés répondent au besoin de l'architecture. Il a donc été choisi pour ces travaux de thèse de privilégier la description RTL de l'architecture, tout en tenant compte de l'opportunité d'utiliser des IP proposées par la communauté si cela est avantageux.

L'intégration d'un accélérateur dans un système complet dépend de la nature de

ce système. En effet, la problématique n'est pas la même s'il s'agit d'un ordinateur classique ou d'un système embarqué. Pour nos perspectives à court terme, le but est de prototyper une solution d'accélération pour le problème direct de l'EEG. Les calculs sont actuellement effectués à l'aide de stations de calcul. Il faut donc réussir à intégrer un accélérateur au sein de ce système. Le bus PCIe est un lien de communication série dont le but est d'assurer un haut débit de transfert de données. Cette technologie est depuis longtemps adoptée par les cartes graphiques. Il y a cependant moins de recul sur cette technologie pour la communication avec un FPGA. C'est pourquoi plusieurs projets de recherche proposent des outils matériels et logiciels pour mettre en place la communication entre un proceseur et un FPGA via un lien PCIe. L'un de ces projets, nommé RIFFA, est notamment intégré au sein de la plate-forme NetFPGA SUME, qui contient également un FPGA ayant une grande capacité de calcul. Nous avons donc décidé de choisir cette carte pour effectuer le développement de l'accélérateur de la fonction présentée dans le Chapitre 2. La conception de cet accélérateur est présentée dans la suite de ce document.

Dans les perspectives à long terme des travaux de thèse, l'accélérateur développé doit être intégré dans un système embarqué. Dans un SoC, l'accélérateur est soit directement relié au processeur, soit connecté au bus système. La première solution nécessite de modifier l'architecture du processeur, ce qui n'est pas tout le temps possible. La connexion d'un accélérateur au bus est une tâche complexe. C'est pourquoi nous avons proposé Ouessant, coprocesseur pouvant être intégré à n'importe quel système embarqué et sur lequel un accélérateur peut facilement être connecté. Cette solution permet également d'assurer une bonne parallélisation entre le proceseur central et l'architecture de calcul dédiée. Cette solution est donc très intéressante pour la perspective de proposer à terme un système complet chargé d'effectuer des calculs pour une ICM.

Chapitre 4

Conception et implantation d'opérateurs clés pour l'électromagnétisme computationnelle

4.1 Introduction

L'étude de l'algorithme à accélérer réalisée dans le Chapitre 2 a montré la possibilité de réaliser un circuit totalement pipeliné avec un bon nombre de calculs parallélisables. La présence de fonctions non-linéaires comme la division, la raciné carrée, le logarithme népérien et l'arctangente a également été mise en avant. Il est donc important de s'intéresser à l'implantation de ces fonctions afin d'être certain que cela ne détériore pas les performances globales de l'accélérateur. En effet, dans un circuit pipeliné, chaque élément doit être optimisé pour assurer la fréquence de fonctionnement la plus haute possible. Ce chapitre est consacré aux opérations de division et de racine carrée qui ont fait l'objet d'optimisations originales. Les autres fonctions non-linéaires ont été implantées selon l'état de l'art et seront détaillées dans le chapitre dédié au prototypage.

Différentes méthodes d'implantation sont tout d'abord revues avant de présenter nos travaux d'implantation sur FPGA Xilinx. Les architectures proposées sont ensuite testées dans un flot de conception ASIC.

4.2 Présentation de différentes méthodes d'implantation

Dans le Chapitre 2, les différents calculs à réaliser ont été présentés. Cela a permis de mettre en valeur la présence de divisions. La majorité de celles-ci ont pour dénominateur une longueur l_i, comme le calcul de la valeur $s_1{}^{\text{-}}$

$$s_1{}^{\text{-}} = -\frac{(l_3 - u_0)(l_3 - u_3) + v_0 v_3}{l_1}. \tag{4.1}$$

Pour rappel, la distance l_1 est donnée par la formule suivante :

$$l_1 = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2 + (z_3 - z_2)^2}. \tag{4.2}$$

Le calcul de $s_1{}^-$ demande donc de calculer tout d'abord une racine carrée, puis d'effectuer une division. Il est possible de simplifier le calcul considéré en calculant directement l'inverse de la racine carrée, $1/\sqrt{x}$. Cela permet de calculer directement la valeur $1/l_1$ puis d'effecteur la multiplication avec le numérateur pour obtenir $s_1{}^-$. Au final, le passage par l'inverse de la racine carrée permet de réduire le nombre d'opérations non-linéaires à effectuer de 2 à 1. Cet opérateur permet également de facilement calculer la racine carrée d'un nombre en utilisant la relation suivante :

$$x \times \frac{1}{\sqrt{x}} = \sqrt{x}. \tag{4.3}$$

Il est donc primordial d'utiliser un opérateur de calcul de l'inverse de la racine carrée performant puisqu'il peut être utilisé à la fois pour les calculs de racine carrée, mais aussi pour la majorité des divisions que comprend l'algorithme. Cet opérateur est de plus un élément clé en électromagnétisme car il est utilisé pour calculer la valeur d'un potentiel électrostatique, qui est inversement proportionnel à une distance [24]. Le fait de travailler sur cet opérateur a donc une double utilité : optimiser l'accélérateur que nous cherchons à concevoir tout en offrant à la communauté de l'électromagnétisme un élément d'accélération matérielle.

Toutefois, cet opérateur d'inverse de la racine carrée, ne permet pas de calculer toutes les divisions nécessaires dans l'algorithme d'intégration en 3 dimensions. Il faut donc toujours considérer l'usage d'un opérateur de division, comme par exemple pour le calcul des valeurs f_{2i} :

$$f_{2i} = \ln\left(\frac{R_i{}^+ + s_i{}^+}{R_i{}^- + s_i{}^-}\right). \tag{4.4}$$

Il y a 2 pistes intéressantes pour effectuer une division. La plus naturelle est de la calculer directement, ce qui implique donc un opérateur à 2 entrées. La seconde piste est de séparer la division en 2 autres opérations : le calcul de l'inverse du dénominateur qui est ensuite multiplié par le numérateur.

Dans la suite, plusieurs méthodes d'implantation de l'état de l'art de ces opérateurs sont présentées et évaluées selon les contraintes de notre accélérateur. En effet, pour apporter une accélération sur FPGA, il faut que l'architecture soit entièrement pipelinée. Cela implique la nécessité d'instancier l'opérateur autant de fois que nécessaire. L'étude de la fonction à accélérer a permis de déterminer qu'il faut 6 blocs de calcul d'inverse, et 8 blocs de calcul de l'inverse de la racine carrée. Il faut donc être prudent sur le nombre et le type de ressources utilisées par rapport à la plate-forme de prototypage choisie à la fin du Chapitre 3. Il est également avantageux d'avoir des implantations génériques par rapport à taille des entrées. En effet, la représentation en virgule fixe implique une évolution du nombre de bits sur lesquels les données sont représentées au fil des opérations.

4.2.1 Fonctions tabulées

Une première méthode pour estimer une fonction $f(x)$ est d'utiliser une LUT. La façon la plus naïve de réaliser une telle implantation est de tabuler toutes les combinaisons possibles. Cela prend la forme d'une mémoire dans laquelle la valeur $f(x)$ peut être lue à l'adresse x. Cette méthode est très simple à mettre en œuvre, mais elle

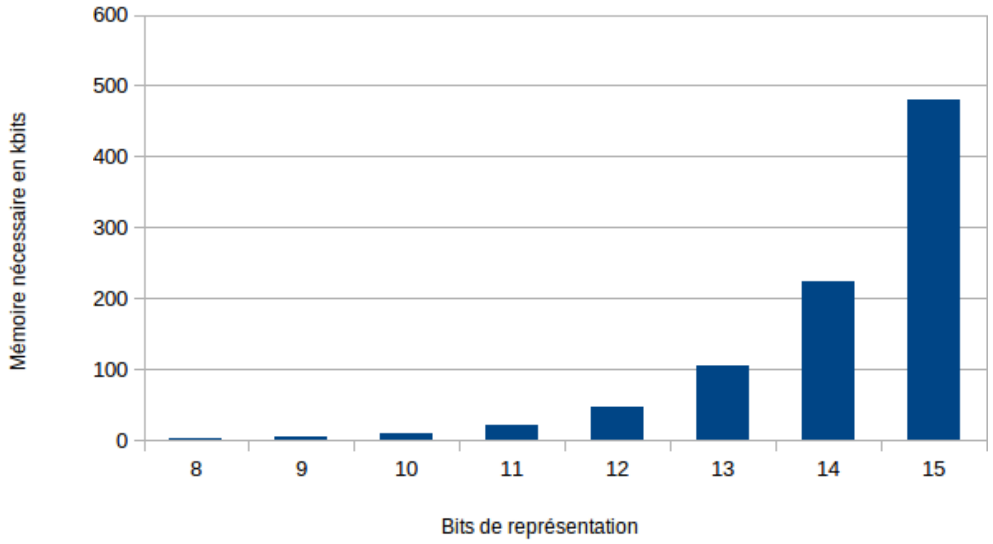


FIGURE 4.1 – Evolution de la capacité de la mémoire en kbits en fonction du nombre de bits de représentation dans le cas ou les entrées et sorties sont codées sur le même nombre de bits.

est surtout efficace quand le nombre de bits de représentation est restreint. En effet, si l'entrée et la sortie sont représentées sur n bits, la taille de la mémoire nécessaire est égale à $n \times 2^n$ bits. La croissance est donc exponentielle (Figure 4.1). Il n'est donc pas pertinent d'utiliser de la tabulation complète des données représentées sur trop de bits. A titre de comparaison, le FPGA utilisé dans le cadre de ces travaux de thèse, le Virtex-7 690T dispose de 1470 BRAM ayant chacun une capacité de 36 kbits. Cela représente un total de 52920 kbits. A titre de comparaison, cela n'est pas suffisant pour tabuler une fonction sur 22 bits, qui nécessite une capacité mémoire de 92274 kbits.

Des utilisations moins basiques des mémoires peuvent être faites. L'une des possibilités est d'effectuer une approximation linéaire. Pour cela, il faut découper l'intervalle de définition de la fonction en m parties égales, avec m étant de préférence une puissance de 2. Cela permet de savoir dans quel intervalle se trouve l'entrée x en analysant les k bits de poids fort, avec $2^k = m$. Il suffit alors pour chaque intervalle de stocker un cœfficient directeur, c, et une ordonnée à l'origine, o, pour pouvoir procéder à l'approximation linéaire avec une multiplication et une addition (Figure 4.2). Le nombre de bits de représentation pour le cœfficient et l'ordonnée peut être choisie en fonction de la précision souhaitée pour ces paramètres. C'est donc cette précision ainsi que le nombre de sous-intervalles considérés qui dimensionnent la mémoire. La précision globale de l'opérateur est déterminée par le nombre de sous-intervalles. Le fait que ces sous-intervalles soient uniformément distribués sur l'intervalle initial rend cette méthode moins efficace pour les fonctions qui changent brusquement de pente puisque la découpe ne se fait pas forcément à l'endroit idéal.

Une amélioration de cette méthode est donc d'adapter la découpe en sous-intervalles à la fonction que l'on cherche à évaluer [11]. L'objectif est d'optimiser chaque sous-intervalle. Dans l'exemple de la fonction $\sqrt{-\ln(x)}$ définie sur l'intervalle $]0,\ 1]$ et avec x représenté sur 32 bits, 59 sous-intervalles sont utilisés et ceux-ci sont concentrés sur les parties non-linéaires de la fonction (Figure 4.3). Une autre possibilité est

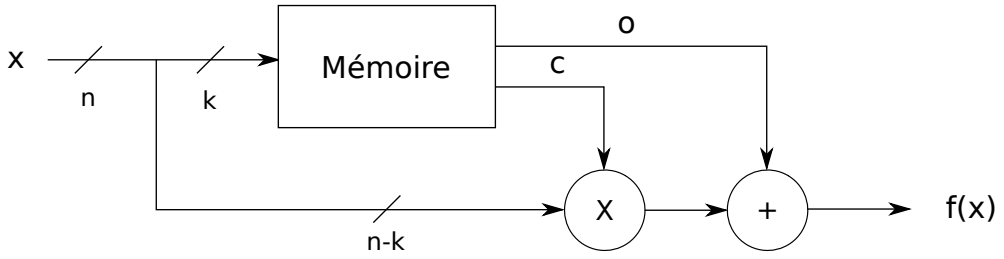


FIGURE 4.2 – Architecture pour l'approximation linéaire pour une fonction $f(x)$.

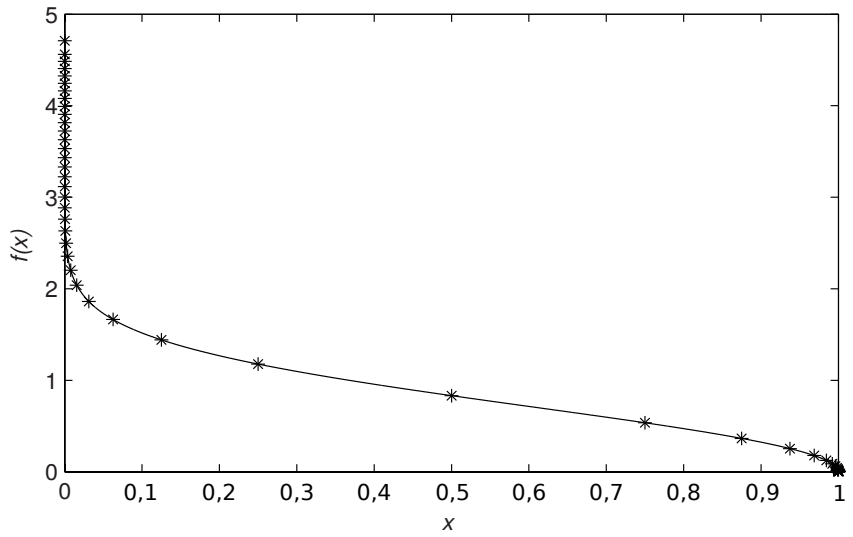


FIGURE 4.3 – Découpe optimisée de la fonctoin $\sqrt{-\ln(x)}$ en 59 segments sur l'intervalle $]0,\ 1]$. Les étoiles représentent les extrémités des segments [11].

de décomposer sur un intervalle connu la fonction à évaluer en produit de fonctions paraboliques dont les cœfficients sont stockés en mémoire [112].

L'utilisation de tables de correspondance permet d'évaluer des fonctions à un seul argument. Elles sont généralement implantées dans des blocs de mémoire. Tabuler entièrement une fonction est la façon la plus simple de procéder mais ne convient pas pour les données représentées sur de nombreux bits à cause du besoin de mémoire. L'utilisation d'une LUT peut aussi permettre de réaliser des approximations linéaires ou polynomiales. Ce sont alors des cœfficients et autres constantes qui sont mémorisés. Cela demande quelques opérations en sortie de la mémoire, ce qui augmente la latence de l'opérateur. Il y a donc un compromis à choisir entre la mémoire nécessaire et la complexité d'implantation. De plus, les techniques d'approximation nécessitent de travailler sur un intervalle défini afin de générer facilement les données à stocker. Cela demande une mise à l'échelle à l'entrée et à la sortie de la fonction. Si cette mise à l'échelle est relativement simple à gérer pour la fonction d'inverse (décalage du même nombre de bits en entrée et en sortie), cela rajoute rapidement des contraintes pour d'autres fonctions. Par exemple, dans le cas de la racine carrée, si l'entrée x n'est pas dans l'intervalle considéré, il faut la multiplier par 2^k, avec k un entier tel que le produit appartient bien à l'intervalle. Pour obtenir la valeur finale, il faut donc diviser la sortie de l'approximation par $2^{\frac{k}{2}}$. Il est alors préférable de s'assurer que la valeur k soit paire

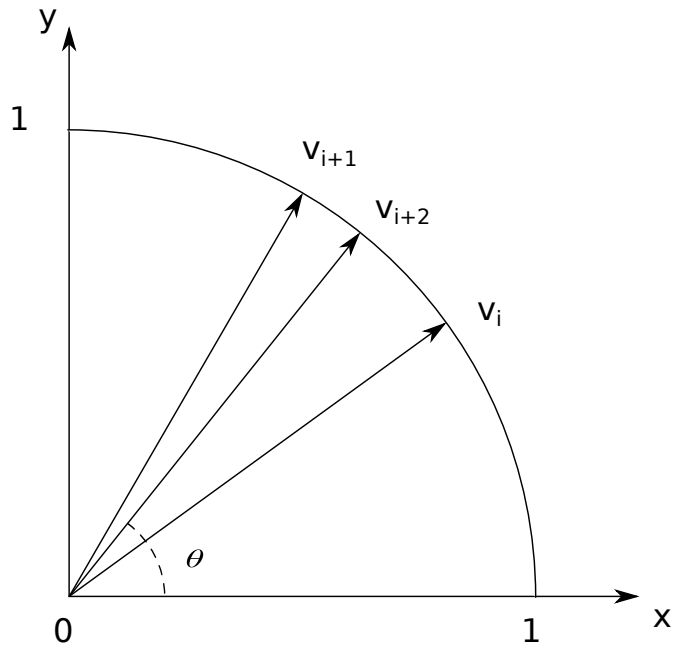


FIGURE 4.4 – Rotations de CORDIC.

afin d'implanter simplement la mise à l'échelle en sortie de l'opérateur.

4.2.2 CORDIC

Le CORDIC est un algorithme itératif permettant le calcul de fonctions trigonométriques et hyperboliques paru en 1959 [113]. Le CORDIC trigonométrique repose sur des rotations de vecteurs dans la zone supérieure droite du cercle trigonométrique (Figure 4.4). Le principe consiste à amener un vecteur d'entrée à une position finale par des rotations successives. L'algorithme considère 3 entrées : x_0 et y_0, les coordonnées du vecteur initial ainsi que z_0 qui représente un angle. Les sorties sont x_n, y_n et z_n, soit l'évolution des entrées après n rotations.

Il existe 2 modes d'utilisation du CORDIC. Le premier est le mode rotation. Il consiste à faire tourner le vecteur d'entrée d'un certain angle, renseigné dans z_0. Pour ce mode, les équations d'itérations sont :

$$x_{i+1} = x_i - y_i d_i 2^{-i}, \tag{4.5}$$

$$y_{i+1} = y_i + x_i d_i 2^{-i}, \tag{4.6}$$

$$z_{i+1} = z_i - d_i \arctan 2^{-i}, \tag{4.7}$$

avec

$$d_i = \begin{cases} 1 & \text{si} \quad z_i \geq 0, \\ -1 & \text{si} \quad z_i < 0. \end{cases} \tag{4.8}$$

Le principe est donc de faire assez d'itérations pour que la donnée z tende vers 0. Ces itérations sont simples à implanter puisque les multiplications par 2^{-i} peuvent

être effectuées par un décalage de la représentation de la donnée. Voici les formules donnant le résultat final après n itérations :

$$x_n = A_n(x_0 \cos z_0 - y_0 \sin z_0), \tag{4.9}$$

$$y_n = A_n(y_0 \cos z_0 + x_0 \sin z_0), \tag{4.10}$$

$$z_n = 0. \tag{4.11}$$

La valeur A_n est un gain correctif pour compenser la diminution de l'amplitude du vecteur au fil des itérations. Sa valeur dépend également du nombre d'itérations effectuées :

$$A_n = \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}}. \tag{4.12}$$

Le second mode est le mode vectoriel. Son but est de faire tourner le vecteur d'entrée jusqu'à ce qu'il soit aligné avec l'axe des abscisses. Les formules itératives sont les mêmes que pour le mode rotation, mais la valeur d_i a une autre définition :

$$d_i = \begin{cases} -1 & \text{si} \quad z_i \geq 0, \\ 1 & \text{si} \quad z_i < 0. \end{cases} \tag{4.13}$$

Les sorties du mode vectoriel après n itérations sont :

$$x_n = A_n \sqrt{x_0^2 + y_0^2}, \tag{4.14}$$

$$y_n = 0, \tag{4.15}$$

$$z_n = z_0 + \arctan \frac{y_0}{x_0}. \tag{4.16}$$

Cet algorithme présente l'avantage d'être simple à implanter. En effet, aucun bloc multiplicateur n'est nécessaire pour les itérations puisque les multiplications par 2^{-i} peuvent directement être effectuées avec des décalages. Cette méthode permet également de gagner 1 bit de précision par itération. Les valeurs $\arctan(2^{-i})$ et A_n sont générées lors de la synthèse puis intégrées à l'architecture en tant que constantes.

Le CORDIC permet de calculer différentes fonctions trigonométriques, selon les entrées qui lui sont appliquées [114]. Par exemple, la racine carrée est réalisable en utilisant le mode vectoriel. Les entrées suivantes permettent de calculer la valeur de $\sqrt{a}$:

$$x_0 = a + 0{,}25, \tag{4.17}$$

$$y_0 = a - 0{,}25, \tag{4.18}$$

$$z_0 = 0. \tag{4.19}$$

Cela permet après le nombre d'itérations nécessaires de récupérer $x_n = A_n\sqrt{v}$. Il y a cependant une contrainte pour assurer la convergence de l'algorithme. Il est en effet nécessaire que la valeur de a respecte la condition suivante :

$$0{,}5 \leq a < 2. \tag{4.20}$$

Cela demande donc d'effectuer une mise à l'échelle à l'entrée de l'opérateur pour toutes les données qui ne sont pas dans cet intervalle. Cela implique de faire la mise à l'échelle correspondante à la sortie afin de bien avoir le résultat souhaité.

Le CORDIC peut également être utilisé pour effectuer une division u/v avec le mode vectoriel et les entrées suivantes :

$$x_0 = v, \tag{4.21}$$

$$y_0 = u, \tag{4.22}$$

$$z_0 = 0. \tag{4.23}$$

Cette combinaison permet d'optenir en sortie $z_n = \arctan\left(\frac{u}{v}\right)$. Comme le développement limité de $\arctan x$ au premier ordre au voisinage de 0 est égal à x, la valeur de z_n obtenue peut donc donner le résultat souhaité si $u \ll v$. Cela demande par conséquent une connaissance *a priori* du résultat, ce qui n'est pas tout le temps possible, et de réaliser une mise à l'échelle si besoin. Une autre possibilité est de fixer $u = 1$ afin de calculer l'inverse de v en s'assurant que sa valeur soit assez élevée.

L'algorithme CORDIC est une méthode intéressante pour évaluer les fonctions trigonométriques et hyperboliques. En effet, il est simple d'en faire une implantation générique et pipelinée. En outre, les itérations ne nécessitent pas de multiplication et se résument qu'à des opérations d'additions et de décalages. Le gain d'un bit de précision par itération explique une latence élevée dans le cas où les entrées sont représentées sur beaucoup de bits. Enfin, le CORDIC souffre de conditions de convergence qui peuvent être restrictives. Ces raisons expliquent que cette méthode n'a pas été choisie pour implanter les opérateurs d'inverse et d'inverse de la racine carrée dont nous avons besoin.

4.2.3 Newton-Raphson

Une autre façon d'estimer une fonction est la méthode de Newton-Raphson. Il s'agit d'un algorithme permettant de trouver de manière itérative la racine d'une fonction dérivable. Il s'agit simplement, à partir d'une première approximation x_0, d'utiliser la tangente à la fonction à évaluer pour se rapprocher de la racine (Figure 4.5). La fonction de récurrence qui permet de calculer les différents x_i est la suivante :

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{4.24}$$

Il est possible d'utiliser cette méthode pour évaluer une fonction en un point précis en choisissant correctement la fonction $f(x)$. Dans le cas du calcul de $1/a$, la fonction sur

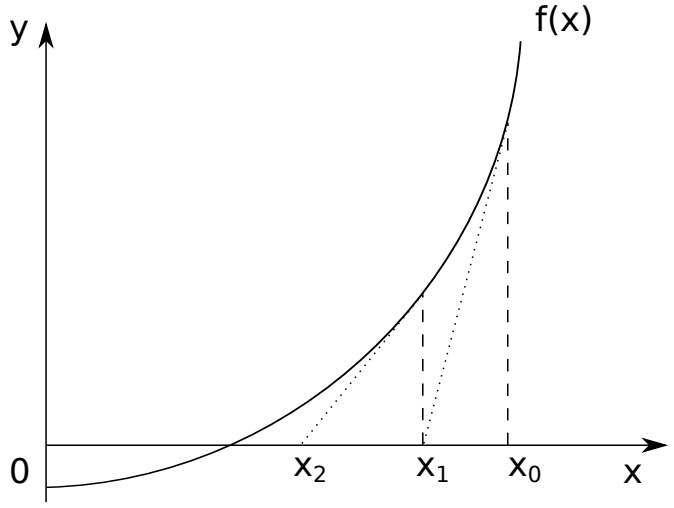


FIGURE 4.5 – Représentation de la méthode de Newton-Raphson pour trouver la racine de la fonction $f(x)$.

laquelle appliquer la méthode de Newton-Raphson est la suivante :

$$f(x) = \frac{1}{x} - a. \tag{4.25}$$

La fonction a bien pour racine la valeur $x = \frac{1}{a}$ et la formule d'itération devient :

$$x_{i+1} = x_i - \frac{\frac{1}{x_i} - a}{-\frac{1}{x_i^2}} \tag{4.26}$$

$$x_{i+1} = x_i(2 - ax_i). \tag{4.27}$$

Il s'agit donc d'une itération simple à implanter puisqu'elle ne contient que 2 multiplications et 1 soustraction. Pour le calcul de la racine carrée, les formules sont les suivantes :

$$f(x) = x^2 - a, \tag{4.28}$$

$$x_{i+1} = \frac{1}{2}(x_i + \frac{a}{x_i}), \tag{4.29}$$

et pour le calcul de l'inverse de la racine carrée :

$$f(x) = \frac{1}{x^2} - a, \tag{4.30}$$

$$x_{i+1} = \frac{x_i}{2}(3 - ax_i^2). \tag{4.31}$$

Les itérations nécessaires pour l'évaluation de la racine carrée sont complexes à mettre en place, puisqu'elles nécessitent une division. Il est donc plus pertinent d'utiliser la méthode de Newton-Raphson pour calculer $\frac{1}{\sqrt{a}}$ et en déduire par la suite $\sqrt{a}$ grâce à une multiplication.

La méthode de Newton-Raphson a une convergence quadratique. Cela signifie que la précision est approximativement doublée lors de chaque itération. Sur une représentation binaire, le nombre de bits significatifs est donc doublé à l'issue de chaque itération, ce qui rend la méthode viable pour des données représentées sur un grand nombre de bits. Il existe également une condition de convergence propre à chaque fonction évaluée :

$$\frac{1}{a} : 0 < ax_0 < 2, \tag{4.32}$$

$$\frac{1}{\sqrt{a}} : 0 < ax_0^2 < 3. \tag{4.33}$$

Il est donc essentiel de définir convenablement la première approximation x_0. Le but est de choisir une valeur telle que le produit évalué dans la condition de convergence concernée soit le plus proche de 1 possible. En effet, plus ce produit est proche de 1 et plus la première approximation est précise, ce qui signifie que moins d'itérations sont nécessaires pour atteindre la précision souhaitée.

Cela explique la statégie usuelle des implantations de la méthode de Newton-Raphson. Celle-ci demande d'avoir l'entrée sur un intervalle prédéterminé, comme $[1,\ 2[$ ou $[0{,}5,\ 1[$ [115, 116]. Cela implique donc une mise à l'échelle, comme pour les méthodes utilisant des LUTs. La similitude ne s'arrête pas à ce point puisque cette connaissance sur l'entrée permet de générer une première approximation à partir de LUTs. L'idée est d'avoir ainsi une valeur x_0 la plus précise possible afin de devoir exécuter le moins d'itérations possible. C'est pourquoi les recherches actuelles sur la méthode de Newton-Raphson se concentrent essentiellement sur l'optimisation du compromis entre la précision de x_0 et le besoin en mémoire pour calculer cette valeur.

4.2.4 Bilan des méthodes possibles

Plusieurs solutions sont possibles pour implanter un opérateur de calcul d'inverse ou d'inverse de racine carrée.

La première piste est de tabuler l'ensemble ou une partie de la fonction. L'idée de tabuler totalement une fonction est intéressante pour sa simplicité d'implantation. Cela n'est cependant pas réaliste quand le nombre de bits de représentation augmente puisque cela fait exploser le besoin en mémoire. Il est possible de trouver un compromis entre besoin en mémoire et complexité d'implantation en ne mémorisant que les données nécessaires pour effectuer une approximation linéaire ou d'ordre supérieur. Cela demande une connaissance sur l'entrée et nécessite une mise à l'échelle sur celle-ci pour l'amener dans un intervalle précis. Le fait de devoir choisir le meilleur compromis pour chaque taille d'entrée possible empêche la création d'une implantation générique.

L'algorithme CORDIC est une méthode itérative qui permet de gagner un bit de précision à chaque itération. Cela est problématique dans le cas de données représentées sur beaucoup de bits en ce qui concerne la latence de l'opérateur. Le CORDIC reste toutefois simple à mettre en œuvre et repose seulement sur des additions et des décalages des données.

La dernière méthode envisagée est l'algorithme de Newton-Raphson. Il s'agit d'une approche itérative qui permet de trouver la racine d'une fonction avec une convergence

Tableau 4.1 – Fréquence maximale en MHz du DSP et de la BRAM pour 3 familles de FPGA Xilinx.

	DSP	BRAM
Virtex-7	741	601
Virtex Ultrascale	741	660
Virtex Ultrascale +	891	825

quadratique. Cette méthode permet donc de doubler le nombre de bits de précision à chaque itération. Cela représente une bonne solution pour traiter des données représentées sur beaucoup de bits. Cette méthode itérative nécessite une première approximation. Celle-ci doit être choisie de sorte à respecter une condition de convergence. C'est pourquoi les implantations habituelles utilisent une mise à l'échelle de l'entrée ainsi que des LUTs pour générer la première approximation.

L'utilisation de l'algorithme de Newton-Raphson semble donc correspondre au mieux aux contraintes fixées. Il est facile de rendre l'implantation générique par rapport au nombre de bits sur lesquels est représentée l'entrée en jouant sur le nombre d'itérations réalisées. Cette généricité est plus difficile à obtenir pour les solutions basées sur des tabulations puisqu'il faut repenser toute l'organisation de la mémoire pour chaque nouvelle taille d'entrée.

4.3 Implantation sur FPGA Xilinx

4.3.1 Analyse de la structure du FPGA

Pour les opérations de calcul d'inverse et d'inverse de la racine carrée, les itérations de la méthode Newton-Raphson sont relativement simples. En effet, elles ne comportent que des multiplications et soustractions. Les multiplications sont effectuées sur les cellules DSP du FPGA. L'autre élément particulier à utiliser est la BRAM qui sert à stocker les données permettant de générer la première approximation. Il est alors intéressant d'étudier ces deux éléments afin de pouvoir déterminer ce qui peut limiter les performances de l'opérateur. Le Tableau 4.1 présente les fréquences maximales de chaque élément pour différentes familles de FPGA Xilinx [117, 118].

La cellule DSP est donc moins limitante que la BRAM pour la montée en fréquence de l'opérateur pour les FPGA Xilinx. Pour le constructeur concurrent Intel FPGA, les fréquences maximales de ces 2 éléments sur plate-forme Stratix 10 sont identiques et égales à 1 GHz [119]. Il est alors pertinent de proposer une implantation de l'algorithme de Newton-Raphson sans utilisation de BRAM en prévision des évolutions technologiques à venir. En effet, comme le débit du bus PCIe augmente au fil des nouveaux standards, il faut s'assurer que l'accélérateur proposé profite au mieux du débit disponible. De plus, il est possible de conserver un chemin critique minimal même pour des multiplications concernant des données représentées sur un grand nombre de bits. Les DSP présents sur les FPGA Xilinx gèrent au mieux une multiplication d'une opérande de 18 bits par une autre de 25 bits. Pour les multiplications en dehors de

cette limitation, il est possible de les subdiviser en produits partiels [120]. Cette découpe permet d'assurer une haute fréquence de fonctionnement en ajoutant quelques cycles de latence. Ces raisons nous ont poussés à proposer une nouvelle implantation de l'algorithme de Newton-Raphson qui n'utilise pas de BRAM. Cela repose donc sur une nouvelle façon de déterminer la première approximation. Le reste de cette partie présente l'architecture proposée et ses résultats sur le FPGA Xilinx Virtex-7 690T.

4.3.2 Impact sur la première estimation de la méthode Newton-Raphson

L'objectif ici est de proposer un moyen de générer la première approximation x_0 sans utiliser de données stockées pour les opérateurs de calcul de l'inverse et de l'inverse de la racine carrée. Cette première approximation doit respecter les conditions de convergence de chaque opérateur :

$$\frac{1}{a} : 0 < ax_0 < 2, \tag{4.34}$$

$$\frac{1}{\sqrt{a}} : 0 < ax_0^2 < 3. \tag{4.35}$$

Contrairement aux implantations habituelles qui visent à adapter l'entrée a à la méthode de génération de x_0 via une mise à l'échelle pour avoir une donnée dans un intervalle fixé, nous proposons ici la démarche inverse : adapter la génération de la première approximation à l'entrée de l'opérateur, comme cela est présenté ci-dessous.

De manière générale, la donnée a a une représentation de la forme $uQm.p$. Il s'agit donc d'une donnée non signée représentée avec m bits en partie entière et p bits en partie fractionnaire. Cette représentation peut être étendue au format $uQn.n$, avec $n = max(m,p)$. La donnée a est alors représentée de la façon suivante :

$$a = a_{n-1}a_{n-2}\ldots a_0 a_{-1}\ldots a_{-n}. \tag{4.36}$$

La sortie de l'opérateur de calcul d'inverse est également représenté selon le même format et a donc une précision absolue de 2^{-n}.

Soit j l'indice du premier bit de poids fort égal à 1. Il est possible d'établir l'inégalité suivante sur a :

$$2^j \leq a < 2^{j+1}. \tag{4.37}$$

Une valeur de x_0 peut ainsi facilement être déduite :

$$x_0 = 2^{-(j+1)}. \tag{4.38}$$

En combinant (4.37) et (4.38), l'inégalité suivante est alors obtenue :

$$0{,}5 \leq a \times x_0 < 1. \tag{4.39}$$

Cette valeur de x_0 respecte la condition de convergence. Il faut alors effectuer une opération de détection du premier bit à 1 (ou *Leading One Detector* (LOD)) pour déterminer l'indice j. Cette opération consiste simplement à recopier le bit à 1 ayant

le poids le plus fort et de laisser tous les autres bits à 0. Ainsi pour l'entrée a, la valeur $LOD(a)$ a seulement le bit d'indice j égal à 1. Une fois cette valeur obtenue, une symétrie permet d'avoir la valeur x_0 souhaitée.

Les techniques usuelles de calcul de LOD reposent sur un bloc de base de 4 ou 8 bits [121, 122]. Des assemblages de ces blocs sont ensuite réalisés pour créer un module de LOD des puissances de 2 supérieures. Ces solutions ne proposent donc pas d'architecture optimisée dans le cas où le nombre de bits considérés se situe entre 2 puissances de 2. Nous proposons ici une solution plus flexible sur cet aspect.

La première approximation peut en effet être générée en quelques étapes simples :

1. prendre le symétrique de a, noté $Sym(a)$;
2. effectuer un complément à 2 de $Sym(a)$;
3. effectuer un ET bit à bit entre $Sym(a)$ et son complément.

La démonstration de cette méthode est obtenue en partant de l'équation (4.37) qui permet de préciser le format de a :

$$a = 0 \dots 01 a_{j-1} \dots a_{-n}. \tag{4.40}$$

Ce qui permet de définir $Sym(a)$ ainsi que son complément à 2, $C2(Sym(a))$:

$$Sym(a) = a_{-n} \dots a_{j-1} 10 \dots 0, \tag{4.41}$$

$$C2(Sym(a)) = \overline{a_{-n}} \dots \overline{a_{j-1}} 10 \dots 0. \tag{4.42}$$

Au final, le ET bit à bit, noté $\&$, entre ces deux dernières valeurs permet d'obtenir la valeur souhaitée pour x_0 :

$$a_{-n} \dots a_{j-1} 10 \dots 0 \quad \& \quad \overline{a_{-n}} \dots \overline{a_{j-1}} 10 \dots 0 = 2^{-(j+1)}. \tag{4.43}$$

Cette méthode est réalisable peu importe le nombre de bits de représentation.

Une méthode de génération de x_0 générique et qui s'adapte directement à la valeur de a a donc été trouvée. L'équation (4.39) implique que cette première approximation a une précision d'un seul bit. Une itération de la méthode de Newton-Raphson double la précision à chaque itération. Il est donc pertinent de chercher si un circuit combinatoire simple permet de calculer x_0 avec 2 bits de précision. En effet, cela permet d'effectuer une itération en moins, ce qui représente 2 multiplications et 1 soustraction. Pour cela, il faut passer par une distinction de cas sur la valeur de a_{j-1}.

Si $a_{j-1} = 1$, l'équation (4.37) peut être réécrite :

$$1{,}5 \times 2^j \leq a < 2^{j+1}. \tag{4.44}$$

Si x_0 a toujours pour valeur $2^{-(j+1)}$, le produit devient :

$$0{,}75 \leq a \times x_0 < 1. \tag{4.45}$$

Cela signifie que dans ce cas, x_0 est déjà une approximation à 2 bits de précision.

Si $a_{j-1} = 0$, l'équation (4.37) devient :

$$2^j \leq a < 1{,}5 \times 2^j, \tag{4.46}$$

Ce qui donne :

$$0{,}5 \leq a \times x_0 < 0{,}75. \tag{4.47}$$

Il faut donc modifier la valeur de x_0 afin d'avoir les 2 bits de précision souhaités. Une possibilité est de multiplier x_0 par 1,5. Cela donne les résultats suivants :

$$x_0 = 2^{-(j+1)} + 2^{-(j+2)}, \tag{4.48}$$

$$0{,}75 \leq a \times x_0 < 1{,}125. \tag{4.49}$$

La condition suivante peut donc être déduite pour x_0 : le bit d'indice $-(j+1)$ doit être le complément de a_{j-1}. L'implantation de cette solution reste simple. Il est néanmoins nécessaire d'ajouter un bit de représentation pour prévoir le cas où $j = n$. Il est possible d'aller plus loin en faisant une nouvelle distinction de cas et aboutir à une première approximation avec 3 bits de précision. Ces 3 bits sont donnés par les formules suivantes :

$$x_{0_{-(j+1)}} = a_j, \tag{4.50}$$

$$x_{0_{-(j+2)}} = \overline{a_{j-1}}, \tag{4.51}$$

$$x_{0_{-(j+3)}} = \overline{a_{j-2}}. \tag{4.52}$$

Cette combinaison permet d'obtenir l'inégalité suivante : $0{,}875 \leq a \times x_0 < 1{,}125$. Dans la même logique que précédemment, cette approximation nécessite d'ajouter un autre bit de représentation. Il est donc possible de générer une première approximation pour l'opération d'inverse avec 3 bits de précision et sans avoir besoin de données stockées dans des éléments de mémorisation.

Une étude similaire peut être effectuée pour le calcul de l'inverse de la racine carrée. L'équation (4.37) sert toujours de point de départ. Une première valeur de x_0 respectant la condition de convergence peut être déterminée :

$$x_0 = 2^{-E(\frac{(j+1)}{2})}, \tag{4.53}$$

où la fonction $E(x)$ représente la fonction donnant la partie entière de l'argument. Cela implique l'égalité suivante pour x_0^2 :

$$x_0^2 = 2^{-2E(\frac{(j+1)}{2})}. \tag{4.54}$$

Une distinction de cas peut maintenant être effectuée selon la parité de l'indice j. Si l'indice j est impair, l'équation (4.54) devient :

$$x_0^2 = 2^{-(j+1)}. \tag{4.55}$$

Cela donne l'inégalité suivante :

$$0.5 \leq a \times x_0^2 < 1. \tag{4.56}$$

Cette valeur de x_0 respecte donc la condition de convergence dans le cas où l'indice j est impair.

Dans le cas où l'indice j est pair, l'équation (4.54) peut être réécrite de la façon suivante :

$$x_0^2 = 2^{-(j+1)}. \tag{4.57}$$

Et donc :

$$0.5 \leq a \times x_0^2 < 1, \tag{4.58}$$

ce qui correspond également à la condition de convergence de la méthode de Newton-Raphson pour l'inverse de la racine carrée. La valeur choisie pour x_0 permet donc d'assurer l'inégalité suivante : $0{,}5 \leq a \times x_0^2 < 2$. La précision de ce résultat peut être améliorée pour le cas où j est pair. La valeur x_0 peut en effet être déterminée de la façon suivante :

$$x_0 = 2^{-(\frac{j}{2}+1)} + 2^{-(\frac{j}{2}+2)}. \tag{4.59}$$

D'où :

$$x_0^2 = 0{,}5625 \times 2^{-j}, \tag{4.60}$$

$$0{,}5625 \leq a \times x_0^2 < 1{,}125, \tag{4.61}$$

Finallement, une première approximation de $\frac{1}{\sqrt{a}}$ peut être obtenue selon les formules suivantes :

$$x_0 = 2^{-\frac{j+1}{2}} \text{ si } j \text{ est impair}, \tag{4.62}$$

$$x_0 = 2^{-(\frac{j}{2}+1)} + 2^{-(\frac{j}{2}+2)} \text{ si } j \text{ est pair}. \tag{4.63}$$

Ces conditions permettent d'assurer l'inégalité $0{,}5 \leq a \times x_0^2 < 1{,}125$.

Au cours de ces travaux de thèse, il a été choisi d'utiliser des opérateurs permettant d'atteindre la plus haute fréquence de fonctionnement possible afin d'assurer un débit de traitement maximal sur FPGA. Cette logique répond à l'évolution actuelle du débit du bus PCIe qui double son débit de données à chaque nouvelle version du standard. Pour assurer cette montée en fréquence sur les FPGA Xilinx, une solution de génération de la première approximation d'opérateurs utilisant la méthode de Newton-Raphson a été proposée. Contrairement à la stratégie usuelle de l'état de l'art, aucune donnée ne doit être stockée en BRAM, ce qui permet une meilleure montée en fréquence. Cette méthode de génération est pipelinable et totalement générique par rapport au nombre de bits de représentation.

4.3.3 Implantation de l'inverse et de l'inverse de la racine carrée et comparaison à l'état de l'art

Les opérateurs d'inverse et d'inverse de la racine carrée ont été implantés en se basant sur la méthode de génération de la valeur x_0 présentée précédemment. La Figure 4.6 présente les premiers blocs de l'architecture de l'opérateur de calcul de l'inverse. Le bloc PA correspond à la première approximation à 3 bits de précision définie précédemment. Ce bloc prend pour entrée la valeur a, représentée sur $2n$ bits et la transmet au reste de l'architecture ainsi que la valeur x_0, représentée sur $2n+2$ bits. Tous les blocs d'itération de Newton-Raphson, nommés NR_i sont identiques. Le dernier de la chaine peut cependant être simplifié s'il n'est pas nécessaire de transmettre

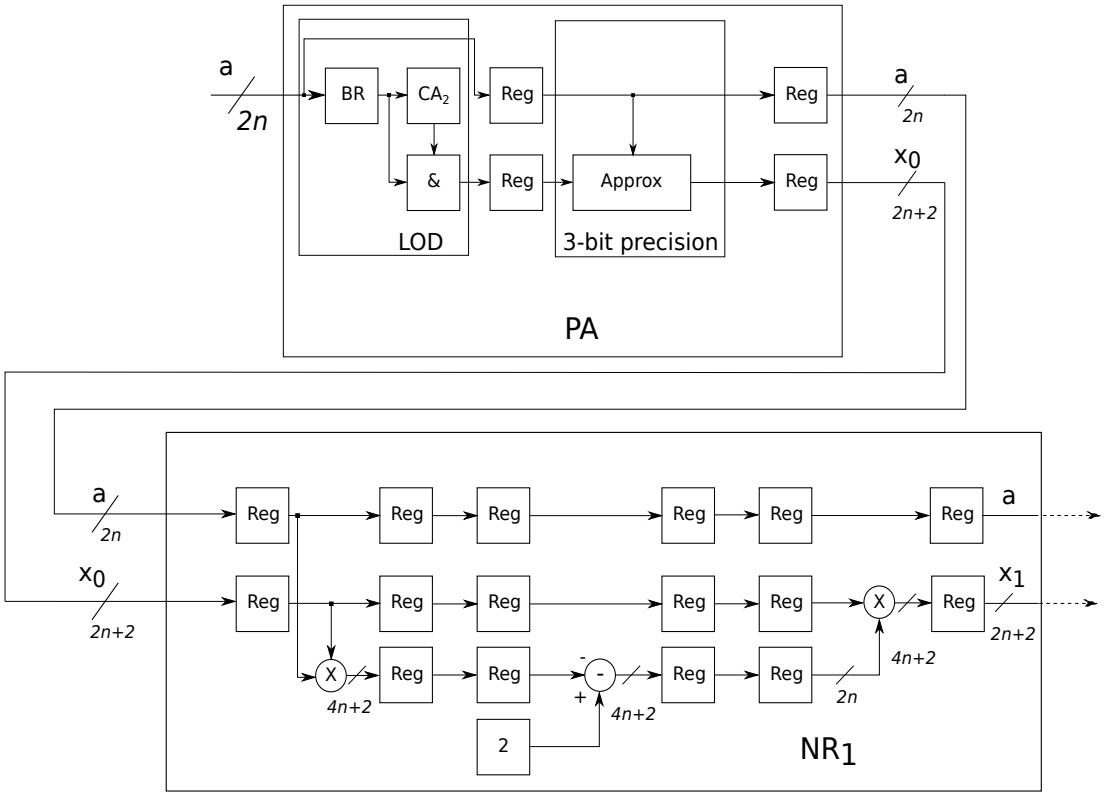


FIGURE 4.6 – Architecture de l'opérateur de calcul d'inverse.

la valeur de a en sortie de l'opérateur. Chaque bloc représente une implantation classique de la formule d'itération de la méthode de Newton-Raphson pour le calcul de l'inverse :

$$x_{i+1} = x_i(2 - ax_i). \tag{4.64}$$

Une évalutation de $1/a$ est donnée par x_m, sortie de la m^e itération. Des étages de registres doublés sont présents dans les blocs d'itération. Cela est nécessaire afin de profiter au mieux des registres présents dans la cellule DSP et ainsi permettre la montée en fréquence de l'opérateur. Cette implantation est totalement générique, ce qui fait que la chaine d'itérations accepte en entrée toute entrée a de format $2n$, ce qui est fixé par la méthode de génération de la première approximation.

Le Tableau 4.2 présente les résultats après placement et routage d'un opérateur de calcul de l'inverse sur 16 bits sur le FPGA Virtex-7 690T utilisé pour le démonstrateur. Deux configurations sont comparées : l'une avec la première approximation avec 1 seul bit de précision et l'autre avec 3 bits de précision.

Les 2 versions proposées atteignent une fréquence de fonctionnement de 740 MHz ce qui est bien la fréquence maximale d'une cellule DSP. La différence du nombre de DSP utilisés vient du nombre d'itérations nécessaires pour un calcul d'inverse sur 16 bits. En effet, avec une approximation à 3 bits de précision, seulement 3 itérations sont nécessaires. Cela donne directement le nombre de DSP utilisés (2 par itération), ainsi que le nombre de cycles pour produire le premier résultat (2 pour la première approximation et 6 par itération). Dans l'autre cas, où la première approximation ne demande qu'un cycle, il est conseillé de réaliser 5 itérations. En effet, la convergence de l'algo-

Tableau 4.2 – Résultats pour l'opérateur d'inverse 16 bits sur Virtex-7 690T.

	Architecture avec PA 1 bit	Architecture avec PA 3 bits
LUTs	159	111
Flip-Flop	325	240
DSP48E1	10	6
Latence (cycles)	31	20
Fréquence max (MHz)	740	740

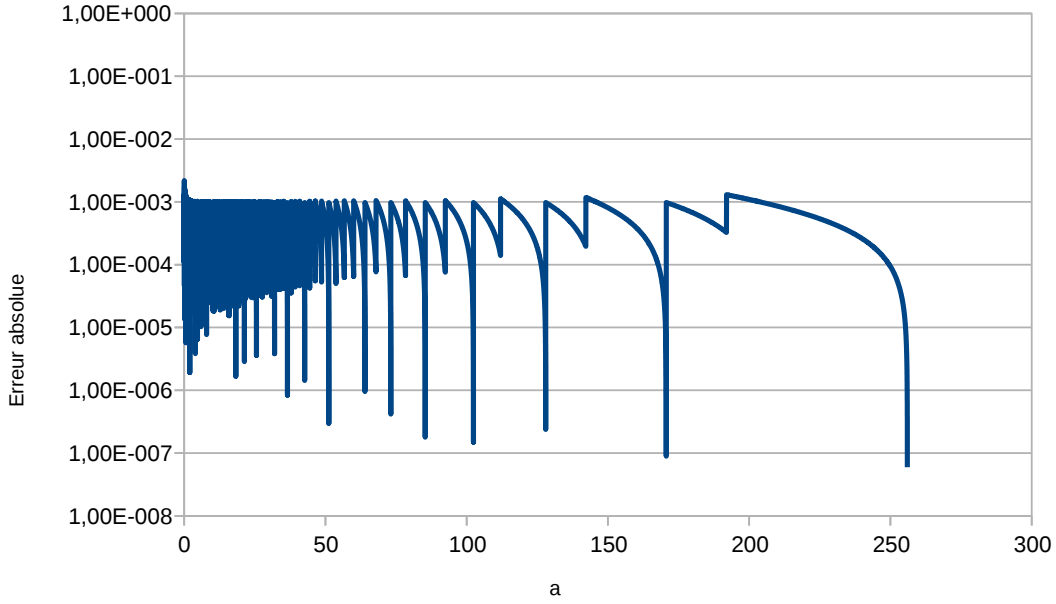


FIGURE 4.7 – Évolution de l'erreur absolue de l'opérateur de calcul d'inverse sur 16 bits.

rithme de Newton-Raphson double approximativement la précision à chaque itération. Il n'est donc pas suffisant de réaliser seulement 4 itérations pour assurer la précision visée. Ces résultats confirment l'intérêt d'améliorer la précision de la première approximation. Cela permet en effet de réduire les ressources utilisées par l'ensemble de l'opérateur, ainsi que d'en réduire la latence.

La précision de l'opérateur a été analysée (Figure 4.7). L'entrée a est une valeur non signée représentée sur 16 bits, avec 8 bits en partie entière et 8 bits en partie fractionnaire. Elle peut donc varier entre 0 et 256 - 2^{-8}. L'objectif est alors d'avoir une erreur inférieure à $2^{-8} = 3{,}90625.10^{-3}$ afin de profiter au maximum de la représentation utilisée. L'objectif de précision est bien atteint.

Une comparaison a été faite avec l'architecture pipelinée la plus récente trouvée dans l'état de l'art [123]. Cette architecture propose un opérateur de division en calculant l'inverse du diviseur avec des itérations de Newton-Raphson puis en effectuant une multiplication avec le dividende. La première approximation est réalisée de ma-

Tableau 4.3 – Division 16 bits sur Virtex-4 SX35.

	Architecture avec PA 3 bits	[123]
Slices	347	1478
LUTs	372	2091
Flip-Flop	568	1820
DSP48E1	7	7
Latence (cycles)	25	112
Mémoire	0	34Kb
Fréquence max (MHz)	294	294

nière polynomiale et utilise 34 kbits de données mémorisées. Les résultats de cette architecture ont été communiqués pour un FPGA Virtex-4 SX35. Notre architecture a donc été portée sur cette cible, en ajoutant une multiplication pour faire une division complète, afin de réaliser la comparaison présentée dans le Tableau 4.3. Les 2 architectures atteignent la fréquence de 294.1 MHz qui est la fréquence maximale des cellules DSP de cette génération de FPGA. Cela sert donc de point de comparaison, en plus du nombre de DSP utilisés qui est également identique. Notre proposition utilise 82% de LUT et 69% de Flip-Flop en moins que [123]. Le premier échantillon en sortie de l'opérateur est disponible environ 4,5 fois plus rapidement avec notre solution.

Ces travaux sur l'opérateur de calcul de l'inverse ont été communiqués dans la revue *IEEE Signal Processing Letters* [124] ainsi que lors du colloque du GRETSI 2017 [125].

La Figure 4.9 présente les premiers blocs de l'opérateur dédié au calcul de l'inverse de la racine carrée. Le principe est similaire à celui de l'opérateur de calcul d'inverse présenté précédemment. Les étages de registres doublés ne sont pas représentés par souci de lisibilité. Les blocs d'itération traitent les données en 10 cycles et le bloc de première approximation en 4 cycles. Cela explique la latence globale de 34 cycles pour un opérateur sur 16 bits, où 3 itérations sont nécessaires. Les résultats d'utilisation de ressources après placement et routage sont donnés dans le Tableau 4.4. La fréquence de 740 MHz est également atteinte. L'architecture est également pipelinée et générique.

La précision de l'opérateur de calcul d'inverse de la racine carrée a également été analysée (Figure 4.8). L'erreur est beaucoup plus proche du seuil de 2^{-8} que l'opérateur d'inverse. Cela peut s'expliquer par la multiplication supplémentaire dans la formule d'itération qui favorise la propagation d'une approximation.

L'opérateur a été comparé à une autre architecture de l'état de l'art [126] sur Virtex-6. Cette dernière prend en entrée une valeur sur 15 bits et effectue une seule itération suite à une première approximation déterminée grâce à des données stockées en mémoire. Cette architecture utilise donc moins de ressources et a moins de latence que celle que nous proposons (Tableau 4.5), mais en contrepartie, la fréquence maximale de fonctionnement est 1,8 fois inférieure. Les travaux sur cet opérateur ont fait l'objet d'une publication à NEWCAS 2017 [127].

Afin de répondre au défi du débit de traitement de l'accélérateur à concevoir, l'im-

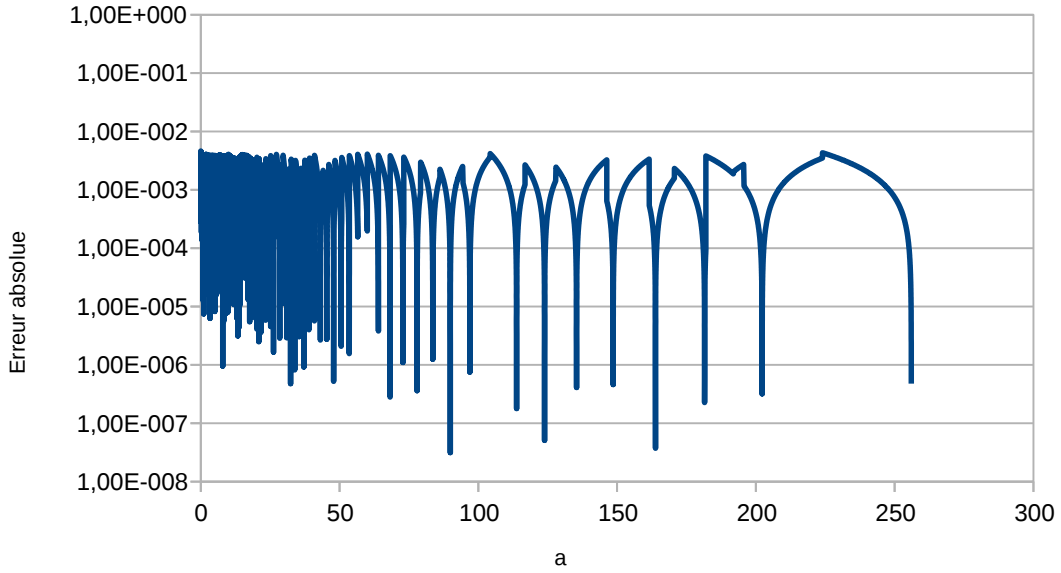


FIGURE 4.8 – Évolution de l'erreur absolue de l'opérateur de calcul d'inverse de la racine carrée sur 16 bits.

Tableau 4.4 – Résultats pour l'opérateur d'inverse de la racine carrée sur Virtex-7 690T.

Flip-Flop	261
LUTs	221
DSP48E1s	9
Latence	34
Fréquence maximale (MHz)	740

Tableau 4.5 – FPGA Implementation results on Virtex-6.

	Architecture proposée	[126]
Flip-Flop	258	160
LUTs	203	160
DSP48E1s	9	4
BRAMs	0	1
Latence	34	5
Fréquence maximale (MHz)	639	352

plantation de la méthode de Newton-Raphson a été modifiée afin de ne plus avoir besoin de mémorisation de données. Cette modification permet d'avoir des opérateurs génériques de calcul d'inverse et d'inverse de la racine carrée pipelinés, pouvant atteindre une fréquence de fonctionnement de 740 MHz sur des opérateurs de 16 bits sur FPGA Virtex-7. Ces opérateurs sont facilement adaptables à toute taille de l'entrée. Ce

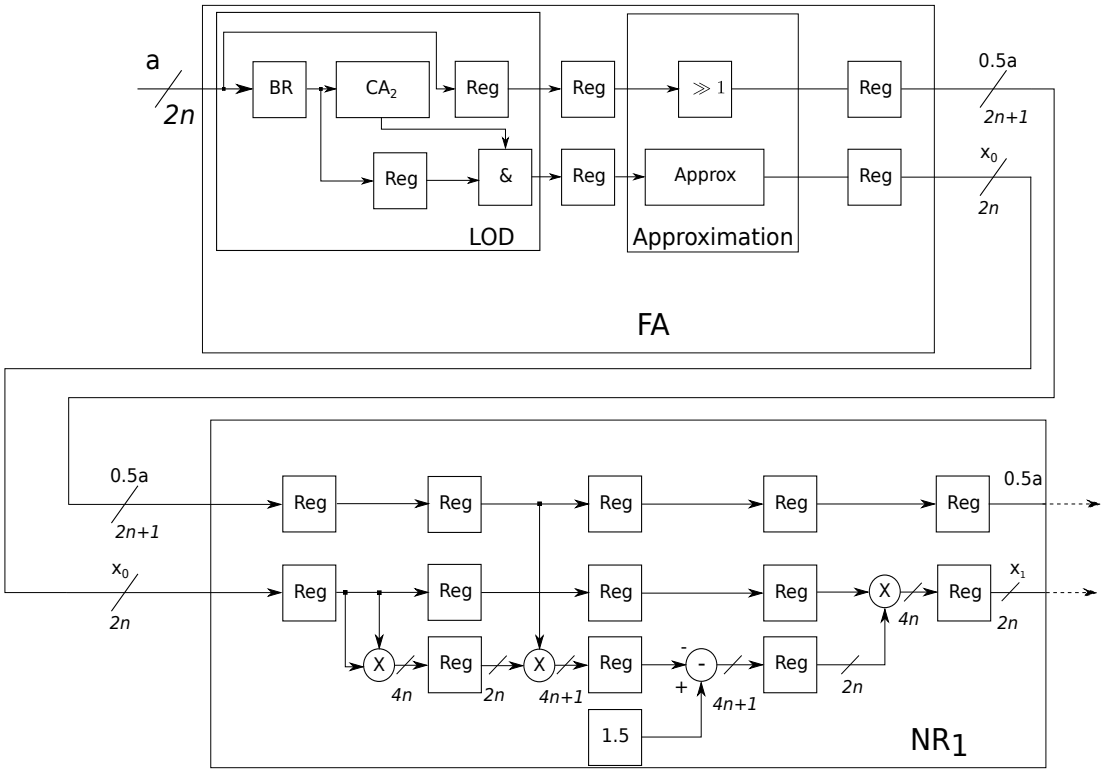


FIGURE 4.9 – Architecture de l'opérateur de calcul d'inverse de la racine carrée.

sont donc les opérateurs idéaux pour être intégrés dans l'accélérateur de l'algorithme présenté dans le Chapitre 2.

4.4 Implantation ASIC

Les opérateurs proposés répondent bien aux contraintes fixées pour le prototypage de l'accélérateur de la fonction d'intégration sur plate-forme FPGA. Il est maintenant intéressant de se projeter à plus long terme et déterminer si l'approche proposée est toujours valide pour une intégration ASIC qui n'a pas les mêmes contraintes physiques qu'un FPGA.

4.4.1 Multiplication et bloc mémoire : analyse des temps de propagation

La flot de conception ASIC étudié ici concerne la technologie CMOS 65nm de ST-Microelectronics. L'étude se fait une nouvelle fois pour des opérateurs de 16 bits, qui est le format classique dans l'état de l'art. L'objectif est de comparer la solution présentée précédemment qui n'utilise que des multiplieurs et de la logique combinatoire et séquentielle élémentaire, avec une solution utilisant en plus un bloc de mémoire. Des synthèses successives des opérateurs de calcul de l'inverse et de l'inverse de la racine carrée conçus pour FPGA ont abouti à un chemin critique minimal de 593 ps, ce qui correspond à une fréquence de fonctionnement de 1,686 GHz. Ce chemin critique

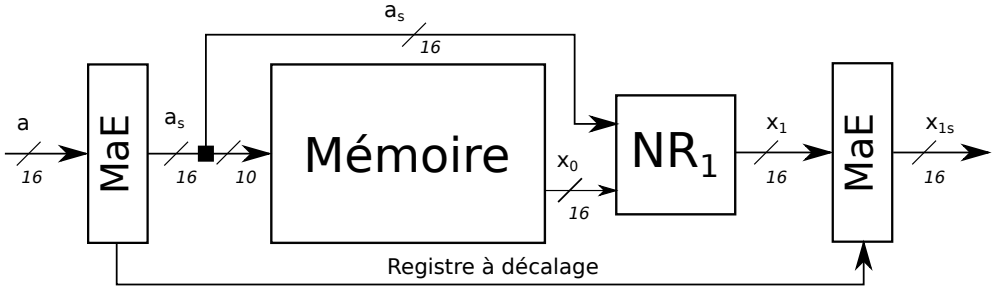


FIGURE 4.10 – Architecture de l'opérateur de calcul d'inverse utilisant une mémoire.

Tableau 4.6 – Caractéristiques de la mémoire SRAM 1024 x 16 de STMicroelectronics.

Caractéristique	Valeur
Temps de réponse (ps)	630
Surface (μm^2)	17 781

se situe au niveau d'une multiplication, qui reste alors l'élément limitant en terme de fréquence pour un ASIC.

La solution avec mémoire envisagée est représentée en Figure 4.10. Cette architecture nécessite 2 blocs de mise à l'échelle (MaE) avec l'opération à exécuter transmise via un registre à décalage. La mémoire est adressée avec les 10 bits de poids fort de la valeur a_s qui est la valeur de a mise convenablement à l'échelle. Cette mémoire fournit en sortie une première approximation sur 16 bits qui doit passer par un bloc d'itération de l'algorithme de Newton-Raphson. Les caractéristiques de la mémoire fournies par STMicroelectronics sont données dans le Tableau 4.6. L'utilisation d'une mémoire pénalise donc plus la fréquence maximale du circuit qu'un multiplieur. Toutefois, il y a une autre métrique à prendre en compte. En effet, la performance de calcul doit être mise en balance avec la surface nécessaire. Une métrique intéressante est donc le nombre d'opérations par seconde et par millimètre carré (Gop/s/mm^2). Il est donc nécessaire de comparer les 2 méthodes par rapport à cette métrique.

4.4.2 Résultats et conséquences

La comparaison présentée ici concerne l'opérateur de calcul d'inverse. Les résultats sont présentés dans le Tableau 4.7. L'utilisation d'une mémoire demande d'effectuer moins d'itérations et le gain en surface permet d'avoir une meilleure performance par millimètre carré. Cette solution est donc optimale pour être intégrée dans une architecture dont la fréquence de fonctionnement est moins élevée. A l'inverse, si l'opérateur doit être intégré dans une architecture à la fréquence plus élévée, il vaut mieux utiliser la version sans bloc mémoire qui est moins limitante pour l'ensemble de l'architecture. Une comparaison a été faite avec une autre architecture de calcul d'inverse en technologie 65nm [128]. Elle n'utilise pas la méthode de Newton-Raphson mais un interpolateur de Chebyshev cubique. Il s'agit de la seule architecture trouvée dans l'état de l'art pour cette technologie et sur une entrée de 16 bits. L'interpolateur cubique utilise 4 blocs mémoire. La comparaison avec l'opérateur utilisant l'algorithme de Newton-

Tableau 4.7 – Comparaison entre 2 opérateurs de calcul d'inverse utilisant la méthode de Newton-Raphson, avec ou sans bloc mémoire.

Architecture	Sans mémoire	Avec mémoire
Mémoire	0	1024 x 16
Fréquence maximale (GHz)	1,686	1,587
Surface (μm^2)	44 273	38 527
Gop/s/mm^2	38	41

Tableau 4.8 – Comparaison de l'opérateur d'inverse 16 bits à l'état de l'art.

Architecture	Newton-Raphson avec mémoire	[128]
Fréquence maximale (GHz)	1,587	0,820
Surface (μm^2)	38 527	95 700
Gop/s/mm^2	41	9

Raphson est présentée dans le Tableau 4.8. L'utilisation de la méthode de Newton-Raphson permet donc d'améliorer les performances de l'état de l'art par un facteur 4. Cet algorithme itératif est donc une bonne solution pour l'implantation d'opérateurs de calcul d'inverse et d'inverse de la racine carrée sur plate-forme ASIC. Cette étude a fait l'objet d'une publication à ISCAS 2018 [129].

4.5 Conclusion

L'implantation de fonctions non-linéaires comme l'inverse ou l'inverse de la racine carrée n'est pas immédiate et demande une architecture particulière. Dans le cadre des travaux de thèse présentés ici, les contraintes de pipeline, de fréquence de fonctionnement ainsi que de généricité augmentent la difficulté d'implantation. L'algorithme de Newton-Raphson est la méthode qui répond au mieux à ces contraintes, notamment grâce à son aspect itératif et sa convergence quadratique.

Les mises en œuvre usuelles de l'algorithme de Newton-Raphson utilisent un bloc mémoire pour générer la première approximation. Si cela permet de réduire le nombre d'itérations nécessaires, le bloc mémoire limite néanmoins la fréquence de fonctionnement de l'opérateur sur les plate-formes FPGA Xilinx, tout en n'ayant aucun impact sur les produits de Intel FPGA. Nous avons donc proposé des opérateurs n'utilisant pas de bloc mémoire et atteignant ainsi la fréquence maximale permise par les blocs chargés des multiplications. Ces opérateurs sont génériques et pipelinés. Ils répondent donc bien aux contraintes pour l'accélération de l'algorithme présenté dans le Chapitre 2. De plus, le calcul de l'inverse de la racine carrée est un élément clé en électromagnétisme, notamment pour calculer les potentiels électrostatiques. C'est pourquoi la description en code VHDL des opérateurs présentés dans ce Chapitre ont été mis à la disposition de la communauté scientifique sous forme de projet libre [130].

Les opérateurs proposés ont donc été validés pour être utilisé sur plate-forme FPGA. Dans un souci de projection à plus long terme, leurs performances ont été analysées pour une intégration ASIC de technologie CMOS 65nm. Sur 16 bits, la fréquence de fonctionnement de ces opérateurs atteint 1,6 GHz. La surface d'occupation reste suffisamment restreinte pour multiplier par 4 le nombre d'opérations par seconde et par millimètre carré par rapport à l'état de l'art. Les opérateurs proposés conviennent donc également pour un ASIC et sont intégrés dans l'accélérateur global qui est présenté dans le Chapitre suivant.

Chapitre 5

Implantation d'un accélérateur pour l'imagerie cérébrale

5.1 Introduction

Dans ce dernier chapitre, l'implantation de l'accélérateur dédié à l'algorithme d'intégration en 3 dimensions sur un triangle présenté dans le Chapitre 2 est détaillée. L'objectif ici est de démontrer que l'accélération matérielle par circuit dédié offre des performantes inégalées pour développer les recherches sur l'ICM basée sur le problème inverse de l'EEG.

Dans un premier temps, la mise en œuvre de l'architecture est présentée. Les résultats du prototype sur plate-forme FPGA et les perspectives ASIC sont ensuite abordés.

5.2 Architecture de l'accélérateur

L'architecture conçue vise à accélérer la fonction d'intégration présentée dans la partie 2.2. La Figure 5.1 est rappelée pour faciliter la compréhension. L'algorithme se déroule en 3 grandes phases. La première consiste à calculer les coordonnées de la base locale à partir des coordonnées du triangle et du point d'observation. Vient ensuite le calcul des distances et des projections (Figure 5.1c). La dernière étape permet de calculer l'intégrale définie par la formule suivante :

$$I = -|w_0| \sum_{i=1}^{3} \beta_i + \sum_{i=1}^{3} t_i^0 f_{2i}, \tag{5.1}$$

avec

$$\beta_i = \arctan \frac{{t_i}^0 {s_i}^+}{({R_i}^0)^2 + |w_0| {R_i}^+} - \arctan \frac{{t_i}^0 {s_i}^-}{({R_i}^0)^2 + |w_0| {R_i}^-}, \tag{5.2}$$

$$f_{2i} = \ln \left(\frac{{R_i}^+ + {s_i}^+}{{R_i}^- + {s_i}^-} \right). \tag{5.3}$$

Cet algorithme fait appel aux opérations non-linéaires suivantes : inverse, inverse de la racine carrée, arctangente et logarithme népérien. La structure de l'accélérateur de la fonction suit la même logique (Figure 5.2). L'opérateur est totalement pipeliné.

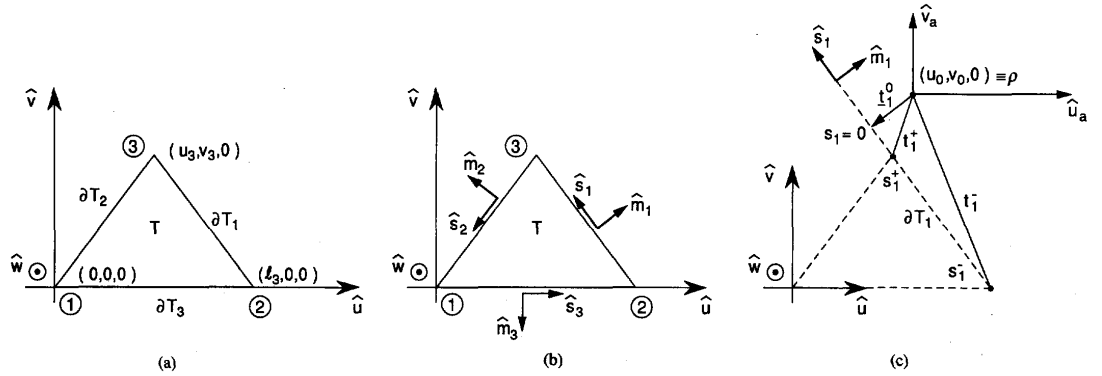


FIGURE 5.1 – (a) Définition de la base locale selon le triangle considéré. (b) Définition des vecteurs tangents $\hat{s}$ et normaux $\hat{m}$ sur le contour du triangle. (c) Exemple de la distance entre le côté 1 et ρ, la projection du point d'observation [8] ;

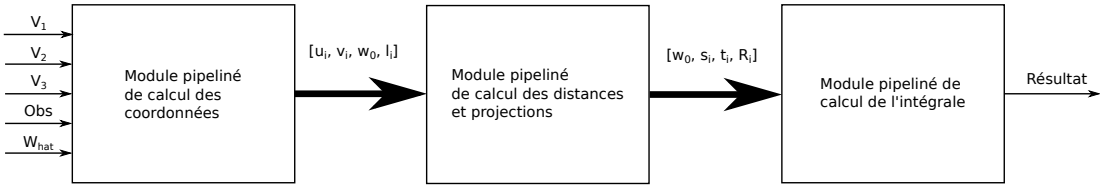


FIGURE 5.2 – Structure de l'accélérateur conçu.

Le chapitre précédent a permis de présenter notre contributions pour les opérateurs de calcul de l'inverse et de l'inverse de la racine carrée qui sont instanciés à plusieurs reprises au sein de l'accélérateur. Dans cette partie, d'autres points clés, comme les autres fonctions non-linéaires ou l'organisation de certaines opérations, sont décrits en se cantonnant à un usage de l'état de l'art mais en conservant la philosophie d'implantation développée au Chapitre 4.

5.2.1 Arctangente et logarithme népérien

L'étude effectuée dans le Chapitre 2 de l'algorithme à accélérer a permis de mettre en lumière la présence de fonctions non-linéaires. Le Chapitre 4 présente les solutions retenues concernant la gestion des divisions et des racines carrées. Les 2 autres fonctions à traiter sont l'arctangente et le logarithme népérien. L'objectif étant toujours d'opérer les calculs à des fréquences élevées, il a été décidé de ne pas utiliser d'architectures demandant l'utilisation de BRAM. L'algorithme de Newton-Raphson n'est pas utilisable ici, puisque les formules d'itération sont trop complexes à implanter car elles font appel à d'autres fonctions non-linéaires. En effet, la formule d'itération pour estimer le logarithme népérien demande de calculer une exponentielle :

$$x_{i+1} = x_i + a\exp(-x_i). \tag{5.4}$$

Pour l'arctangente, c'est la tangente qui est nécessaire :

$$x_{i+1} = x_i - \frac{\tan(x_i) - a}{1 + \tan^2(x_i)}. \tag{5.5}$$

La solution finalement retenue est d'implanter une solution de l'état de l'art en utilisant l'algorithme CORDIC présenté dans le Chapitre précédent.

Pour l'arctangente, comme indiqué dans le Chapitre 2, il s'agit plus précisément de la fonction $atan2(y,x)$ qui doit être implantée. Cette fonction est une variante de l'arctangente classique qui prend en entrée 2 arguments. Pour $y \neq 0$, cette fonction suit la définition suivante :

$$atan2(y,x) = \begin{cases} \varphi sgn(y) & \text{si} \quad x > 0, \\ \frac{\pi}{2} sgn(y) & \text{si} \quad x = 0, \\ (\pi - \varphi) sgn(y) & \text{si} \quad x < 0, \end{cases} \tag{5.6}$$

où φ est l'angle dans l'intervalle $[0,\frac{\pi}{2}[$ tel que $\tan\varphi = \left|\frac{y}{x}\right|$ et sgn est la fonction signe. Pour $y = 0$, la définition devient :

$$atan2(0,x) = \begin{cases} 0 & \text{si} \quad x > 0, \\ \text{Non défini} & \text{si} \quad x = 0, \\ \pi & \text{si} \quad x < 0. \end{cases} \tag{5.7}$$

Cette variante est intéressante car elle permet d'éviter une division dans le cas du calcul des valeurs β_i (5.2). Les dénominateurs sont toujours strictement positifs. Cela permet d'avoir $atan2(y,x) = atan(\frac{y}{x})$ dans ce cas d'application. Le CORDIC trigonométrique en mode vectoriel permet d'implanter la fonction $atan2$. En effet, les sorties obtenues après n itérations sont :

$$x_n = A_n \sqrt{x_0^2 + y_0^2}, \tag{5.8}$$

$$y_n = 0, \tag{5.9}$$

$$z_n = z_0 + \arctan\frac{y_0}{x_0}. \tag{5.10}$$

La valeur $atan2(y,x)$ est alors obtenue en utilisant les entrées suivantes pour l'implantation du CORDIC :

$$x_0 = x, \tag{5.11}$$

$$y_0 = y, \tag{5.12}$$

$$z_0 = 0. \tag{5.13}$$

L'unique contrainte est la nécessité d'avoir le même format de représentation pour x et y, c'est-à-dire le même nombre de bits en partie entière et en partie fractionnaire. Le fait de savoir que la valeur x est strictement positive permet d'exclure tout risque de ne pas être dans l'intervalle de convergence du CORDIC trigonométrique en mode vectoriel [131]. Il s'agit donc d'une méthode permettant la conception rapide d'un opérateur de calcul d'arctangente.

Pour le logarithme népérien, il faut utiliser le CORDIC hyperbolique en mode vectoriel. Les formules diffèrent du CORDIC trigonométrique :

$$x_{i+1} = x_i - y_i d_i 2^{-i}, \tag{5.14}$$

$$y_{i+1} = y_i - x_i d_i 2^{-i}, \tag{5.15}$$

$$z_{i+1} = z_i + d_i \tanh^{-1}(2^{-i}), \tag{5.16}$$

avec

$$d_i = \begin{cases} 1 & \text{si} \quad x_i y_i \geq 0, \\ -1 & \text{si} \quad x_i y_i < 0. \end{cases} \tag{5.17}$$

Ce qui donne également de nouvelles formules pour les résultats finaux :

$$x_n = A_n \sqrt{x_0^2 - y_0^2}, \tag{5.18}$$

$$y_n = 0, \tag{5.19}$$

$$z_n = z_0 + \tanh^{-1} \frac{y_0}{x_0}, \tag{5.20}$$

$$A_n = \prod_{i=0}^{n-1} \sqrt{1 - 2^{-2i}}. \tag{5.21}$$

Le CORDIC hyperbolique a la particularité d'avoir une condition supplémentaire afin de converger. En effet, il est nécessaire de répéter toutes les itérations dont l'indice est de la forme $i = 3k + 1$ avec k un entier naturel.

Il est ainsi possible d'évaluer la valeur $\ln(a)$ en utilisant la relation suivante :

$$\ln(\frac{1+x}{1-x}) = 2 \tanh^{-1}(x). \tag{5.22}$$

Il faut donc appliquer en entrée du CORDIC les valeurs suivantes :

$$x_0 = a + 1, \tag{5.23}$$

$$y_0 = a - 1, \tag{5.24}$$

$$z_0 = 0. \tag{5.25}$$

Dans le cas du CORDIC hyperbolique, il faut considérer la condition de convergence. Celle-ci est donnée par un paramètre nommé θ_{max} [131] :

$$|\tanh^{-1}(\frac{y_0}{x_0})| \leq \theta_{max} \approx 1{,}1182, \tag{5.26}$$

$$|\frac{y_0}{x_0}| \leq \tanh(\theta_{max}) = 0{,}8069. \tag{5.27}$$

Cela donne alors une condition sur le ratio $\frac{a-1}{a+1}$, sachant que dans le cas de l'algorithme à accélérer, la donnée a est strictement positive. Finalement une contrainte pour la donnée a peut être déterminée :

$$a \leq \frac{1 + \tanh(\theta_{max})}{1 - \tanh(\theta_{max})} = 9{,}3596. \tag{5.28}$$

Tableau 5.1 – Evolution de θ_{max} et de a en fonction du nombre de pré-itérations.

Nombre de pré-itérations	θ_{max}	a_{max}
0	1,1182	9,3596
1	2,0911	65,5137
2	3,4451	982,696
3	5,1622	30463
4	7,2171	1932258

La contrainte pour le calcul de $\ln(a)$ est donc forte et la mise à l'échelle n'est pas aussi simple que pour l'opération d'inverse ou l'inverse de la racine carrée. En effet, dans le cas du logarithme, la mise à l'échelle utilise la relation suivante :

$$\ln(a \times 2^i) = \ln(a) + \ln(2^i). \tag{5.29}$$

Il faut alors calculer et mémoriser tous les $\ln(2^i)$ possibles pour pouvoir gérer la mise à l'échelle en sortie de l'opérateur de calcul du logarithme. Cette solution ne correspond donc pas à notre volonté d'éviter l'utilisation de bloc mémoire afin de conserver la perspective de montée en fréquence. Une autre méthode pour résoudre le problème de convergence est d'étendre l'intervalle de validité pour a.

En 1991, il a été montré que le domaine de convergence du CORDIC hyperbolique peut être étendu en ajoutant des pré-itérations [131]. Les formules d'itérations effectuées, pour $i \leq 0$, sont les suivantes :

$$x_{i+1} = x_i - y_i d_i (1 - 2^{i-2}), \tag{5.30}$$

$$y_{i+1} = y_i - x_i d_i (1 - 2^{i-2}), \tag{5.31}$$

$$z_{i+1} = z_i + d_i \tanh^{-1}(1 - 2^{i-2}), \tag{5.32}$$

avec

$$d_i = \begin{cases} 1 & \text{si} \quad x_i y_i \geq 0, \\ -1 & \text{si} \quad x_i y_i < 0. \end{cases} \tag{5.33}$$

Ces itérations supplémentaires permettent d'augmenter la valeur de θ_{max} et ainsi la valeur maximale pour l'entrée a, a_{max} (Tableau 5.1). Il est donc possible de considérablement augmenter le domaine de convergence en n'ajoutant que très peu d'itérations. Le CORDIC hyperbolique répond donc à nos besoins pour l'implantation de l'opérateur du logarithme népérien.

5.2.2 Implantation de l'accélérateur

La description de la totalité de l'accélérateur a été réalisée en VHDL. Il a été décidé, après concertation avec l'équipe travaillant sur l'aspect algorithme, de représenter les entrées de l'accélérateur sur 15 bits. Ceux-ci sont distribués en 1 bit de signe, 4 bits

Tableau 5.2 – Amplitude maximale des valeurs de projections.

Valeur	Amplitude maximale
w_0	10
s_1^+	300
s_1^-	300
s_2^+	200
s_2^-	200
s_3^+	100
s_3^-	100
t_1^0	300
t_2^0	200
t_2^0	100
t_1^+	150
t_1^-	150
t_2^+	150

en partie entière et 10 bits en partie fractionnaire. Cela permet de représenter des entrées de l'algorithme, prévues comme comprises dans l'intervalle $[-10, 10]$, avec une précision de 10^{-3}. Les entrées de l'algorithme sont les coordonnées des points du triangle, du point d'observation ainsi que du vecteur $\hat{w}$ de la base locale. Cela représente donc un total de 225 bits de données d'entrée. Le résultat de l'intégrale est quant à lui représenté sur 70 bits : 1 bit de signe, 44 bits en partie entière et 25 bits en partie fractionnaire. Cela permet de représenter la sortie avec une précision de 10^{-7}. Pour le nombre de bits en partie entière, l'évolution du format de réprésentation au fil des opérations a été suivie. Il a cependant été nécessaire de limiter cette évolution afin de ne pas aboutir à des données représentées sur plus de 100 bits. Les amplitudes maximales pour les valeurs de projections ont été déterminées de façon logicielle sur la base de simulations à partir de données mesurées. (Tableau 5.2). Cette connaissance sur ces données permet donc de cadrer le nombre de bits de représentation pour leur partie entière, ce qui permet au final de réduire les ressources nécessaire pour implanter l'algorithme. Finalement, la valeur maximale du nombre de bits de représentation d'une donnée est de 83. L'architecture est totalement pipelinée et peut fournir un résultat de calcul d'intégral à chaque cycle d'horloge. La latence du circuit est de 255 cycles.

Les opérateurs décrits précédemment sont utilisés plusieurs fois. Le Tableau 5.3 liste le nombre d'instanciations de chaque opérateur ainsi que la taille des données traitées. La représentation des données traitées valide donc le fait d'avoir choisi des structures itératives. En effet, les autres possibilités demandent des capacités de mémorisation trop importantes pour ces dimensions.

Afin d'assurer une bonne montée en fréquence de l'algorithme. une attention particulière a été portée sur l'atomisation des opérations. Cela signifie qu'au maximum une opération élémentaire (addition, soustraction, multiplication) est réalisée entre 2 re-

Tableau 5.3 – Liste des instanciations des opérateurs non-linéaires

Opérateur	Taille des entrées et sorties (bits)	Nombre d'instanciations
$\ln(a)$	53	6
$\arctan(\frac{y}{x})$	43	6
$\frac{1}{a}$	40	6
$\frac{1}{\sqrt{a}}$	40	7
$\frac{1}{\sqrt{a}}$	42	1

Tableau 5.4 – Utilisation des ressources du FPGA Virtex-7 690T par l'accélérateur

Ressources	Utilisées	Disponibles	Pourcentage d'utilisation
LUTs	166521	433200	38,4
Flip-Flop	90852	866400	10,5
BRAM	0	1470	0
DSP	1145	3600	31,2

gistres. Par exemple, il faut 2 cycles d'horloge pour réaliser l'opération $a = b + c + d$. Le premier permet de faire une somme partielle, $b + c$ tout en registrant la valeur de d. La somme finale est réalisée au cycle suivant. Un autre exemple moins évident est l'opération $a = -b \times c$. En effet, cette opération doit également être séparée en 2 puisque l'inversion de signe compte également comme une opération élémentaire. Cela s'explique par l'addition nécessaire pour effectuer un complément à 2.

5.3 Résultats d'implantation sur FPGA et ASIC

Cette partie présente les résultats obtenus pour l'accélérateur de la fonction d'intégration en 3 dimensions sur un triangle, présentée en partie 2.2. Une plate-forme FPGA et une plate-forme ASIC ont été considérées.

5.3.1 FPGA

Le FPGA considéré est le Virtex-7 690T de Xilinx. Celui est disponible sur la carte NetFPGA SUME qui propose une connexion PCIe x8 de génération 3. Il s'agit d'une carte fiable, répandue dans le milieu académique avec un bon support et une forte communauté d'usagers. Le Tableau 5.4 présente les ressources utilisées par l'accélérateur par rapport aux capacités du FPGA. Les éléments utilisés en plus grande proportion sont les LUTs et les DSP. Le taux d'utilisation des DSP est une métrique intéressante à étudier car ces éléments sont moins uniformément distribués au sein du FPGA que les *slices*, éléments configurables de base comportant les LUTs et les *Flip-Flop*. En effet, les DSP, ainsi que les BRAM sont disposés en colonne dans le FPGA. Entre ces colonnes se trouvent les *slices*. Quand une grande proportion des DSP est utilisée, les

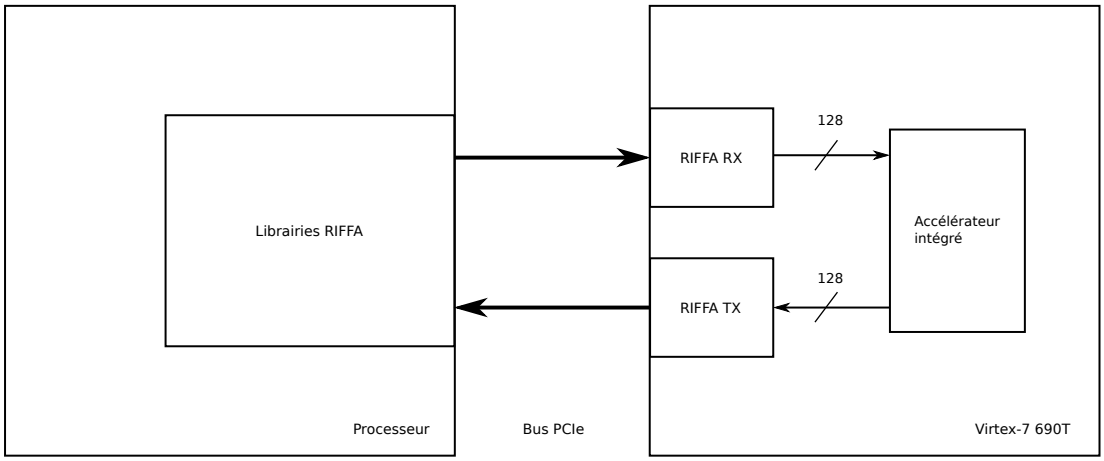


FIGURE 5.3 – Cheminement des données au sein du prototype.

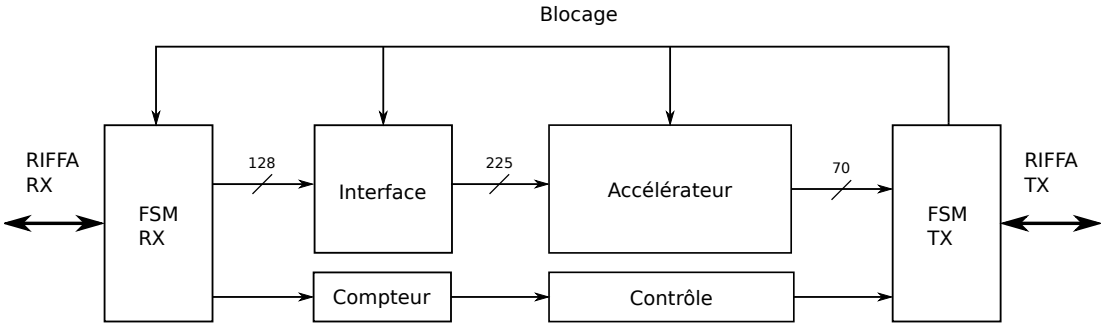


FIGURE 5.4 – Architecture du bloc Accélérateur intégré pour l'utilisation d'un lien PCIe de génération 1.

cellules disponibles peuvent être éloignées des autres cellules où s'effectue le traitement des données. Cela peut donc amener des difficultés à respecter les contraintes temporelles, notamment à cause de temps de routage trop long entre 2 éléments.

L'architecture conçue atteint la fréquence de 250 MHz et a été intégrée à la solution RIFFA afin d'établir la communication via le bus PCIe (Figure 5.3). Cette fréquence est importante à atteindre car il s'agit de la fréquence maximale de fonctionnement des blocs RIFFA *RX* et *TX*. Le fonctionnement du prototype est le suivant : au niveau logiciel, les librairies RIFFA sont utilisées pour envoyer et recevoir les données via le bus PCIe. Sur le FPGA, le bloc RIFFA *RX* reçoit les données et les concatène en paquets de 125 bits. Ces paquets sont ensuite traités par l'accélérateur intégré, qui correspond à notre accélérateur auquel a été ajouté le contrôle nécessaire pour gérer la transmission des données. Les sorties sont envoyées au bloc RIFFA *TX* qui récupère les paquets de données et les transmet sur le lien PCIe. Lors de l'intégration, nous n'agissons donc que sur le bloc accélérateur intégré.

La fréquence de fonctionnement des blocs RIFFA pour la génération 1 est de 125 MHz, et de 250 MHz pour la génération 2. Dans les 2 cas, le module RIFFA transmet et reçoit des paquets de données de 128 bits, ce qui est donc inférieur aux 225 bits de données nécessaires à l'entrée de l'accélérateur. Il a donc fallu adapter l'interface à nos besoins. Pour pouvoir fournir un jeu d'entrées complet à l'accélérateur, il faut recevoir 2 paquets provenant du bloc RIFFA *RX* (Figure 5.4). La réception est gérée

Tableau 5.5 – Temps d'exécution en ms et facteur d'accélération entre exécution logicielle et accélérée matériellement via un bus PCIe génération 1 pour plusieurs configurations d'appels successifs.

Nombre d'appels	Processeur seul	Accéléré par FPGA	Facteur d'accélération
100	0,2432 ms	0,1298 ms	1,870
1000	3,0369 ms	0,1339 ms	22,680
10000	14,258 ms	0,4868 ms	29,289
100000	69,512 ms	3,136 ms	22,166
1000000	611,00 ms	19,973 ms	30,591
10000000	6025,1 ms	189,83 ms	31,739

par la machine d'état *FSM RX* et quand un paquet est reçu, cela active un compteur modulo 2. Le bloc interface recueille les 2 paquets de 128 bits pour former le mot de 225 bits à envoyer à l'accélérateur. Il est vrai que 31 bits sur les 256 envoyés en 2 paquets sont inutilisés avec cette méthode, ce qui représente 12,1% du débit utile. Cependant, l'architecture pouvant profiter de l'intégralité du débit utile demande un routage de 128 entrées vers 225 sorties, ce qui impacte lourdement la fréquence de fonctionnement de l'architecture.

Le compteur permet de remplir un registre à décalage de la même longueur que le pipeline de l'accélérateur. Ce registre à décalage contient un signal de contrôle, codé sur un seul bit. Quand le bloc interface transmet un paquet complet à l'accélérateur, le premier étage de la chaine de contrôle reçoit la retenue du compteur, qui est à l'état logique '1'. Cette chaine de contrôle permet donc de savoir à quels niveaux du pipeline se trouvent les données utiles. Cela est nécessaire car le module RIFFA n'est pas forcément constamment prêt à émettre ou à recevoir des données. Le dernier étage de la chaine de contrôle permet de faire savoir à la machine d'état de transmission, *FSM TX*, qu'un résultat est à transmettre. Si le module RIFFA est prêt à recevoir une donnée, la transmission est réalisée. Dans le cas inverse, un signal de contrôle va bloquer tout ce qui se trouve en amont (accélérateur et réception des données) jusqu'à ce que la transmission puisse être effectuée. Cela permet d'assurer qu'aucune donnée ne soit perdue.

Des tests d'accélération ont été réalisés pour cette intégration avec le standard PCIe de génération 1. L'ensemble de l'accélérateur fonctionne donc à 125 MHz, qui est la fréquence à laquelle le module RIFFA transmet les données sous ce standard. L'accélération a été testée sur plusieurs nombres d'appels successifs de la fonction. Pour chaque configuration, l'expérience a été renouvelée 10 fois afin d'obtenir une moyenne. La même chose a été effectuée pour l'exécution du code Fortran de la fonction d'intégration sur un cœur d'un processeur Intel Xeon E5-1620 v2, fonctionnant à 3,7 GHz. Les résultats présentés dans le Tableau 5.5 montrent que la solution accélérée est plus rapide pour toutes les configurations. Le facteur d'accélération est moins élevé quand il y a peu d'appels successifs de la fonction. En effet, dans le cas où 100 appels de la fonctions sont effectués, il n'y a pas assez de données pour remplir le pipeline. Les temps de transfert des données sont alors trop impactants sur le temps d'exécution.

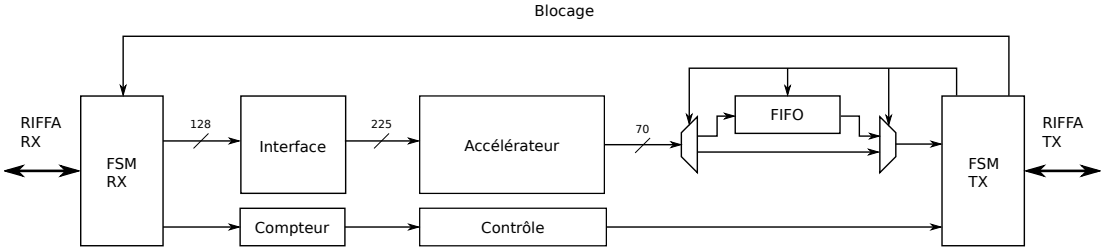


FIGURE 5.5 – Architecture du bloc Accélérateur intégré pour l'utilisation d'un lien PCIe de génération 2.

Cela explique le facteur d'accélération plus élevé quand le nombre d'appels successifs est suffisamment élevé pour pouvoir profiter pleinement de l'architecture pipelinée. L'accélération matérielle via un lien PCIe x8 de génération 1 permet d'aller 30 fois plus vite qu'un cœur de processeur.

Un facteur d'accélération de 60 doit alors être atteignable avec l'utilisation de la génération 2 du standard PCIe, puisque celle-ci propose un débit 2 fois supérieur à celui de la génération 1. Cela demande de faire fonctionner l'accélérateur intégré de la Figure 5.4 à 250 MHz. Cela n'a pas été possible à cause d'un problème de temps de propagation. En effet, le signal de blocage contrôle plus de 1000 éléments, ce qui est critique en terme de routage. Il a donc fallu faire évoluer la technique d'intégration de l'accélérateur au module RIFFA. Il a été choisi de ne plus bloquer l'avancée des données dans l'accélérateur afin de ne plus avoir le problème de propagation du signal de blocage. Il faut alors une FIFO pour stocker les sorties (Figure 5.5, les blocs RIFFA *RX* et *TX* ne sont pas représentés pour avoir une meilleure lisibilité). La gestion des sorties valides (indiquées par la chaine de contrôle) est la suivante :

— si le module RIFFA est prêt à recevoir et que la FIFO n'est pas vide, la donnée est transmise directement ;
— si le module RIFFA n'est pas prêt à recevoir, la donnée est envoyée dans la FIFO ;
— si la FIFO n'est pas vide, la donnée est envoyée dans la FIFO, peu importe l'état du module RIFFA.

Ce fonctionnement permet de ne pas perdre de résultats et de les conserver dans le bon ordre sachant que tant que la FIFO n'est pas vide, les données s'y trouvant sont tramises en priorité. Le signal de blocage est actif quand il y a des données à transmettre (dans la FIFO ou directement à la sortie de l'accélérateur) mais que le module RIFFA n'est pas prêt à recevoir. Le blocage ne concerne que la réception des données. Dans le pire des cas, il faut pouvoir stocker un nombre de résultats égal à la longueur du pipeline de l'accélérateur. La longueur de la FIFO correspond à cette dimension qui est de 255 dans notre cas.

Cette intégration est la meilleure solution atteinte en utilisant le module RIFFA et permet de proposer une plate-forme d'accélération matérielle basée sur un lien PCIe x8 de génération 2 en saturant son débit. Le Tableau 5.6 présente les ressources utilisées par l'ensemble de l'architecture sur le FPGA Virtex-7 690T. Le module RIFFA utilise un nombre négligeable de ressources par rapport de l'accélérateur seul. L'outil Vivado fournit une estimation de la consommation de la plate-forme FPGA égale à 4,369 W. De nouveaux tests d'accélération ont été effectués (Tableau 5.7). Ces résul-

Tableau 5.6 – Utilisation des ressources du FPGA Virtex-7 690T par la solution d'accélération proposée.

Ressources	Utilisées	Disponibles	Pourcentage d'utilisation
LUTs	177111	433200	40,9
Flip-Flop	119492	866400	13,8
BRAM	70	1470	4,7
DSP	1145	3600	31,2

Tableau 5.7 – Temps d'exécution en ms et facteur d'accélération entre exécution logicielle et accélérée matériellement via un bus PCIe génération 2 pour plusieurs configurations d'appels successifs.

Nombre d'appels	Processeur seul	Accéléré par FPGA	Facteur d'accélération
100	0,2432 ms	0,1338 ms	1,818
1000	3,0369 ms	0,14358 ms	21,151
10000	14,258 ms	0,2938 ms	48,530
100000	69,512 ms	2,197 ms	31,64
1000000	611,00 ms	10,479 ms	58,307
10000000	6025.1 ms	95,864 ms	62,850

tats confirment le facteur d'accélération de 60 prédit précédemment. La plate-forme proposée permet donc par exemple de réaliser en 1 minute ce que réalise un cœur de processeur en 1 heure. Cela permet également d'envisager un facteur d'accélération de 120 si la génération 3 du standard PCIe est utilisé. Toutefois, la solution RIFFA ne permet pas d'utiliser cette version du bus. Il faut donc se tourner vers d'autres solutions d'intégration.

Une fonction d'intégration similaire a été accélérée sur un GPU Nvidia Tesla K40 [72]. Le facteur d'accélération par rapport à un cœur de processeur est de 80. L'accélérateur implanté sur FPGA a donc un meilleur potentiel d'accélération, tout en consommant moins d'énergie. En effet, le Tesla K40 consomme 235 W en fonctionnement soit 54 fois la consommation de notre prototype FPGA. L'accélération matérielle sur FPGA est donc plus qu'une alternative à ce qui se fait actuellement sur GPU puisqu'il est possible de proposer une meilleure accélération pour un coût énergétique beaucoup moins élevé.

Le facteur d'accélération est actuellement limité par le lien de communication. Il est tout à fait possible de faire fonctionner l'accélérateur jusqu'à 740 MHz. En effet, des cycles de latence peuvent être ajoutés afin de subdiviser des opérations effectuées sur des données représentées sur un grand nombre de bits et ainsi augmenter la fréquence de fonctionnement. La limite dépend alors de l'élément le plus limitant : le DSP. Cela appuie notre approche de ne pas utiliser de BRAM pour la conception de l'accélérateur. Ce problème de débit de communication concerne aussi les traitements par GPU, où la bande passante du bus PCIe ne suffit plus, surtout avec l'arrivée de

FIGURE 5.6 – Maillage d'un cerveau en 1000 triangles utilisé pour les tests de précision de l'accélérateur.

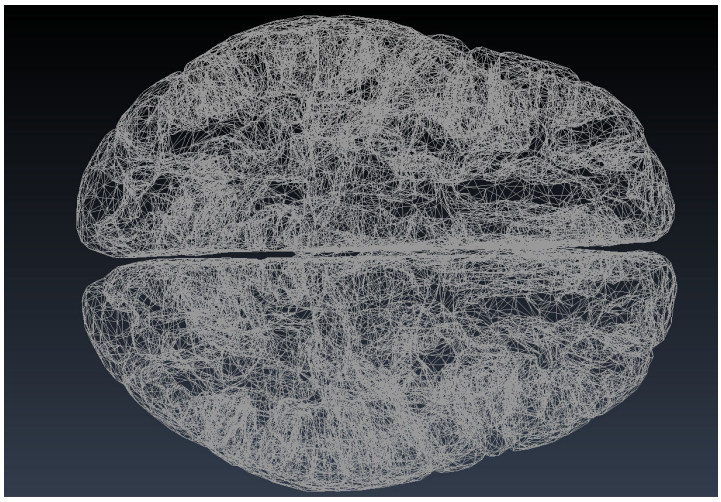


FIGURE 5.7 – Maillage d'un cerveau en 15000 triangles.

systèmes multi-GPU. Cela a notamment poussé Nvidia à développer un nouveau lien de communication inter-GPU, nommé NVLink. Nvidia annonce un gain en débit de 46% par rapport à un lien PCIe [132]. Cela montre bien la demande de liens de communication performants pour le calcul déporté et l'accélération pas FPGA profiterait totalement de ce genre d'évolution.

La précision de l'accélérateur conçu a également été analysée. Pour cela nous nous sommes basés sur un maillage de cerveau en 1000 triangles (Figure 5.6). Un maillage de cette précision nécessite environ 6,5 millions d'appels de la fonction accélérée. Il faut garder à l'esprit qu'un maillage de 1000 triangles n'offre pas une très bonne résolution. En effet, il est par exemple possible d'utiliser des maillages de 15000 triangles (Figure 5.7). Le nombre d'appels de la fonction, noté N, dépend logiquement

Tableau 5.8 – Maximum, minimum et valeur moyenne de l'erreur relative et absolue pour 1000000 appels de la fonction d'intégration.

	Relative	Absolue
Maximum	78	6,62
Minimum	$1{,}20.10^{-8}$	$1{,}16.10^{-9}$
Moyenne	$6{,}19.10^{-2}$	$4{,}80.10^{-3}$

Tableau 5.9 – Maximum, minimum et valeur moyenne de l'erreur relative et absolue pour 1000000 appels de la fonction d'intégration après étude des appels critiques.

	Relative	Absolue
Maximum	2,67	$3{,}05.10^{-1}$
Minimum	$7{,}15.10^{-9}$	$2{,}85.10^{-9}$
Moyenne	$1{,}18.10^{-2}$	$1{,}29.10^{-3}$

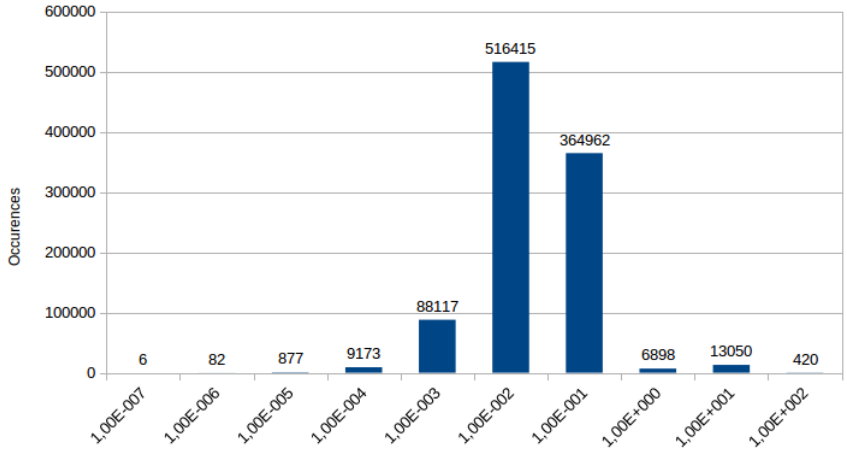
du nombre de triangles considérés, selon la formule suivante [133] :

$$N = N_i \times \frac{N_t^2}{2}, \tag{5.34}$$

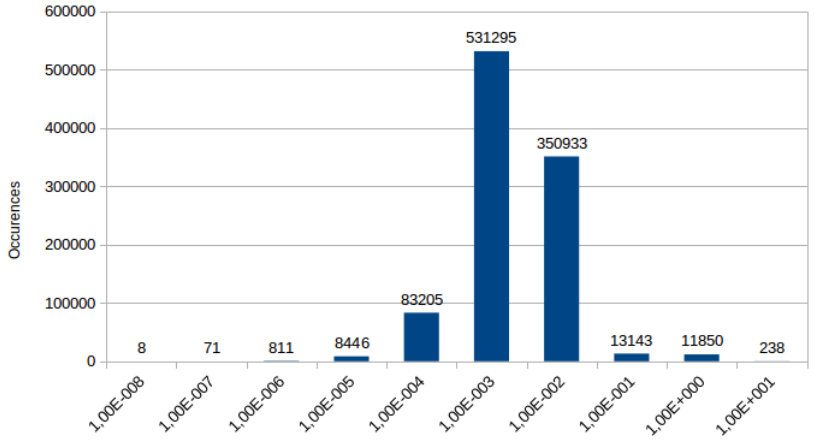
avec N_t le nombre de triangles et N_i le nombre de points d'intégration, qui est égal à 13 dans le cas étudié ici. Cette formule permet de définir que pour un maillage 15000 triangles, 1,5 milliards d'appels sont nécessaires. Cette donnée montre bien l'importance de chercher à accélérer au mieux cette fonction d'intégration puisque le nombre d'appels évolue de façon quadratique par rapport au nombre de triangles considérés.

L'erreur relative et absolue du premier million d'appels a été étudiée (Tableau 5.9). L'accélérateur a donc une précision relative moyenne de l'ordre de 10^{-2} et une précision absolue moyenne de l'ordre de 10^{-3}. Cependant, certains appels divergent totalement de ces moyennes (Figure 5.8). Ces appels ont alors été étudiés pour améliorer la précision de l'accélérateur. Leurs déroulements logiciel et matériel ont été comparés pour mettre en lumière les points de divergences. Cela a permis de révéler des situations inattendues. L'exemple le plus marquant est que, dans certains cas particuliers, l'architecture matérielle pouvait avoir une entrée négative à l'entrée d'un module de calcul de logarithme à cause des approximations. Il a alors fallu ajouter un test sur le signe de certaines valeurs pour régler ce problème.

Après plusieurs itérations de cette démarche, la précision de notre prototype a considérablement augmenté. En effet, l'erreur relative moyenne a été divisée par 5 et l'erreur absolue moyenne par 3,5. La nouvelle distribution des erreurs est présentée en Figure 5.9. Le traitement des appels critiques en terme de précision a légèrement augmenté les ressources utilisées par notre prototype (Tableau 5.10). L'augmentation du nombre de DSP utilisés vient de la nécessité d'ajouter un bit de représentation pour une donnée intermédiaire, ce qui s'est répercuté sur quelques multiplications le long de la chaine de traitement. La précision de l'accélérateur conçu est satisfaisante pour les partenaires du côté algorithmique. Il reste cependant à vérifier que celle-ci



(a) Erreur relative



(b) Erreur absolue

FIGURE 5.8 – Distribution de l'erreur relative (a) et absolue (b) pour 1000000 appels de la fonction accélérée.

Tableau 5.10 – Utilisation des ressources du FPGA par la solution d'accélération proposée après étude des points critiques.

Ressources	Utilisées	Disponibles	Pourcentage d'utilisation
LUTs	177483	433200	41,0
Flip-Flop	119604	866400	13,8
BRAM	70	1470	4,7
DSP	1161	3600	32,3

n'impacte pas l'algorithme global de conception de l'opérateur Calderon compressé présenté dans le Chapitre 1.

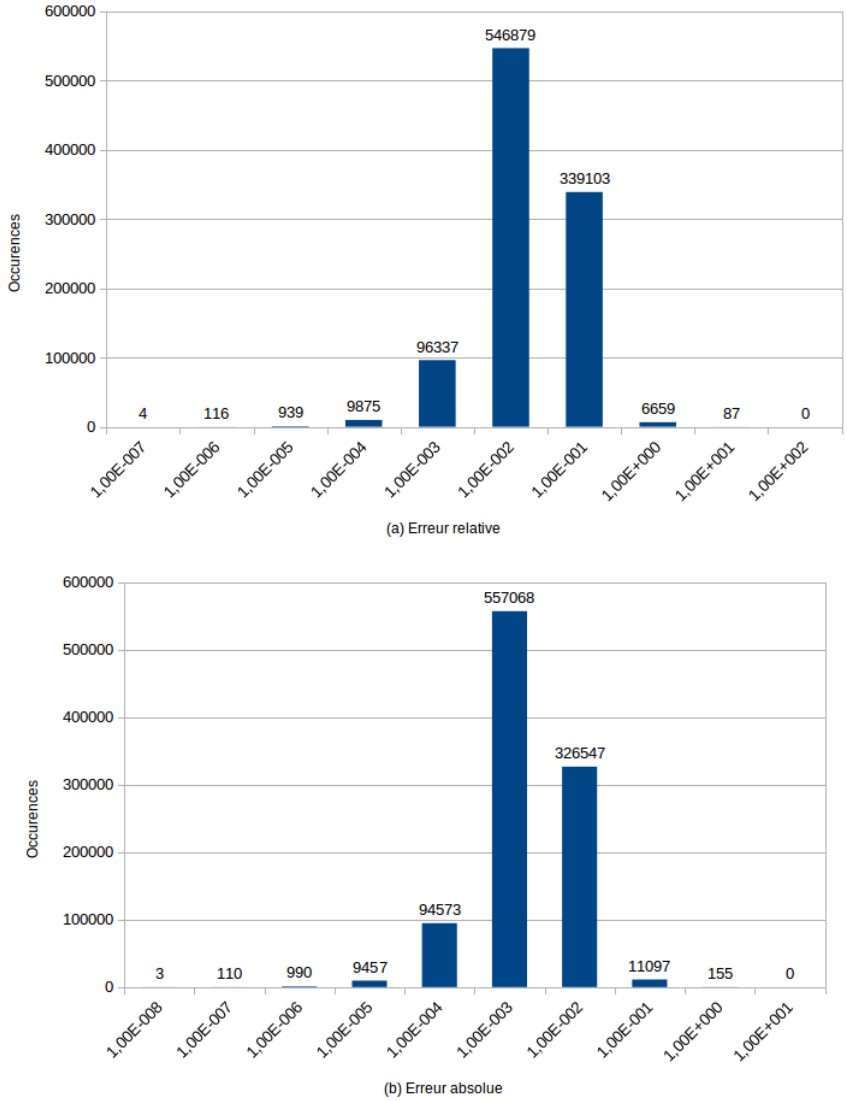


FIGURE 5.9 – Distribution de l'erreur relative (a) et absolue (b) pour 1000000 appels de la fonction accélérée après étude des points critiques.

5.3.2 ASIC

Nous avons commencé à explorer le portage ASIC du prototype réalisé sur FPGA. Le flot de conception Cadence a été utilisé, notamment les outils Genus et Innovus pour la technologie CMOS 65nm de STMicrolectronics. Le circuit étudié ne concerne que l'accélérateur. Aucun système d'interface n'y est connecté. Le chemin critique obtenu est de 764 ps, ce qui correspond à une fréquence de fonctionnement de 1,309 GHz. Ce chemin critique se situe au niveau d'une multiplication. Il est donc possible, si besoin, de réduire ce chemin critique en découpant cette multiplication en plusieurs multiplications partielles. Cette fréquence est 5 fois plus élévée que les 250 MHz du prototype FPGA et 1,8 fois plus élevée que la fréquence maximale sur un FPGA Virtex-7. Cela signifie qu'avec la perspective d'un lien de communication pouvant amener les entrées à cette fréquence, le facteur d'accélération peut être multiplié par 5 également.

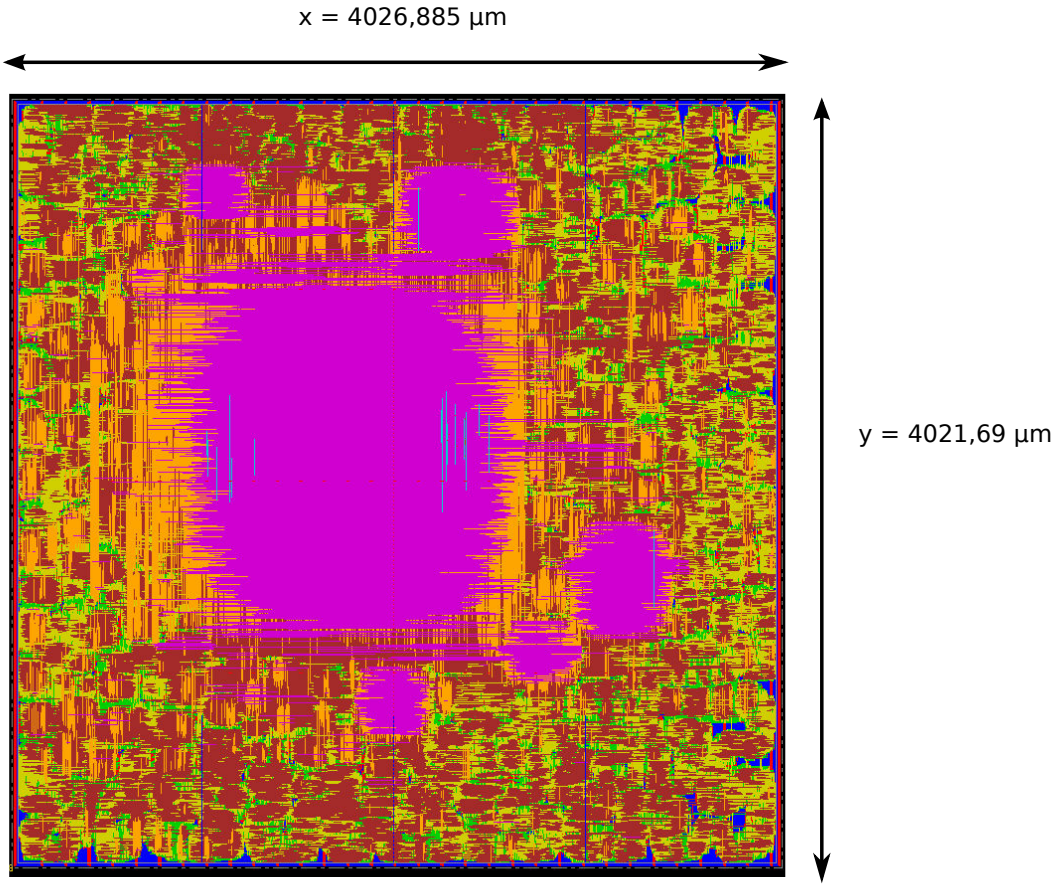


FIGURE 5.10 – Masque ASIC généré pour l'accélérateur.

Tableau 5.11 – Caractéristiques du circuit ASIC.

Caractéristique	Valeur
Surface	16,195 mm^2
Fréquence maximale	1,3 GHz
Capacité de calcul	80,83 Mop/s/mm^2
Alimentation	1,10 V
Température de fonctionnement	25 °C

Nous avons donc une perspective d'accélération par un facteur de 600 par rapport à un cœur de processeur. Pour illustrer l'impact d'une telle accélération, cela permet de réduire 24 heures de calcul à 2 minutes et 2 secondes.

L'opération de placement et routage a également été réalisée (Figure 5.10). Les dimensions du circuit obtenu sont de 4026,885 par 4021,690 pm. Ce qui donne une surface de 16,195 mm^2. L'accélérateur proposé a donc une performance de calcul de 80,83 Mop/s/mm^2, l'opération considérée ici étant l'intégration en 3 dimensions. Le Tableau 5.11 rassemble quelques caractéristiques de l'ASIC. Ceci reste encore des résultats préliminaires, la conception n'étant pas encore finalisée, mais cela valide la dé-

marche suivie pour la conception de cet accélérateur. Les prochaines étapes concernant la conception du circuit ASIC consistent à effectuer la simulation du circuit, récupérer certaines métriques comme la consommation énergétique, et enfin fondre une puce pour tester la fonctionnalité. La surface annoncée précédemment étant trop élevée par rapport au budget prévu pour la puce, il a été décidé que seul un sous-bloc dédié au calcul d'une valeur β_i sera concerné. Ce bloc fait appel aux fonctions d'arctangente, de logarithme et d'inverse, ce qui le rend intéressant pour un test fonctionnel.

5.4 Conclusion

Dans ce Chapitre, l'implantation de l'accélérateur conçu pour la fonction d'intégration en 3 dimensions a été présentée. L'algorithme CORDIC a été utilisé pour implanter les fonctions non-linéaires qui n'avaient pas encore été traitées, soit l'arctangente et le logarithme népérien. Cela respecte tout à fait la philosophie d'implantation du Chapite 4 de ne pas utiliser de BRAM pour maximiser la fréquence. La gestion du format de la représentation des signaux a été réalisée conjointement avec l'équipe ayant conçu l'algorithme, ce qui a permis d'optimiser les ressources nécessaire à l'implantation de l'accélérateur.

Des raffinements de l'architecture ont été effectués après l'analyse de la précision de l'accélérateur. Le fait de traiter les appels les moins précis a permis en 5 modifications mineures de diviser par 5 l'erreur relative. La valeur finale de l'erreur relative moyenne est de $1{,}18.10^{-2}$.

L'architecture conçue a été testée sur la carte NetFPGA SUME, connectée par un lien PCIe à la station de travail. La gestion de la transmission des données est réalisée par la solution RIFFA, qui permet d'utiliser la génération 2 du protocole. L'intégration ainsi réalisée a permis de mesurer un facteur d'accélération de 60 par rapport à un cœur de processeur. Ce résultat est appelé à être doublé dans un futur proche avec l'utilisation de la génération 3 du standard PCIe. Cela a permis de montrer que l'utilisation d'une accélération matérielle sur plate-forme FPGA est une très bonne alternative à ce qui se fait actuellement sur GPU puisqu'elle a un potentiel d'accélération plus élevée tout en consommant 54 fois moins d'énergie.

Les travaux concernant le portage de l'accélérateur sur ASIC ont également commencé. Le flot Cadence sur la technologie CMOS 65nm de STMicroelectronics a été utilisé. Les premiers résultats montrent une fréquence atteignable de 1,3 GHz et une surface de 16 mm^2, soit une capacité de calcul de 80,83 Mop/s/mm^2, l'opération considérée ici étant l'intégrale complète. Cela ouvre la possibilité d'obtenir un facteur d'accélération de 600, sous réserve d'avoir des liens de communication ayant le débit adapté.

L'accélération matérielle par circuit dédié a donc tous les atouts pour répondre aux soucis de temps de calcul de la matrice opérateur de la formulation symétrique. Cela ouvre donc la possibilité d'explorer plus facilement l'utilisation du problème inverse de l'EEG pour des applications d'ICM, dans un premier temps avec des FPGA peu coûteux. Si le marché se développe, une implantation ASIC offre des performances bien supérieures à celles de l'état de l'art sur GPU.

Conclusion

Les recherches dans le domaine de l'ICM visent à établir un moyen fiable de communication entre un cerveau humain et une machine électronique. Les domaines d'application possibles sont multiples comme la médecine, la défense, les transports ou encore le divertissement. Il est alors important de bien choisir la méthode de mesure et d'analyse des signaux cérébraux afin de préserver le potentiel applicatif du domaine. C'est pour cette raison que l'EEG est la méthode de mesure la plus étudiée actuellement. En plus d'être moins dangereuse pour l'utilisateur que les méthodes de mesure invasives, elle est également simple à mettre en œuvre, est facilement portable et propose une très bonne résolution temporelle.

Le Chapitre 1 dresse un bilan des méthodes utilisées pour réaliser une ICM à partir de signaux EEG. Plusieurs paradigmes peuvent être utilisés. Ils reposent essentiellement sur des évolutions spécifiques des potentiels mesurés sur le cuir chevelu de l'utilisateur quand ce dernier est sujet à des stimuli, internes ou externes. Le fonctionnement de ces potentiels évoqués limite directement les ICM qui les utilisent. Malheureusement, chacune de ces méthodes souffre de la faible résolution spatiale de la mesure EEG.

Ce problème de résolution spatiale peut être résolu en s'intéressant au problème inverse de l'EEG. Cela consiste à déterminer, à partir des signaux mesurés sur le cuir chevelu, les zones actives du cerveau. Un modèle volumique de la distribution des sources peut donc être déterminé à partir de mesures surfaciques. Afin de résoudre le problème inverse de l'EEG, il faut tout d'abord résoudre tous les problèmes directs du système considéré. Il est par conséquent important d'accélérer la résolution de ce problème direct.

L'aspect algorithmique du problème a été étudié lors d'une thèse antérieure aux travaux présentés dans ce document [7]. Cette analyse mathématique a permis de réduire la complexité algorithmique de la conception de l'élément clé à la résolution du problème direct. Cette amélioration est toutefois insuffisante et les temps de calcul nécessaires empêchent toujours des tests cliniques à grande échelle d'ICM basées sur la résolution du problème inverse. C'est dans ce contexte que l'utilisation de l'accélération matérielle a été envisagée. Ce document présente les travaux effectués pour la mise en œuvre de cette solution.

Contributions

La fonction à accélérer est présentée et étudiée dans le Chapitre 2. Il s'agit d'une fonction d'intégration en 3 dimensions qui a été repérée comme la plus critique en

terme de temps de calcul suite à un profilage logiciel. Cette fonction prend en entrée les coordonnées d'un triangle et d'un point d'observation et fournit un scalaire en sortie, une intégrale calculée après une première phase de calculs de distances et de projections. La suite de calculs à réaliser est partiellement parallélisable et comprend des fonctions non-linéaires comme la division, la racine carré, l'arctangente ou le logarithme népérien. Chaque appel de fonction est indépendant. Il y a donc un potentiel d'accélération important.

Le choix de la plate-forme matérielle est essentiel pour assurer le traitement le plus rapide. Les 2 solutions les plus simples à utiliser sont le processeur généraliste et le GPU. Si la première de ces solutions n'offre pas les performances de parallélisation nécessaires, une accélération matérielle sur GPU a déjà été utilisée sur une fonction d'intégration similaire. Nous avons choisi d'explorer une autre solution en concevant une architecture dédiée. Cette solution est certes plus complexe à mettre en œuvre mais permet d'avoir une architecture optimisée pour l'application visée. Le but est d'obtenir une solution offrant le meilleur ratio entre vitesse de calcul et consommation énergétique, tout en occupant le moins de silicium possible. Tous ces avantages rendent cette solution pertinente pour une production à grande échelle et ainsi équiper chaque hôpital ou laboratoire travaillant sur de l'imagerie cérébrale. L'objectif à court terme est de proposer un prototype sur plate-forme FPGA pour ensuite étudier la possibilité d'une implantation ASIC dans l'idée de fournir une solution embarquée.

Le Chapitre 3 détaille la mise en œuvre d'une accélération matérielle par circuit dédié. Il faut dans un premier temps réaliser l'architecture puis intégrer l'opérateur conçu dans un système complet. La conception d'un circuit dédié demande des connaissances spécifiques. Les outils de synthèse de haut niveau permettent de démocratiser la conception d'architecture et offrent un excellent compromis entre performances et temps de conception. Toutefois, pour des architectures complexes, ces outils offrent de moins bonnes performances (fréquence de fonctionnement, ressources utilisées...) qu'une description matérielle réalisée en description RTL.

L'utilisation d'IPs permet également d'accélérer la conception d'une description matérielle. Ces éléments sont fournis soit par des entreprises (spécialisées dans cette activité, ou des fondeurs de circuit et des fabricants de FPGA), soit par des projets communautaires. Les IPs provenant des constructeurs ne sont cependant compatibles qu'avec les flots de conception du constructeur correspondant. Cela empêche notamment de porter facilement le prototype FPGA sur circuit ASIC. L'utilisation d'IPs libres de droit est intéressante si ces blocs répondent à nos contraintes et perspectives. Ces raisons nous ont poussé à privilégier l'écriture RTL de la description matérielle, tout en considérant les IPs proposées par la communauté.

L'intégration de l'accélérateur dépend tout d'abord du système considéré. Notre objectif à court terme est de proposer un prototype sur plate-forme FPGA, il faut alors un moyen de relier la carte à une station de calcul. Le standard PCIe permet de mettre en œuvre une communication série à haut débit entre le processeur d'une machine et un périphérique. Cette technologie est déjà utilisée pour les GPU. Son utilisation pour établir une communication entre un processeur et un FPGA est toutefois moins démocratisée. Cela explique l'existence de plusieurs projets de recherche proposant des solutions matérielles et logicielles pour mettre en place cette communication. La solu-

tion RIFFA est l'une d'entre elles et est intégrée au sein de la plate-forme NetFPGA SUME, qui dispose d'un FPGA intéressant en terme de ressources de calcul. Cette plate-forme a donc été choisie pour la mise en œuvre du prototype d'accélérateur.

Il faut aussi prendre en compte la problématique d'intégration au sein d'un système embarqué, qui est la forme que doit prendre le projet à plus long terme. Si un processeur du SoC ne peut pas être modifié, l'accélérateur doit alors être connecté au bus du système. Pour faciliter cette connexion, nous avons proposé le coprocesseur Ouessant, qui peut être intégré à n'importe quel SoC et ainsi servir d'intermédiaire entre le processeur et l'accélérateur.

La conception et l'implantation d'opérateurs non-linéaires sont décrites dans le Chapitre 4. Les fonctions concernées sont l'inverse et l'inverse de la racine carrée. L'algorithme itératif de Newton-Raphson a été choisi pour l'implantation de ces opérateurs. Il s'agit d'un algorithme où la précision est doublée à chaque itération. Cette option est en effet celle qui répond au mieux à nos contraintes (pipeline, nombre d'instanciations, représentation des données...).

Dans l'état de l'art, cet algorithme utilise un bloc mémoire pour générer la première approximation dont la précision définit le nombre d'itérations nécessaires. Cependant, les blocs mémoire limitent la fréquence atteignable des architectures implantées sur les FPGA Xilinx et sur la technologie ASIC à laquelle nous avons accès. Nous avons alors proposé de nouvelles implantations de l'algorithme de Newton-Raphson pour les opérateurs de calcul d'inverse et de l'inverse de la racine carrée. Ces implantations pipelinées n'utilisent pas de blocs mémoire et leur fréquence n'est limitée que par les blocs chargés d'effectuer les multiplications. De plus, leurs descriptions sont génériques, ce qui fait qu'elles s'intègrent simplement dans toute architecture. Ces opérateurs ont été mis à la disposition de la communauté scientifique. Il s'agit en effet d'opérations usuelles, comme en électromagnétique pour la loi de Coulomb.

Une étude préliminaire pour une intégration ASIC a été réalisée pour les opérateurs conçus et validés sur plate-forme FPGA. La technologie concernée est le CMOS 65nm de STMicroelectronics. Ces opérateurs atteignent la fréquence maximale de fonctionnement de 1,6 GHz. La surface d'occupation permet d'obtenir une capacité de calcul de 40 Gop/s/mm^2 dans le cas d'un opérateur sur 16 bits, ce qui multiplie par 4 les performances de l'état de l'art sur cette technologie.

Finalement, l'accélérateur conçu est présenté et testé dans le Chapitre 5. Le calcul de l'arctangente et du logarithme népérien est réalisé grâce à une implantation de l'algorithme CORDIC. La gestion de la représentation des données a fait l'objet d'une discussion avec l'équipe ayant conçu l'algorithme.

Le test est réalisé sur la plate-forme NetFPGA SUME, connectée à la station de travail grâce à un lien PCIe. La communication entre le processeur et le FPGA est mise en œuvre par la solution RIFFA, qui permet d'utiliser la génération 2 du protocole. Cette intégration a permis de mesurer un facteur d'accélération de 60 par rapport à un cœur de processeur. Il y a donc la possibilité d'atteindre un facteur de 120 en utilisant la génération 3 du standard PCIe. Cela a permis de démontrer que l'accélération matérielle sur FPGA est plus intéressante que l'accélération sur GPU tout en consommant plus de 50 fois moins d'énergie.

Des premiers résultats concernant l'implantation de l'accélérateur sur technologie ASIC CMOS 65nm ont été obtenus. Le flot Cadence a été utilisé. La fréquence maximale atteignable est de 1,3 GHz, pour une surface de surface de 16 mm^2, soit une capacité de calcul de 80,83 Mop/s/mm^2, l'opération étant ici l'intégration complète. Un facteur d'accélération de 600 est donc envisageable si un lien de communication propose le débit nécessaire. Ce facteur d'accélération permettrait de réduire 10 heures de calcul sur un cœur de processeur en 1 minute sur le circuit dédié.

Perspectives

L'accélération matérielle par FPGA prend de plus en plus d'importance au fil des années. Le rachat d'Altera par Intel est la parution de son *Acceleration Stack* est une preuve certaine du potentiel de ce domaine. Cela explique les efforts des constructeurs mais aussi des projets communautaires pour proposer des solutions d'intégration via un bus PCIe. L'annonce pour 2019 de Xillybus d'une solution dédiée pour le calcul de haute performance est une autre preuve de la demande de solutions de la part de la communauté. Il y a donc de grandes chances que l'utilisation de circuit dédié pour de l'accélération matérielle via un lien PCIe soit aussi simple d'accès que l'utilisation de GPU d'ici quelques années. Nous pouvons aussi envisager l'essort de la communication entre plates-formes FPGA pour mettre en place des chaines d'accélérateurs, comme le propose Nvidia avec son lien NVlink pour GPU.

Notre perspective à court terme concerne essentiellement le prototype implanté sur FPGA. En effet, des travaux d'intégration ont commencé sur des cartes BittWare, qui fournit un projet contenant un lien PCIe de génération 3. Les cartes concernées sont la XUPP3R et la XUPVV4 qui contiennent respectivement un FPGA VU9P et VU13P. Ces cartes disposent d'une interface PCIe x16 de génération 3. Nous pouvons donc, par rapport à notre prototype, passer de la génération 2 à la génération 3 tout en utilisant le double de lignes. Cela ouvre la possibilité de multiplier par 4 le facteur d'accélération, pour obtenir finalement un facteur de 240. Un autre tâche, qui peut être réalisée en parallèle, est d'intégrer le prototype dans l'algorithme global qui gère la conception de la matrice permettant la résolution du problème inverse de l'EEG. Cette intégration vise à d'évaluer l'impact de l'accélération matérielle sur les performances du système complet. Les points à évaluer sont bien évidemment le temps de calcul mais aussi la charge mémoire de la station de calcul ou les possibles modifications nécessaires comme l'ajout d'itérations pour les algorithmes de résolution pour palier la perte de précision si celle-ci s'avère critique. Selon les résultats, des modifications pourront être apportées au prototype, notamment pour améliorer la précision des résultats. Tous les résultats concernant le prototype feront l'objet d'une publication qui peut intéresser à la fois la communauté des architectures électroniques mais aussi celle s'intéressant à l'imagerie cérébrale. Par exemple, ces résultats peuvent permettre d'accélérer la détection des foyers épiloptogènes.

Autre perspective à court terme, l'axe d'implantation ASIC reste à finaliser. Comme indiqué dans le Chapitre 5, l'accélérateur complet ne peut pas être fondu sur puce pour raison budgétaire. La piste retenue est donc l'implantation d'un bloc de calcul d'une valeur β_i qui fait appel aux fonctions d'arctangente, de logarithme et d'inverse. Il est également possible d'ajouter en parallèle un opérateur de calcul de l'inverse de la ra-

cine carrée afin de tester chacun des opérateurs non-linéaires utilisés au sein de l'accélérateur. La puce générée sera alors testée avec un FPGA contrôlant les entrées et sorties de la puce ASIC. L'idée d'un tel test est de démontrer par la mesure la validité des hypothèses et confirmer les métriques de performances que sont le débit, la surface occupée et la consommation.

Concernant l'évolution de ces travaux sur la durée, une piste possible est d'implanter d'autres fonctions utilisées par l'équipe ayant fourni la fonction d'intégration en 3 dimensions. En effet, la fonction accélérée par notre prototype ne concerne qu'une partie des intégrations réalisées. Cependant, les autres intégrales utilisent les mêmes calculs de distances et de projections. Le prototype conçu peut alors servir de base pour un accélérateur encore plus complexe.

L'accélération matérielle ouvre donc de nouvelles perspectives pour la mise en place d'applications d'ICM. En effet, la réduction du temps de calcul de la matrice opérateur de la formulation symétrique permet de faciliter l'exploration de l'utilisation du problème inverse de l'EEG en fluidifiant les phases de tests cliniques. Cela permet également de simplifier l'utilisation de maillage de cerveau plus complexes afin d'améliorer la résolution du système et le nombre de commandes possible pour une application d'ICM. Par exemple, dans le cas du paradigme de l'imagerie motrice, l'augmentation de la résolution spatiale permettrait de passer de l'imagination du mouvement d'une main à celui d'un doigt. Cela pourrait même ouvrir la possibilité d'utiliser de tous nouveaux paradigmes d'ICM utilisant des signaux EEG.

Publications associées

Publication en journal

— E. Libessart, M. Arzel, C. Lahuec, and F. Andriulli. A Scaling-Less Newton-Raphson Pipelined Implementation for a Fixed-Point Reciprocal Operator. *IEEE Signal Processing Letters*, 24(6) :789– 793, June 2017.

Publications en conférence

— P. Horrein, P. Gleonec, E. Libessart, A. Lalevée, and M. Arzel. Ouessant : Flexible integration of dedicated coprocessors in Systems on Chip. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1493–1496, March 2016,

— E. Libessart, M. Arzel, C. Lahuec, and F. Andriulli. A scalingless Newton-Raphson pipelined implementation for a fixed-point inverse square root operator. In *2017 15th IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 157–160, June 2017,

— E. Libessart, M. Arzel, C. Lahuec, and F. Andriulli. Implantation en virgule fixe d'un opérateur de calcul d'inverse à base de Newton-Raphson, sans normalisation et sans bloc mémoire. In *GRETSI 2017 : 26ème colloque du Groupement de Recherche en Traitement du Signal et des Images*, Juan-Les- Pins, France, September 2017,

— E. Libessart, M. Arzel, C. Lahuec, and F. Andriulli. 40 Gop/s/mm^2 fixed-point operators for Brain Computer Interface in 65 nm CMOS. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2018.

Glossaire

3EGBCI *Embedded EEG-based BCI System*. 11

AGP *Accelerated Graphics Port*. 49, 50

ASIC *Application-Specific Integrated Circuit*. X, XIII, XVI, 3, 34, 41–49, 53, 60, 63, 81–85, 91, 99–101, 104–107

BRAM *Block RAM*. XV, 58, 65, 72, 73, 76, 86, 91, 95, 101

CCI *Core Cache Interface*. 59

CORDIC *COordinate Rotation Digital Computer*. X, XII, 67–69, 71, 87–89, 101, 105

CSP *Common Spatial Pattern*. 11

DFT *Discrete Fourier Transform*. 57–59

DLE Désynchronisation Liée à l'Événement. 10, 11

DMA *Direct Memory Access*. 57

DSLE Désynchronisations et Synchronisations Liées à l'Événement. 2, 10, 11

DSP *Digital Signal Processing*. XV, 51, 72, 77, 79, 91, 95, 97

EEG Électro-Encéphalographie. IX, XI, 1–17, 19–23, 25, 27, 43, 45, 61, 85, 101, 103, 106, 107

eop *End of Operation*. 56

Ex *Execute*. 35

exec *Execution*. 56

FF *Flip-Flop*. 58

FIFO *First in First Out*. 52, 54, 57, 58, 94

FloPoCo *Floating-Point Cores*. 48

FPGA *Field-Programmable Gate Array*. X, XV, XVI, 3, 34, 42–49, 51–53, 57–61, 63–65, 72, 73, 76, 77, 79–81, 83–85, 91–96, 98, 99, 101, 104–107

FPU *Floating-Point Unit*. 38

GPGPU *General-Purpose computings on Graphics Processing Unit*. 40

GPU *Graphics Processing Unit*. 39–44, 50, 95, 96, 101, 104–106

HDL *Hardware Description Language*. 42, 45, 46, 48, 49
HLS *High Level Synthesis*. XII, 46, 47

ICM Interface Cerveau-Machine. IX, XI, 1–13, 15, 23–25, 41, 49, 61, 85, 101, 103, 107
ID *Instruction Decode*. 35
IDCT *Inverse Discrete Cosine Transform*. 57, 58
IDFT *Inverse Discrete Fourier Transform*. 59
IF *Instruction Fetch*. 35
IM Imagerie Motrice. 11
IP *Intellectual Property*. X, 46–49, 51, 52, 60, 104
IRM Imagerie par Résonance Magnétique. 2, 5, 6, 15, 22
IRMf Imagerie par Résonance Magnétique fonctionnelle. 2, 5

LFM *Lead-Field Matrix*. 22–25, 27
LOD *Leading One Detector*. 73, 74
LUT *Look-Up Table*. 42, 58, 64, 66, 71, 72, 79, 91

MEG Magnéto-Encéphalographie. 2, 5, 6
Mem *Memory*. 35
mvfc *Move From Coprocessor*. 56
mvtc *Move To Coprocessor*. 56

PC Partie Contrôle. 34, 35
PCI *Peripheral Component Interconnect*. 49, 50
PCIe *Peripheral Component Interconnect Express*. X, XIII, XV, 49–53, 61, 72, 76, 91–96, 101, 104–106
PEVRP Potentiels Évoqués Visuels de Régime Permanent. XI, 2, 9, 10
PLE Potentiels Liés à l'Événement. 8, 10
PMV Produit Matrice-Vecteur. XV, 20, 21
PO Partie Opérative. 34, 35

RIFFA *Reusable Integration Framework for FPGA Accelerators*. XII, 51–53, 61, 92–95, 101, 105
RISC *Reduced Instruction Set Computer*. 35
RTL *Register-Transfer Level*. 42, 43, 45–49, 60, 104

SABRE *Seizing Advances in BCI from high Resolution EEG imaging in runtime*. 3
SLE Synchronisation Liée à l'Événement. 10, 11
SoC *System on Chip*. X, XII, 53, 54, 58, 59, 61, 105

UAL Unité Arithmétique et Logique. 34, 35, 38
UPI *Ultra Path Interconnect*. 59
UVF Unité de calcul en Virgule Flottante. 38

WB *Write Back*. 35

Bibliographie

[1] Entraide ESI IDE, "L'Imagerie par Résonnance Magnétique (IRM) – ENTRAIDE ESI IDE," 2017. [Online]. Available : http://entraide-esi-ide.com/limagerie-par-resonnance-magnetique-irm/

[2] U. of Washington, "MEG Brain Imaging at I-LABS | Institute for Learning and Brain Sciences (I-LABS)," 2012. [Online]. Available : http://ilabs.washington.edu/meg-brain-imaging-i-labs

[3] U. Magazine, "UP Magazine - Les « sciences computationnelles », nouvelle frontière pour la connaissance ?" 2017. [Online]. Available : http://up-magazine.info/index.php/technologies-a-la-pointe/technologies/7131-les-sciences-computationnelles-nouvelle-frontiere-pour-la-connaissance

[4] A. Turnip, K.-S. Hong, and M.-Y. Jeong, "Real-time feature extraction of P300 component using adaptive nonlinear principal component analysis," *Biomed Eng Online*, vol. 10, p. 83, Sep. 2011. [Online]. Available : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3749271/

[5] W. Wechselberger, "Image gratuite sur Pixabay - Eeg, Intégration, Hirnstrommessung," 2017. [Online]. Available : /fr/eeg-int%C3%A9gration-hirnstrommessung-2680957/

[6] holdentrils, "Image gratuite sur Pixabay - Cerveau, Anatomie, Humaine," 2014. [Online]. Available : /fr/cerveau-anatomie-humaine-la-science-512758/

[7] J. E. O. Guzman, "Fast boundary element formulations for electromagnetic modelling in biological tissues," phdthesis, Ecole nationale supérieure Mines-Télécom Atlantique, Nov. 2017. [Online]. Available : https://tel.archives-ouvertes.fr/tel-01801755/document

[8] R. D. Graglia, "On the numerical integration of the linear shape functions times the 3-D Green's function or its gradient on a plane triangle," *IEEE Transactions on Antennas and Propagation*, vol. 41, no. 10, pp. 1448–1455, Oct. 1993.

[9] M. B. Giles and I. Reguly, "Trends in high-performance computing for engineering calculations," *Phil. Trans. R. Soc. A*, vol. 372, no. 2022, p. 20130319, Aug. 2014. [Online]. Available : http://rsta.royalsocietypublishing.org/content/372/2022/20130319

[10] E. Visan, "Free Image on Pixabay - Electronics, Computer, Technology," 2015. [Online]. Available : /en/electronics-computer-technology-1070489/

[11] D.-U. Lee, W. Luk, J. Villasenor, and P. Y. K. Cheung, "Non-uniform Segmentation for Hardware Function Evaluation," in *Field Programmable Logic and*

Application, ser. Lecture Notes in Computer Science, P. Y. K. Cheung and G. A. Constantinides, Eds. Springer Berlin Heidelberg, 2003, pp. 796–807.

[12] T. Ebrahimi, J. M. Vesin, and G. Garcia, "Brain-computer interface in multimedia communication," *IEEE Signal Processing Magazine*, vol. 20, no. 1, pp. 14–24, Jan. 2003.

[13] Y. Daabaj, "An evaluation of the usability of human-computer interaction methods in support of the development of interactive systems," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, Jan. 2002, pp. 1868–1877.

[14] H. Hongo, M. Ohya, M. Yasumoto, and K. Yamamoto, "Face and hand gesture recognition for human-computer interaction," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 2, 2000, pp. 921–924 vol.2.

[15] K. H. Englmeier, C. Krapichler, M. Haubner, M. Seemann, and M. Reiser, "Virtual reality and multimedia human-computer interaction in medicine," in *1998 IEEE Second Workshop on Multimedia Signal Processing (Cat. No.98EX175)*, Dec. 1998, pp. 193–202.

[16] K.-K. Shyu, Y.-J. Chiu, P.-L. Lee, M.-H. Lee, J.-J. Sie, C.-H. Wu, Y.-T. Wu, and P.-C. Tung, "Total Design of an FPGA-Based Brain #x2013 ;Computer Interface Control Hospital Bed Nursing System," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 7, pp. 2731–2739, Jul. 2013.

[17] S. N. Abdulkader, A. Atia, and M.-S. M. Mostafa, "Brain computer interfacing : Applications and challenges," *Egyptian Informatics Journal*, vol. 16, no. 2, pp. 213–230, Jul. 2015. [Online]. Available : http://www.sciencedirect.com/science/article/pii/S1110866515000237

[18] B. He, Ed., *Neural engineering*, ser. Bioelectric engineering. New York : Kluwer Acad. / Plenum Publ, 2005, no. 3, oCLC : 255272017.

[19] J. J. Vidal, "Toward direct brain-computer communication," *Annu. Rev. Biophys. Bioeng.*, vol. 2, pp. 157–180, 1973.

[20] ——, "Real-time detection of brain events in EEG," *Proceedings of the IEEE*, vol. 65, no. 5, pp. 633–641, May 1977.

[21] W. H. Dobelle, "Artificial Vision for the Blind by Connecting a Television Camera to the Visual Cortex," *ASAIO Journal*, vol. 46, no. 1, p. 3, Feb. 2000. [Online]. Available : https://journals.lww.com/asaiojournal/Fulltext/2000/01000/Artificial_Vision_for_the_Blind_by_Connecting_a.2.aspx

[22] J. C. Sanchez, J. M. Carmena, M. A. Lebedev, M. A. L. Nicolelis, J. G. Harris, and J. C. Principe, "Ascertaining the importance of neurons to develop better brain-machine interfaces," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 943–953, Jun. 2004.

[23] M. Sarracanie, C. D. LaPierre, N. Salameh, D. E. J. Waddington, T. Witzel, and M. S. Rosen, "Low-Cost High-Performance MRI," *Scientific Reports*, vol. 5, p. 15177, Oct. 2015. [Online]. Available : https://www.nature.com/articles/srep15177

[24] K. O. Dimon, *Foundations Of Potential Theory*. Berlin Verlag Von Julius Springer., 1929. [Online]. Available : http://archive.org/details/foundationsofpot033485mbp

[25] Inria, "Discover OpenViBE," May 2011. [Online]. Available : http://openvibe.inria.fr/discover/

[26] S. M. Solutions, "Mobile MRI Scanner," 2006. [Online]. Available : https://usa.healthcare.siemens.com/medical-imaging/magnetic-resonance-imaging/0-35-to-1-5t-mri-scanner/mobile-mri-scanner

[27] R. W. Homan, J. Herman, and P. Purdy, "Cerebral location of international 10–20 system electrode placement," *Electroencephalography and Clinical Neurophysiology*, vol. 66, no. 4, pp. 376–382, Apr. 1987. [Online]. Available : http://www.sciencedirect.com/science/article/pii/0013469487902069

[28] B. Farnsworth and Ph.D., "EEG Headset Prices – An Overview of 15+ EEG Devices," Jul. 2017. [Online]. Available : https://imotions.com/blog/eeg-headset-prices/

[29] D. L. Schomer and F. L. d. Silva, *Niedermeyer's Electroencephalography : Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins, Oct. 2012, google-Books-ID : NPeefSGSbfEC.

[30] H. Gollee, I. Volosyak, A. J. McLachlan, K. J. Hunt, and A. Gräser, "An SSVEP-Based Brain #x2013 ;Computer Interface for the Control of Functional Electrical Stimulation," *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 8, pp. 1847–1855, Aug. 2010.

[31] K.-K. Shyu, P.-L. Lee, M.-H. Lee, M.-H. Lin, R.-J. Lai, and Y.-J. Chiu, "Development of a Low-Cost FPGA-Based SSVEP BCI Multimedia Control System," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 4, no. 2, pp. 125–132, Apr. 2010.

[32] C.-W. Feng, T.-K. Hu, J.-C. Chang, and W.-C. Fang, "A reliable brain computer interface implemented on an FPGA for a mobile dialing system," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, Jun. 2014, pp. 654–657.

[33] D. Zhu, J. Bieger, G. Garcia Molina, and R. M. Aarts, "A Survey of Stimulation Methods Used in SSVEP-Based BCIs," *Comput Intell Neurosci*, vol. 2010, 2010. [Online]. Available : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2833411/

[34] P. Derambure, L. Defebvre, J. L. Bourriez, F. Cassim, and J. D. Guieu, "Désynchronisation et synchronisation liées à l'événement Étude de la réactivité des rythmes électrocorticaux en relation avec la planification et l'exécution du mouvement volontaire," *Neurophysiologie Clinique/Clinical Neurophysiology*, vol. 29, no. 1, pp. 53–70, Feb. 1999. [Online]. Available : http://www.sciencedirect.com/science/article/pii/S0987705399800410

[35] G. Pfurtscheller and C. Neuper, "Motor imagery and direct brain-computer communication," *Proceedings of the IEEE*, vol. 89, no. 7, pp. 1123–1134, Jul. 2001.

[36] K. Belwafi, F. Ghaffari, R. Djemal, and O. Romain, "A Hardware/Software Prototype of EEG-based BCI System for Home Device Control," *J Sign Process Syst*, vol. 89, no. 2, pp. 263–279, Nov. 2017. [Online]. Available : https://link.springer.com/article/10.1007/s11265-016-1192-8

[37] F. Lotte and C. Guan, "Regularizing Common Spatial Patterns to Improve BCI Designs : Unified Theory and New Algorithms," *IEEE Transactions on Biomedical Engineering*, vol. 58, no. 2, pp. 355–362, Feb. 2011.

[38] R. M. Chapman and H. R. Bragdon, "Evoked Responses to Numerical and Non-Numerical Visual Stimuli while Problem Solving," *Nature*, vol. 203, no. 4950, pp. 1155–1157, Sep. 1964. [Online]. Available : https://www.nature.com/articles/2031155a0

[39] Z. Lin, C. Zhang, Y. Zeng, L. Tong, and B. Yan, "A novel P300 BCI speller based on the Triple RSVP paradigm," *Scientific Reports*, vol. 8, no. 1, p. 3350, Feb. 2018. [Online]. Available : https://www.nature.com/articles/s41598-018-21717-y

[40] K. Khurana, P. Gupta, R. C. Panicker, and A. Kumar, "Development of an FPGA-based real-time P300 speller," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012, pp. 551–554.

[41] L. Qin, L. Ding, and B. He, "Motor Imagery Classification by Means of Source Analysis for Brain Computer Interface Applications," *J Neural Eng*, vol. 2, no. 4, pp. 65–72, Dec. 2005. [Online]. Available : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1945182/

[42] P. L. Nunez, R. B. Silberstein, P. J. Cadusch, R. S. Wijesinghe, A. F. Westdorp, and R. Srinivasan, "A theoretical and experimental study of high resolution EEG based on surface Laplacians and cortical imaging," *Electroencephalography and Clinical Neurophysiology*, vol. 90, no. 1, pp. 40–57, Jan. 1994. [Online]. Available : http://www.sciencedirect.com/science/article/pii/0013469494901120

[43] A. Gevins, J. Le, N. K. Martin, P. Brickett, J. Desmond, and B. Reutter, "High resolution EEG : 124-channel recording, spatial deblurring and MRI integration methods," *Electroencephalography and Clinical Neurophysiology*, vol. 90, no. 5, pp. 337–358, May 1994. [Online]. Available : http://linkinghub.elsevier.com/retrieve/pii/0013469494900507

[44] P. L. Nunez and R. Srinivasan, *Electric fields of the brain : the neurophysics of EEG*, 2nd ed. Oxford : Oxford Univ. Press, 2006, oCLC : 265761217.

[45] J. Malmivuo and R. Plonsey, "Bioelectromagnetism. 2. Nerve and Muscle Cells," Jan. 1995, pp. 63–77.

[46] C. M. Michel, G. Lantz, L. Spinelli, R. G. de Peralta, T. Landis, and M. Seeck, "128-Channel EEG Source Imaging in Epilepsy : Clinical Yield and Localization Precision," *Journal of Clinical Neurophysiology*, vol. 21, no. 2, p. 71, Apr. 2004. [Online]. Available : https://journals.lww.com/clinicalneurophys/Abstract/2004/03000/128_Channel_EEG_Source_Imaging_in_Epilepsy_.1.aspx

[47] R. Grave de Peralta Menendez, S. Gonzalez Andino, G. Lantz, C. M. Michel, and T. Landis, "Noninvasive localization of electromagnetic epileptic activity. I. Method descriptions and simulations," *Brain Topogr*, vol. 14, no. 2, pp. 131–137, 2001.

[48] G. Huiskamp, M. Vroeijenstijn, R. v. Dijk, G. Wieneke, and A. C. v. Huffelen, "The need for correct realistic geometry in the inverse EEG problem," *IEEE Transactions on Biomedical Engineering*, vol. 46, no. 11, pp. 1281–1287, Nov. 1999.

[49] R. Grech, T. Cassar, J. Muscat, K. P. Camilleri, S. G. Fabri, M. Zervakis, P. Xanthopoulos, V. Sakkalis, and B. Vanrumste, "Review on solving the inverse problem in EEG source analysis," *Journal of NeuroEngineering and Rehabilitation*, vol. 5, p. 25, Nov. 2008. [Online]. Available : https://doi.org/10.1186/1743-0003-5-25

[50] J. Vorwerk, M. Clerc, M. Burger, and C. H. Wolters, "Comparison of Boundary Element and Finite Element Approaches to the EEG Forward Problem," *Biomedical Engineering / Biomedizinische Technik*, vol. 57, no. SI-1 Track-O, pp. 795–798, 2012. [Online]. Available : https://www.degruyter.com/view/j/bmte.2012.57.issue-s1-O/bmt-2012-4152/bmt-2012-4152.xml

[51] J. Kybic, M. Clerc, T. Abboud, O. Faugeras, R. Keriven, and T. Papadopoulo, "A common formalism for the Integral formulations of the forward EEG problem," *IEEE Transactions on Medical Imaging*, vol. 24, no. 1, pp. 12–28, Jan. 2005.

[52] S. Homma, T. Musha, Y. Nakajima, Y. Okamoto, S. Blom, R. Flink, and K.-E. Hagbarth, "Conductivity ratios of the scalp-skull-brain head model in estimating equivalent dipole sources in human brain," *Neuroscience Research*, vol. 22, no. 1, pp. 51–55, Mar. 1995. [Online]. Available : http://www.sciencedirect.com/science/article/pii/0168010295008803

[53] H. Hallez, B. Vanrumste, R. Grech, J. Muscat, W. De Clercq, A. Vergult, Y. D'Asseler, K. P. Camilleri, S. G. Fabri, S. Van Huffel, and I. Lemahieu, "Review on solving the forward problem in EEG source analysis," *Journal of NeuroEngineering and Rehabilitation*, vol. 4, p. 46, Nov. 2007. [Online]. Available : https://doi.org/10.1186/1743-0003-4-46

[54] R. Beauwens, "Iterative solution methods," *Applied Numerical Mathematics*, vol. 51, no. 4, pp. 437–450, Dec. 2004. [Online]. Available : http://www.sciencedirect.com/science/article/pii/S0168927404000935

[55] J. E. O. Guzman, A. Pillain, L. Rahmouni, and F. P. Andriulli, "On the preconditioning of the symmetric formulation for the EEG forward problem by leveraging on calderon formulas," in *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, Apr. 2016, pp. 755–758.

[56] M. Bebendorf, "Approximation of boundary element matrices," *Numer. Math.*, vol. 86, no. 4, pp. 565–589, Oct. 2000. [Online]. Available : https://link.springer.com/article/10.1007/PL00005410

[57] K. Zhao, M. N. Vouvakis, and J.-F. Lee, "The adaptive cross approximation algorithm for accelerated method of moments computations of EMC problems,"

IEEE Transactions on Electromagnetic Compatibility, vol. 47, no. 4, pp. 763–773, Nov. 2005.

[58] gcc.gnu.org, "The GNU Fortran Compiler : ATAN2." [Online]. Available : https://gcc.gnu.org/onlinedocs/gfortran/ATAN2.html

[59] A. Sultan, "CORDIC : How Hand Calculators Calculate," *THE COLLEGE MATHEMATICS JOURNAL*, vol. 40, no. 2, p. 6, 2009.

[60] ARITH, "http ://www.arithsymposium.org/," 2018. [Online]. Available : http://www.arithsymposium.org/

[61] CPU-World, "Intel C4004," 2003. [Online]. Available : http://www.cpu-world.com/CPUs/4004/Intel-C4004.html

[62] V. Zhislina, "Why has CPU frequency ceased to grow ? | Intel® Software," 2014. [Online]. Available : https://software.intel.com/en-us/blogs/2014/02/19/why-has-cpu-frequency-ceased-to-grow

[63] A. FOG, "Instruction tables. Lists of instructions latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs." 2011. [Online]. Available : https://www.agner.org/optimize/instruction_tables.pdf

[64] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm, "Exploiting Choice : Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," in *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, ser. ISCA '96. New York, NY, USA : ACM, 1996, pp. 191–202. [Online]. Available : http://doi.acm.org/10.1145/232973.232993

[65] M. D. Marino, "L2-Cache Hierarchical Organizations for Multi-core Architectures," in *Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Dec. 2006, pp. 74–83. [Online]. Available : https://link.springer.com/chapter/10.1007/11942634_9

[66] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, Reprinted from the AFIPS Conference Proceedings, Vol. 30 (Atlantic City, N.J., Apr. 18 #x2013 ;20), AFIPS Press, Reston, Va., 1967, pp. 483 #x2013 ;485, when Dr. Amdahl was at International Business Machines Corporation, Sunnyvale, California," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 3, pp. 19–20, 2007.

[67] G. Singer, "The History of the Modern Graphics Processor," 2013. [Online]. Available : https://www.techspot.com/article/650-history-of-the-gpu/

[68] L. W. Howes, P. Price, O. Mencer, O. Beckmann, and O. Pell, "Comparing FPGAs to Graphics Accelerators and the Playstation 2 Using a Unified Source Description," in *2006 International Conference on Field Programmable Logic and Applications*, Aug. 2006, pp. 1–6.

[69] A. Buttari, J. Dongarra, and J. Kurzak, "Limitations of the PlayStation 3 for High Performance Cluster Computing," Jul. 2007. [Online]. Available : http://eprints.ma.man.ac.uk/819/

[70] Nvidia, "NVIDIA TITAN V, la carte graphique pour PC la plus puissante au monde," 2018. [Online]. Available : https://www.nvidia.fr/titan/titan-v/

[71] ——, “CUDA Zone,” Jul. 2017. [Online]. Available : https://developer.nvidia.com/cuda-zone

[72] R. Adelman, N. A. Gumerov, and R. Duraiswami, “FMM/GPU-Accelerated Boundary Element Method for Computational Magnetics and Electrostatics,” *IEEE Transactions on Magnetics*, vol. 53, no. 12, pp. 1–11, Dec. 2017.

[73] CMP, “Price list - CMP : Circuits Multi-Projets,” 2018. [Online]. Available : https://mycmp.fr/technologies/price-list.html

[74] K. Barr, *ASIC Design in the Silicon Sandbox : A Complete Guide to Building Mixed-Signal Integrated Circuits*. New York : McGraw-Hill, 2007.

[75] O. Rahnama, D. Frost, O. Miksik, and P. H. S. Torr, “Real-Time Dense Stereo Matching With ELAS on FPGA-Accelerated Embedded Devices,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2008–2015, Jul. 2018.

[76] S. Perri, F. Frustaci, F. Spagnolo, and P. Corsonello, “Design of Real-Time FPGA-based Embedded System for Stereo Vision,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[77] M. Krim, “FPGA : une techno désormais au cœur de la stratégie d’AWS et Azure,” 2017. [Online]. Available : https://www.journaldunet.com/solutions/cloud-computing/1192645-les-puces-fpga-solution-miracle-pour-desengorger-les-datacenters/

[78] R. Flechaux, “FPGA : l’arme secrète d’OVH pour parer les attaques DDoS,” Oct. 2016. [Online]. Available : https://www.silicon.fr/fpga-arme-secrete-ovh-parer-attaques-ddos-159959.html

[79] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, “Features, Design Tools, and Application Domains of FPGAs,” *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1810–1823, Aug. 2007.

[80] S. Asano, T. Maruyama, and Y. Yamaguchi, “Performance comparison of FPGA, GPU and CPU in image processing,” in *2009 International Conference on Field Programmable Logic and Applications*, Aug. 2009, pp. 126–131.

[81] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, “Accelerating recurrent neural networks in analytics servers : Comparison of FPGA, CPU, GPU, and ASIC,” in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2016, pp. 1–4.

[82] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, “A Survey and Evaluation of FPGA High-Level Synthesis Tools,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[83] C. Li, Y. Bi, Y. Benezeth, D. Ginhac, and F. Yang, “High-level synthesis for FPGAs : code optimization strategies for real-time image processing,” *J Real-Time Image Proc*, vol. 14, no. 3, pp. 701–712, Mar. 2018. [Online]. Available : https://link.springer.com/article/10.1007/s11554-017-0722-3

[84] Y. Liang, K. Rupnow, Y. Li, D. Min, M. N. Do, and D. Chen, “High-level Synthesis : Productivity, Performance, and Software Constraints,”

JECE, vol. 2012, pp. 1 :1–1 :1, Jan. 2012. [Online]. Available : http://dx.doi.org/10.1155/2012/649057

[85] M. Pelcat, C. Bourrasset, L. Maggiani, and F. Berry, "Design productivity of a high level synthesis compiler versus HDL," in *2016 International Conference on Embedded Computer Systems : Architectures, Modeling and Simulation (SAMOS)*, Jul. 2016, pp. 140–147.

[86] Xilinx, "40g/100g Ethernet Core," 2018. [Online]. Available : https://www.xilinx.com/products/intellectual-property/40_100g_ethernet.html

[87] M. Puschel, J. M. F. Moura, J. R. Johnson, D. Padua, M. M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo, "SPIRAL : Code Generation for DSP Transforms," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 232–275, Feb. 2005.

[88] P. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Computer Generation of Hardware for Linear Digital Signal Processing Transforms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 17, no. 2, pp. 15 :1–15 :33, Apr. 2012. [Online]. Available : http://doi.acm.org/10.1145/2159542.2159547

[89] OpenCores, "Home : : OpenCores," 1999. [Online]. Available : https://opencores.org/

[90] FloPoCo, "Welcome to the FloPoCo project." [Online]. Available : http://flopoco.gforge.inria.fr/

[91] S. Hsiao, P. Wu, C. Wen, and P. K. Meher, "Table Size Reduction Methods for Faithfully Rounded Lookup-Table-Based Multiplierless Function Evaluation," *IEEE Transactions on Circuits and Systems II : Express Briefs*, vol. 62, no. 5, pp. 466–470, May 2015.

[92] F. d. Dinechin and M. Istoan, "Hardware Implementations of Fixed-Point Atan2," in *2015 IEEE 22nd Symposium on Computer Arithmetic*, Jun. 2015, pp. 34–41.

[93] PCI-SIG, "Welcome to PCI-SIG | PCI-SIG," 2018. [Online]. Available : https://pcisig.com/

[94] C. McGinnis, "PCI-SIG® Fast Tracks Evolution to 32gt/s with PCI Express 5.0 Architecture," Jun. 2017. [Online]. Available : https://www.businesswire.com/news/home/20170607005351/en/PCI-SIG%C2%AE-Fast-Tracks-Evolution-32GTs-PCI-Express

[95] PCI-SIG, "PCI-SIG - FAQ - PCI Express 3.0," Feb. 2014. [Online]. Available : https://web.archive.org/web/20140201172536/http://www.pcisig.com/news_room/faqs/pcie3.0_faq/#EQ2

[96] Xilinx, "Industry's First Gen3 x 16 PCIe Solution Built into a Programmable Device," 2016. [Online]. Available : https://www.xilinx.com/video/technology/industrys-first-gen3x16-pcie-solution.html

[97] J. Gong, J. Chen, H. Wu, F. ye, S. Lu, J. Cong, and T. Wang, "EPEE : an efficient PCIe communication library with easy-host-integration property for FPGA accelerators (abstract only)," Feb. 2014, pp. 255–255.

[98] M. Vesper, D. Koch, K. Vipin, and S. A. Fahmy, "JetStream : An open-source high-performance PCI Express 3 streaming library for FPGA-to-Host and FPGA-to-FPGA communication," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2016, pp. 1–9.

[99] M. Jacobsen and R. Kastner, "RIFFA 2.0 : A reusable integration framework for FPGA accelerators," in *2013 23rd International Conference on Field programmable Logic and Applications*, Sep. 2013, pp. 1–8.

[100] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME : Toward 100 Gbps as Research Commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sep. 2014.

[101] BittWare, "XUPP3r PCIe FPGA Board," 2017. [Online]. Available : https://www.bittware.com/fpga/xilinx/boards/xupp3r/

[102] Xilinx, "Zynq-7000 SoC," 2018. [Online]. Available : https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[103] R. Bansal and A. Karmakar, "Efficient integration of coprocessor in LEON3 processor pipeline for System-on-Chip design," *Microprocessors and Microsystems*, vol. 51, pp. 56–75, Jun. 2017. [Online]. Available : https://www.sciencedirect.com/science/article/pii/S0141933117302090

[104] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The MOLEN polymorphic processor," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1363–1375, Nov. 2004.

[105] P. Horrein, P. Gleonec, E. Libessart, A. Lalevée, and M. Arzel, "Ouessant : Flexible integration of dedicated coprocessors in Systems on Chip," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2016, pp. 1493–1496.

[106] Gaisler, "LEON3." [Online]. Available : https://www.gaisler.com/index.php/products/processors/leon3

[107] P. Horrein, B. Porteboeuf, and A. Lalevée, "Ouessant : Microcontroller approach for flexible accelerator integration and control in System-on-Chip," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2016, pp. 1–4.

[108] P. H. Horrein, "Overview - Ouessant - Gestion de projets," 2016. [Online]. Available : https://redmine.telecom-bretagne.eu/projects/ouessant

[109] I. FPGA, "Intel® Programmable Acceleration Card with Arria® 10 GX FPGA," 2018. [Online]. Available : https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/acceleration-card-arria-10-gx.html

[110] Intel, "Open Programmable Acceleration Engine — OPAE," 2017. [Online]. Available : https://opae.github.io/latest/index.html

[111] ——, "Acceleration Stack for Intel Xeon CPU with FPGAs Core Cache Interface (CCI-P) Reference Manual," 2018. [Online]. Available : https://www.intel.com/content/www/us/en/programmable/documentation/buf1506187769663.html

[112] E. Hertz and P. Nilsson, "A methodology for parabolic synthesis of unary functions for hardware implementation," in *2008 2nd International Conference on Signals, Circuits and Systems*, Nov. 2008, pp. 1–6.

[113] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[114] R. Andraka, "A Survey of CORDIC Algorithms for FPGA Based Computers," in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, ser. FPGA '98. New York, NY, USA : ACM, 1998, pp. 191–200. [Online]. Available : http://doi.acm.org/10.1145/275107.275139

[115] A. Rodriguez-Garcia, L. Pizano-Escalante, R. Parra-Michel, O. Longoria-Gandara, and J. Cortez, "Fast fixed-point divider based on Newton-Raphson method and piecewise polynomial approximation," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec. 2013, pp. 1–6.

[116] H. C. Neto and M. P. Vestias, "Very low resource table-based FPGA evaluation of elementary functions," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec. 2013, pp. 1–6.

[117] Xilinx, "Virtex-7 FPGA Family," 2017. [Online]. Available : https://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html#documentation

[118] ——, "Virtex UltraScale," 2018. [Online]. Available : https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale.html#documentation

[119] Intel, "Intel Stratix 10 Device Datasheet," 2016. [Online]. Available : https://www.intel.com/content/www/us/en/programmable/documentation/mcn1441092958198.html

[120] F. d. Dinechin and B. Pasca, "Large multipliers with fewer DSP blocks," in *2009 International Conference on Field Programmable Logic and Applications*, Aug. 2009, pp. 250–255.

[121] K. Kunaraj and R. Seshasayanan, "Leading one detectors and leading one position detectors - An evolutionary design methodology," *Canadian Journal of Electrical and Computer Engineering*, vol. 36, no. 3, pp. 103–110, 2013.

[122] K. H. Abed and R. E. Siferd, "VLSI Implementations of Low-Power Leading-One Detector Circuits," in *Proceedings of the IEEE SoutheastCon, 2006*, Mar. 2006, pp. 279–284.

[123] M. P. Vestias and H. C. Neto, "Revisiting the Newton-Raphson Iterative Method for Decimal Division," in *2011 International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2011, pp. 138–143.

[124] E. Libessart, M. Arzel, C. Lahuec, and F. Andriulli, "A scaling-less Newton-Raphson pipelined implementation for a fixed-point inverse square root operator," in *2017 15th IEEE International New Circuits and Systems Conference (NEWCAS)*, Jun. 2017, pp. 157–160.

[125] ——, "Implantation en virgule fixe d'un opérateur de calcul d'inverse à base de Newton-Raphson, sans normalisation et sans bloc mémoire," in *GRETSI 2017 : 26ème colloque du Groupement de Recherche en Traitement du*

Signal et des Images, Juan-Les-Pins, France, Sep. 2017. [Online]. Available : https://hal.archives-ouvertes.fr/hal-01630144

[126] G. R. Prabhu, B. Johnson, and J. S. Rani, "FPGA Based Scalable Fixed Point QRD Core Using Dynamic Partial Reconfiguration," in *2015 28th International Conference on VLSI Design*, Jan. 2015, pp. 345–350.

[127] E. Libessart, M. Arzel, C. Lahuec, and F. Andriulli, "A Scaling-Less Newton-Raphson Pipelined Implementation for a Fixed-Point Reciprocal Operator," *IEEE Signal Processing Letters*, vol. 24, no. 6, pp. 789–793, Jun. 2017.

[128] M. Sadeghian, J. E. Stine, and E. G. Walters, "Optimized cubic chebyshev interpolator for elementary function hardware implementations," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, Jun. 2014, pp. 1536–1539.

[129] E. Libessart, M. Arzel, C. Lahuec, and F. Andriulli, "40 Gop/s/mm lt ;sup gt ;2 lt ;/sup gt ; fixed-point operators for Brain Computer Interface in 65 nm CMOS," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–4.

[130] Libessart, "Aperçu - Scaling-less Newton-Raphson - Gestion de projets," 2017. [Online]. Available : https://redmine.telecom-bretagne.eu/projects/scaling-less-newton-raphson

[131] X. Hu, R. G. Harber, and S. C. Bass, "Expanding the range of convergence of the CORDIC algorithm," *IEEE Transactions on Computers*, vol. 40, no. 1, pp. 13–21, Jan. 1991.

[132] Nvidia, "Technologies d'interconnexion NVLink et NVSwitch," 2018. [Online]. Available : https://www.nvidia.fr/data-center/nvlink/

[133] L. Rahmouni, R. Mitharwal, and F. P. Andriulli, "A novel volume integral equation for solving the Electroencephalography forward problem," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Aug. 2015, pp. 4061–4064.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Interface cerveau-machine : de nouvelles perspectives grâce à l'accélération matérielle

Mots clés : Adéquation algorithme-architecture, Electronique numérique, FPGA, ASIC, EEG, Interface cerveau-machine

Résumé : Les interfaces cerveau-machine (ICM) permettent de contrôler un appareil électronique grâce aux signaux cérébraux. Plusieurs méthodes de mesure de ces signaux, invasives ou non, peuvent être utilisées. L'électro-encéphalographie (EEG) est la méthode non-invasive la plus étudiée car elle propose une bonne résolution temporelle et le matériel nécessaire est bien moins volumineux que les systèmes de mesure des champs magnétiques. L'EEG a cependant une faible résolution spatiale, ce qui limite les performances des ICM utilisant cette méthode de mesure. Ce souci de résolution spatiale peut être réglé en utilisant le problème inverse de l'EEG, qui permet de passer des potentiels mesurés en surface à une distribution volumique des sources de courant dans le cerveau. Le principal verrou de cette technique est le temps nécessaire (plusieurs heures) pour calculer avec une station de travail la matrice permettant de résoudre le problème inverse. Dans le cadre de cette thèse, nous avons étudié les solutions actuelles pour accélérer matériellement la conception de cette matrice. Nous avons ainsi proposé, conçu et testé une architecture électronique dédiée à ces traitements pour ICM. Les premiers résultats démontrent que notre solution permet de passer de plusieurs heures de calcul sur une station de travail à quelques minutes sur circuit reconfigurable. Cette accélération des traitements d'imagerie par EEG facilitera grandement la recherche sur l'utilisation du problème inverse et ouvrira ainsi de nouvelles perspectives pour le domaine de l'ICM.

Title : Brain-computer interface: new perspectives through hardware acceleration

Keywords : architecture-silicon adequacy, digital electronics, FPGA, ASIC, EEG, brain-computer interface

Abstract : Brain-Computer Interfaces (BCI) are systems that use brain activity to control an external device. Various techniques can be used to collect the neural signals. The measurement can be invasive or non-invasive. Electroencephalography (EEG) is the most studied non-invasive method. Indeed, EEG offers a fine temporal resolution and ease of use but its spatial resolution limits the performances of BCI based on EEG. The spatial resolution of EEG can be improved by solving the EEG inverse problem, which allows to determine the distribution of electrical sources in the brain from EEG. Currently, the main difficulty is the time needed (several hours) to compute the matrix which is used to solve the EEG inverse problem. This document describes the proposed solution to provide a hardware acceleration of the matrix computation. A dedicated electronic architecture has been implemented and tested. First results show that the proposed architecture divides the calculation time by a factor of 60 on a programmable circuit. This acceleration opens up new perspectives for EEG BCI.