UNIVERSITÉ DU QUÉBEC À MONTRÉAL

INFRASTRUCTURE DE CALCUL EN TEMPS RÉEL POUR LE PROTOCOLE DE COMMUNICATION SANS-FIL LTE

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

OUAJDI BRINI

OCTOBRE 2017

AVANT-PROPOS

Les réseaux cellulaires ont connu une dizaine de versions différentes jusqu'à ce jour forçant les fournisseurs des services de télécommunications à modifier leurs équipements constamment. En raison de cette évolution considérable au cours des dernières décennies, de nombreuses recherches ont été menées pour réduire le coût de ces transitions en temps et argent. Ces recherches se sont principalement concentrées sur la conversion de cette technologie du matériel vers le logiciel au niveau de la couche physique du protocole en particulier. Une nouvelle approche utilise les ordinateurs de bureau comme station de base cellulaire qui peut, à un certain degré, remplacer l'équipement non évolutif actuellement utilisé. Les fournisseurs des services de télécommunications, au niveau des stations de base, utilisent des systèmes modulaires pour traiter l'information échangée qui sont basés sur des processeurs temps réel dédié et cette approche rend la modification physique de ces derniers nécessaire pour mettre à niveau le protocole. En conséquence les algorithmes utilisés dans les stations de base cellulaires sont intégrés dans les processeurs comme microprogramme ou gravés sur une puce intégrée dédiée. Si une recherche pouvait prouver que certains algorithmes ou techniques convertis en logiciel peuvent respecter le standard en ce qui concerne les contraintes temporelles, une couche physique purement logicielle pourrait voir le jour pour rendre la commutation d'une version à l'autre du protocole aussi simple qu'une modification d'un code de programme. Le matériel informatique grand public peut-il traiter des trames d'un réseau cellulaire en temps réel ? Cette étude se penche sur cette question dans le cas du standard le plus récent LTE et en particulier l'utilisation des technologies de parallélisation pour résoudre cette problématique.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

3G	Troisième génération
3GPP	Third Generation Partnership Project
4G	Quatrième génération
ASIC	Circuit intégré propre à une application
AWGN	Bruit blanc gaussien
BER	Taux d'erreur sur les bits
BPSK	Modulation de phase binaire
BW	Bande passante
BS	Station de base
CAN	Convertisseur analogique numérique
CP	Préfixe cyclique
CPU	Unité centrale de traitement
CRC	Contrôle de Redondance Cyclique
CW	Mot de code binaire
DC	sous-porteuse à valeur nulle
DFT	Transformation de Fourier discrète
DRS	Signal référence pour démodulation
eNodeB	Station de base des réseaux mobiles
FDD	Duplex par séparation fréquentielle

FFT	Transformation de Fourier rapide
GFLOPS	Giga opération en virgule flottante par seconde
GPU	Unité de traitement graphique
ICI	Interférence inter-porteuse
IDFT	Transformation de Fourier discrète inverse
IFFT	Transformation de Fourier rapide inverse
LTE	Long Term Evolution
LUT	Table de correspondance
Mbps	Mégaoctet par seconde
MHz	Mégahertz
MIMO	Entrées sorties multiples
MMSE	Erreur quadratique moyenne minimale
M_{sc}^{PUSCH}	Nombre de sous-porteuses dans le canal partagé en mode « Uplink » à envoyer
M_{symb}	Nombre de symbole utilisé
M_{symb}^{layer}	Nombre de symbole à transmettre par matrice
M_{sc}^{RS}	Nombre de sous-porteuses par signal de référence
N_{RB}^{UL}	Nombre de bloc de ressource en mode « Uplink »par symbole
N_{sc}^{RB}	Nombre de sous-porteuses par bloc de ressource
N_{ZC}^{RS}	Taille de la séquence « Zadoff-Chu » par signal de référence
OFDM	Accès multiple par division en fréquence
OFDMA	Accès multiple par division en fréquence orthogonal
OR	Opération réel
PAPR	Facteur de crête

QPSK	Modulation de phase en quadrature
QAM	Modulation d'amplitude en quadrature
RBG	Groupe de Bloc de ressources
RG	Bloc de ressources
SIMD	Instruction unique sur des données multiples
SISD	Instruction unique sur des données uniques
SISO	Entrées / sorties simples
SC	Sous-porteuse
SF	Sous-trame LTE
TDD	Duplex par séparation temporelle
TFLOPS	Tera opération en virgule flottante par seconde
TB	Bloc de transport
UE	Équipement utilisateur

RÉSUMÉ

Ce mémoire de maîtrise étudie la possibilité de conversion des techniques utilisées dans la couche physique du protocole de communication LTE en logiciels. Plus précisément, on traite la faisabilité pratique de concevoir une station de base LTE sur un ordinateur à usage général. Puisque le coût de la mise à niveau des équipements d'une station de base LTE est très élevé pour les entreprises œuvrant dans ce domaine, une solution alternative qui remplacerait l'ancienne approche, en tout ou en partie, peut être considérée innovatrice pour eux. Comme LTE utilise des techniques communes avec d'autres protocoles sans-fils, cette recherche est également avantageuse pour les futures études sur la virtualisation des protocoles de communication. Cependant, la réalisation de l'objectif exige un traitement de données très rapide pour maintenir le fonctionnement en temps réel, c'est pourquoi dans ce travail, nous identifierons et décrirons non seulement les exigences nécessaires pour développer un tel système, mais aussi le flux des données qu'il faut traiter afin de s'approcher au maximum d'une virtualisation complète et fiable. Si cela n'est pas possible, on identifiera les goulets d'étranglement qui peuvent empêcher les fabricants d'équipements LTE de passer à une solution de la couche physique purement logicielle .

Mots-clés : CUDA, LTE, VIRTUALISATION, 4G, 3G, 3GPP.

CHAPITRE I

INTRODUCTION

1.1 Introduction

1.1.1 Contexte

Depuis l'émergence des réseaux grands publics dans les années 80 et la création du *3GPP* à la fin des années 90, les protocoles de communication sans fil évoluent constamment, maintenant qu'il existe une bonne coordination entre les continents sur les spécifications techniques des protocoles de réseau cellulaire, les détails sur toutes les couches du protocole sont réunis dans un paquet de documents centralisés que nous appelons «The Mobile Broadband Standards» qui assure l'interopérabilité globale entre les fournisseurs de services et les fabricants de matériels.

1.1.2 Approche

Pour déterminer les exigences requises pour une éventuelle virtualisation de la couche physique de LTE au niveau d'une station de base, notre tâche consiste à implémenter un récepteur LTE fonctionnel capable de traiter les trames radio reçues en temps réel. Une telle approche aidera à identifier les goulets d'étranglement dans les différents modules de la couche physique et ainsi calculer le temps de traitement total qui nous informera sur la faisabilité de l'approche GPU.

1.1.3 Objectif du mémoire

Ce travail se concentre sur la mise en œuvre des modules récepteurs de la couche physique LTE au niveau de la station de base, puis teste les performances avec des trames générées par l'outil « MATLAB LTE Toolbox » qui est utilisé aussi pour la vérification des résultats. Chaque module implémenté doit être chronométré afin d'identifier les fonctions à haute complexité computationnelle. Le temps d'exécution confirmera si le traitement d'une trame LTE peut se faire en moins de 10 ms, qui est la durée maximale allouée par le protocole.

Des statistiques sur le temps d'exécution de tous les modules seront déduites pour vérifier l'aspect temps réel du programme afin de pouvoir estimer la puissance computationnelle minimale requise pour passer à une solution logicielle pour différents modes de transmission.

1.1.4 Études précédentes

Depuis la première version de la plate-forme de développement CUDA en 2007, de nombreux chercheurs ont été tentés d'utiliser la programmation GPU pour faire du traitement sur des données parallèles dans le domaine du la communication afin de remplacer l'approche habituelle par FPGA, qui sont en général plus couteux.

Parmi les recherches précédentes qui se sont penchées sur cette problématique ou une autre similaire, on note :

- (Vieira, 2011) Qui a implémenté une mini station de base LTE réceptrice qui traite des fonctions LTE comme la suppression du préfixe cyclique et le calcul de la FFT des symboles avec le GPU, pour enfin faire un décalage circulaire afin d'éliminer les échantillons qui appartiennent à la SC au milieu du signal. L'auteur a testé son programme sur des trames appartenant à la

bande 5 MHz, dont le nombre d'échantillons par symbole dans la trame reçue est 512. En comparant le temps d'exécution d'un même programme avec le CPU (611 ms) et le GPU (2,71 ms), l'auteur a conclu que ce dernier est presque 250 fois plus rapide et retournant le même résultat.

Dans (Zheng *et al.*, 2013), une autre étude intéressante a été réalisée par une équipe de chercheurs afin de comparer le temps d'exécution de certaines techniques utilisées dans LTE avec les GPU. Les modules du protocole LTE qui ont été testés sur GPU sont : FFT, IFFT, l'estimation et l'égalisation du canal de transmission et finalement un démodulateur (16QAM, 64QAM). Dans leurs tests, ces fonctions s'appliquaient indépendamment sur une sous-trame reçue de 1ms de durée avec les modes de transmissions SISO et MIMO. Les trames ne sont pas testées pour simuler leur fonctionnement en temps réel, mais plutôt pour estimer le temps de traitement de ces techniques sur les cartes graphiques.

- Dans une autre recherche (Li *et al.*, 2014), la couche physique d'une station de base convertie en logiciel est exécutée sur quatre GPU pour traiter des trames WiFi (standard IEEE 802.11), qui est un protocole populaire dans la domotique. Quatre cartes graphiques ont été utilisées pour tester les nouvelles techniques utilisées par ce protocole, vu sa similitude avec LTE. Leurs résultats peuvent nous donner un aperçu sur la mise en application de cette approche en tenant compte qu'ils sont obtenus dans un environnement non-temps réel avec un échantillonnage 64-points pour les opérations de FFT, IFFT.

Un autre système de réseau cellulaire au niveau de la couche physique convertit en logiciel s'exécutant sur un GPU est utilisé comme modem et traite des trames radio générées par le logiciel « USRP2 » (S. *et al.*, 2014). Les auteurs nous donnent des résultats d'un traitement en temps réel complet

d'une trame 3G, mais ne spécifient pas la bande passante ni la taille des symboles, ce qui est important étant donné que les trames utilisant la bande 20 MHz sont les plus complexes à traiter. Dans leurs résultats, les temps de traitement des opérations FFT sur le GPU sont trop courts (0,31 ms) ; donc, il est peu probable qu'une trame FFT / IFFT de 2048 points ait été testée.

- Une implémentation des modules de transformations de Fourrier et d'un démodulateur QPSK en programme traditionnel et avec CUDA ont été testé pour comparer le débit à la sortie d'un receveur de trame LTE (Bhattacharjee *et al.*, 2014). Les auteurs ont présenté le lien entre l'augmentation du nombre d'utilisateurs avec la quantité des données qui transitent par la mémoire du GPU et son effet sur le débit global ainsi que le nombre de *Kernel* CUDA nécessaire pour traiter jusqu'à 20 utilisateurs en même temps.

1.2 Réseau LTE et outils de développement

1.2.1 Introduction

LTE, ou le réseau 4G, est la génération actuelle des technologies de télécommunication mobile. Selon le standard, LTE offre une vitesse de transfert allant jusqu'à 86,4 Mbps utilisant une seule antenne et 326,4 Mb/s avec le mode de transmission à 4 antennes (Agilent, 2010). LTE apporte de nombreux avantages aux réseaux cellulaires, grâce à sa bande passante utilisée allant de 1.4 MHz à 20 MHz. Les opérateurs du réseau choisissent quelle bande utiliser en fonction de la qualité du signal et fournissent différents services comme la voix et les données. L'objectif de cette conception du protocole est d'améliorer l'efficacité spectrale par rapport aux réseaux 3G, ce qui a permis aux opérateurs de fournir un débit amélioré sur un spectre donné.

1.2.2 Architecture LTE

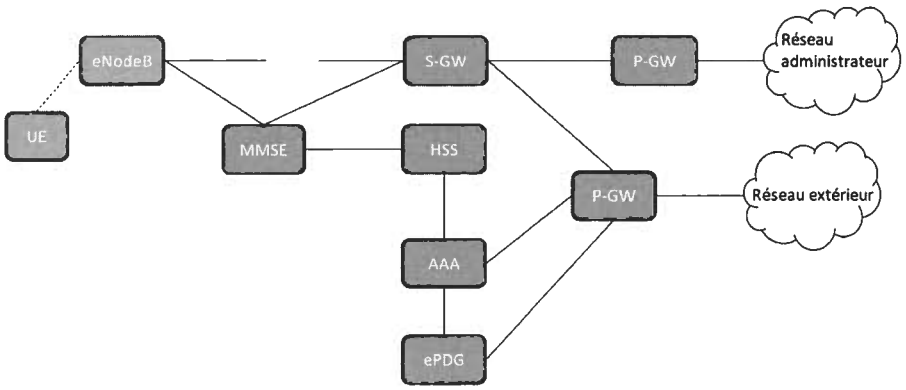


Figure 1.1: Réseau LTE.(TS-23.401, 2017)

Le réseau LTE est basé sur SAE (System Architecture Evolution) qui est une évolution du réseau GPRS (General Packet Radio Service), et comme on peut le voir dans la figure 1.1, LTE est devenu un réseau complètement dépendant du protocole IP. Les principales interfaces entre toutes les entités du système partagent des protocoles similaires pour faciliter la communication et la mise à niveau. C'est une architecture à commutation de paquets, largement inspirée par son prédécesseur (3G), et ses principales entités sont :

— **eNodeB** : est la station de base LTE qui communique directement avec l'équipement des utilisateurs (UE). Elle est équivalente à la station de transmission de base (BTS) dans les réseaux GSM.

— **UE (User Equipment)** : c'est l'équipement (ou terminal) utilisateur, qui est habituellement le téléphone mobile connecté au réseau pour transmettre de la voix et les données internet.

— **MME (Mobility Management Entity)** : c'est l'entité qui gère l'accès au réseau, elle s'occupe de toute communication avec les clients, comme l'enregistrement des abonnements, le marquage la retransmission. Le module est responsable de l'authentification des utilisateurs par exemple en choisissant

le SGW pour l'utilisateur. Il s'occupe aussi du transfert et de l'authentification et d'autres procédures qui maintiennent une communication continue.

— **SGW (Serving Gateway)** : fonctionne comme un routeur et s'occupe de tous les paquets échangés avec tous les utilisateurs en jouant le rôle de relais entre les anciennes technologies 3GPP et LTE. Le trafic global d'un UE en particulier passe par un seul SGW.

PDN-GW (Packet Data Network Gateway) : c'est le point de sortie des paquets internes d'un UE vers les réseaux externes (passerelle PDN), donc il fonctionne comme une interface entre les protocoles 3GPP et non 3GPP. Un UE peut avoir plusieurs PDN-GW (appelé PGW aussi) qui lui sont assignés.

HSS (Home Subscriber Server) : il agit comme la base de données principale du réseau LTE et contient des informations comme les comptes utilisateurs, abonnements, sessions d'appel et autres données utiles.

ePDG (Evolved Packet Data Gateway) : c'est une passerelle qui sécurise les paquets qui circulent entre les UE et la voie aux réseaux externes. À cette fin, l'ePDG établit un tunnel IPsec avec les UE (Davis, 2001).

AAA (Serveur 3GPP) : offre des services comme l'authentification et l'autorisation d'accès aux utilisateurs non 3GPP.

La couche physique LTE a été conçue pour le mode duplex intégral, de sorte que l'information est transmise dans les deux sens sans affecter les performances, même si le mode TDD est permis aussi, le mode FDD est le plus utilisé par les fournisseurs des services de communication.

1.2.3 Conception et paramètres

L'objectif principal de la conception du réseau LTE est d'atteindre une meilleure performance comparée à ses prédécesseurs, ce qui fait que LTE a un débit plus

élevé, une latence plus faible et a été optimisé pour les paquets IP. La couche physique est conçue pour supporter des modes de transmission flexibles avec l'introduction des échanges par plusieurs antennes.

La couche physique LTE a été conçue pour fonctionner dans des bandes de fréquences allant de 1,4 à 20 MHz. Les modes de transmissions sont : le mode *Uplink* (communication de l'UE vers l'eNodeB) et le mode *Downlink* (de l'eNodeB vers UE). Le multiplexage s'effectue via les techniques SC-FDMA/OFDMA (Yang, 2010). L'espacement des sous-porteuses dans le spectre est de 15 kHz. Les techniques de modulation utilisées dans tous les modes LTE sont : QPSK, 16QAM ou 64QAM, elles sont choisies en fonction de la qualité du canal de transmission. Ces techniques seront discutés plus en détail dans les prochaines sections.

1.2.4 Entités de base

Sous-porteuses

Les sous-porteuses sont fortement utilisées dans les systèmes de communication par répartition en fréquence, puisque LTE est basé sur OFDM, Le spectre utilisé est divisé en plusieurs sous-porteuses à bande étroite qui se chevauchent orthogonalement. un symbole LTE représente toutes les sous-porteuses appartenant à une même période de 0.5 ms. Cette technique permet d'annuler les interférences entre ces derniers qu'on appelle ISI (inter Symbol interference), et cela sans insérer des bandes de protection pour les éviter. OFDM est sensible au décalage fré-quentiel à cause de cette orthogonalité des sous-porteuses, donc un décalage en fréquence se traduit en bruit sur les symboles (ICI). Cet avantage a permis à OFDM d'être l'un des meilleurs systèmes d'efficacité spectrale, comme on peut le voir dans la figure 1.2.

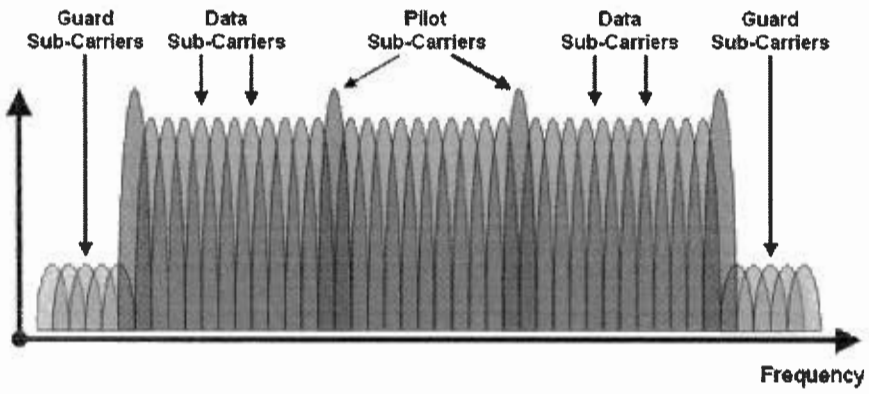


Figure 1.2: Sous-porteuses dans OFDM (NI, 2016)

Blocs de ressources

Les RB (Resource Blocks) d'une durée de 0.5 ms sont organisés en groupe de 12 sous-porteuses dans le domaine de fréquence et en 7 symboles dans le domaine temporel. Dans une autre configuration plus robuste dite « à préfixe étendu », 6 symboles sont utilisés dans un RB. La figure 1.3 illustre une représentation matricielle d'un RB à 7 symboles.

Grille de ressources

Une trame physique LTE est organisée en plusieurs RB qui forment une matrice appelée grille de ressources. Cette dernière facilite les attributions du spectre entre les UE. Dans le domaine temporel, la grille contient 10 sous-trames, chacune d'eux ayant 14 symboles chacune, donc 140 pour former une grille.
Dans le domaine de fréquence, la grille de ressources est organisée en RB, de 6 à 100 blocs précisément et cette configuration est valide pour chaque antenne de transmission. Les RB sont organisés uniformément autour d'une sous-porteuse centrale (centre du spectre) à valeur nulle appelée « DC Sub-Carrier ». L'eNodeB attribue toujours au moins une paire de blocs de ressources par utilisateur dans

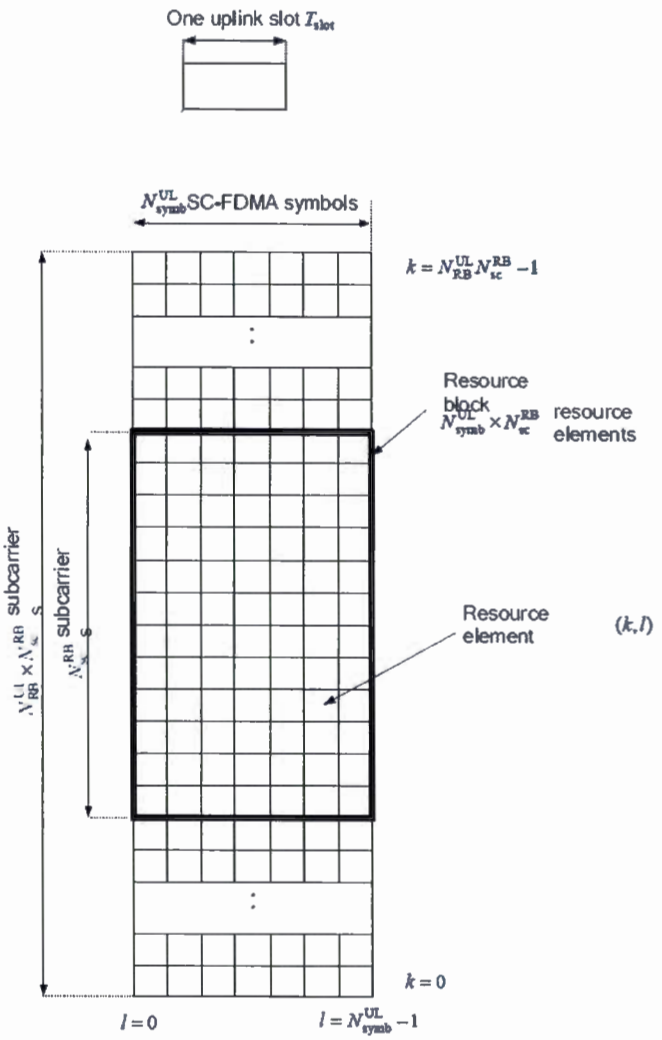


Figure 1.3: Bloc de ressource (TS-36.211, 2015)

le domaine temporel, étant donné chaque RB s'étend sur une période de 0.5 ms, au moins une durée de 1 ms est assignée par UE.

Élément de ressource

Chaque élément de la grille de ressources est appelé RE (Resource Element) et est défini de manière unique par une paire d'une sous-porteuse et d'un symbole temporel.

Canaux et signaux

Il existe deux types de données au niveau de la couche physique du protocole LTE : les données qui arrivent des couches supérieures contenant les informations des UE qui sont appelées canaux, et les données générées à la couche physique elle-même, comme les signaux de référence ou les informations de contrôle inutiles aux couches supérieures. On les appelle signaux.

Les canaux physiques du mode *Downlink* LTE, comme indiqué dans la figure 1.4 sont formés de :

— PDSCH (Physical downlink shared channel) : contient les données des utilisateurs.

— PDCCH (Physical downlink control channel) : contient les informations de contrôle.

— PHICH (Physical hybrid automatic repeat request indicator channel) : utilisé pour ACK/NACK.

— PCFICH (Physical control format indicator channel) : contient l'information concernant le nombre de symboles contenus dans les signaux PDCCH et PHICH.

— PBCH (Physical broadcast channel) : Utilisé pour donner des informations globales sur l'eNodeB comme le MIB (Master Information Block) contenant la configuration actuellement utilisée.

Les signaux physiques du mode *Downlink* LTE sont :

— le signal de référence (Les RE aux positions pilotes) : utilisé pour l'estimation du canal sur lequel la grille de ressource a été envoyée ; et

— les Signaux de synchronisation (PSS et SSS) : utilisés dans la synchronisation

temporelle entre les UE et l'eNodeB, l'identité de la station de base « cell ID » peut être déduite avec ces signaux.

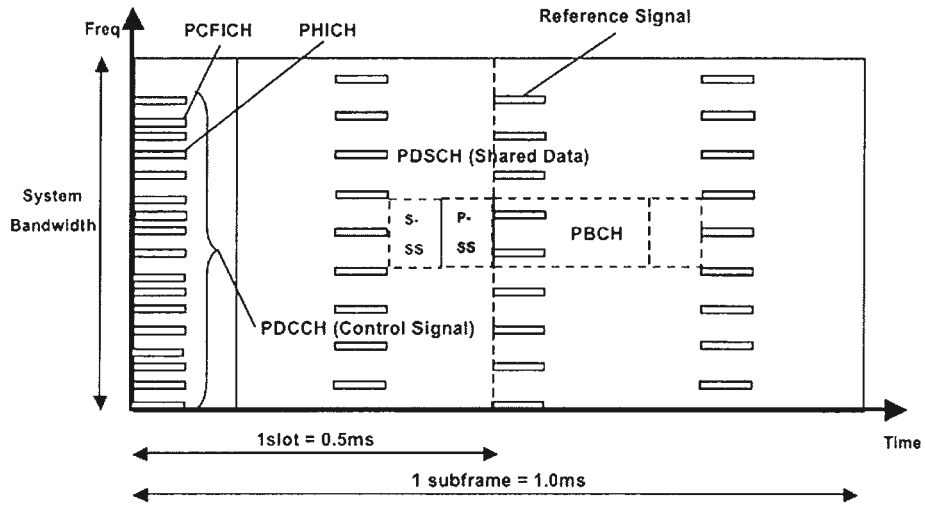


Figure 1.4: Sous-trame en mode *Downlink*(Artizanetworks., 2016)

Les canaux physiques du mode *Uplink* comme indiqué sur la figure 1.5 comprennent :

PUSCH (Physical uplink shared channel) : contient les données de l'utilisateur.

PUCCH (Physical uplink control channel) : utilisé pour ACK / NACKs, RI (Rank Index), CQI (Channel Quality Indicator) et SR (Scheduling Request).

PRACH (Physical random access channel) : transporte un préambule aléatoire pour demander l'accès au réseau.

Les signaux physiques du mode *Uplink* LTE sont :

UL-RS (Demodulation reference signal) : envoyé dans chaque 4e symbole de chaque tranche. Il est utilisé pour l'estimation du canal à la réception.

— SRS (Sounding reference signal) : il est optionnel et n'est pas associé à la transmission des données de l'UE, mais utilisé pour l'estimation de la qualité

du canal à la réception.

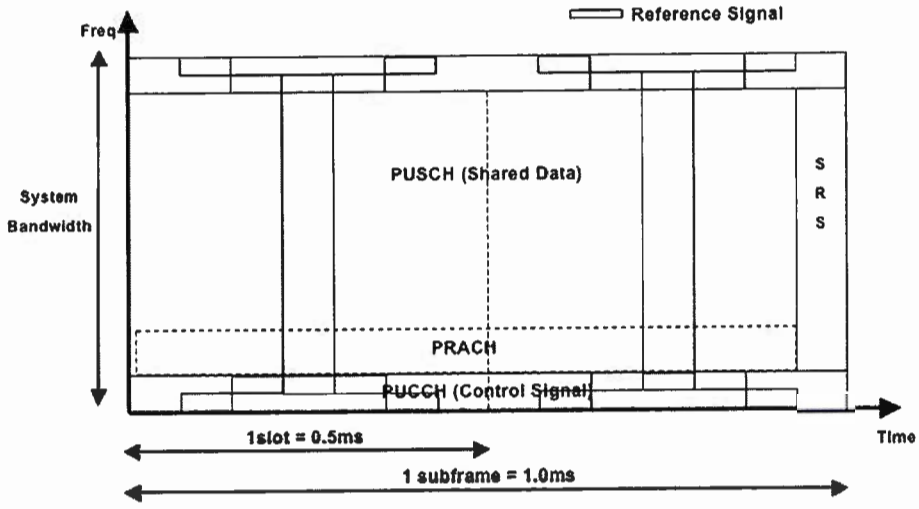


Figure 1.5: Sous-trame LTE dans le mode *Uplink* (Artizanetworks., 2016)

Préfixe cyclique (CP)

Lorsqu'un symbole est envoyé via un canal physique, un retard créé par la propagation du signal prenant des chemins différents peut causer la réception de plusieurs copies d'une même trame. Pour résoudre ce problème, un préfixe cyclique est ajouté à chaque symbole, constitué d'un ensemble d'échantillons pris de son extrémité et attaché à son début. Le but est d'ajouter un temps de garde entre deux symboles successifs. Si la longueur d'un CP est plus longue que le délai de propagation maximum du canal, il n'y aura pas d'ISI (Inter Symbol Interference), ce qui signifie que deux symboles successifs n'interféreront pas. Il évite également l'ICI (Inter Carrier Interference) qui cause la perte d'orthogonalité entre les sous-porteuses, car il utilise une copie de la dernière partie du symbole qui joue le rôle d'un intervalle de garde.

LTE a deux types de CP : normal et étendu. Dans le mode de préfixe normal, nous avons 7 symboles par tranche et la longueur du CP est courte, alors que dans le mode préfixe étendu, une tranche contient 6 symboles avec un plus long CP. La

figure 1.6 illustre le traitement nécessaire pour générer les CP dans le domaine temporel avant de les envoyer sur un canal physique.

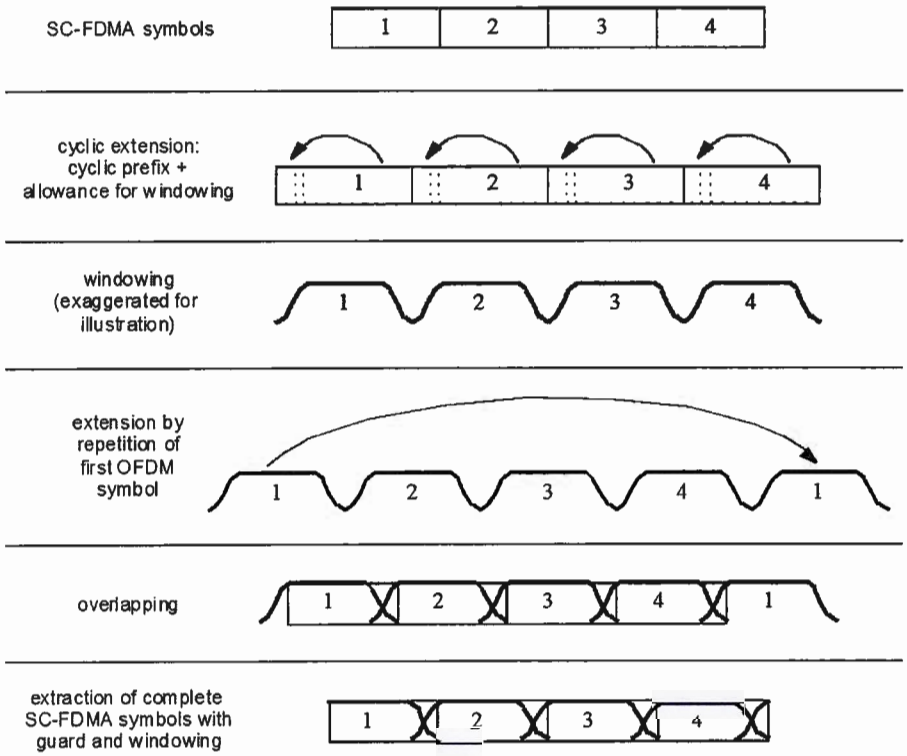


Figure 1.6: Préfixe cyclique CP (Matlab, 2016)

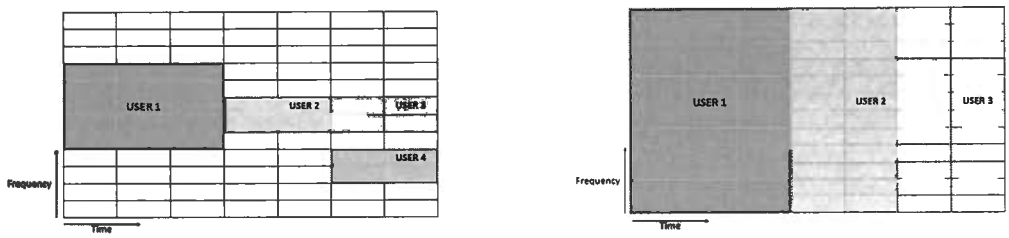
Le tableau 1.1 présente certains paramètres du réseau LTE.

Tableau 1.1: Configuration LTE

BW	1,4 MHz	3 MHz	5 MHz	10 MHz	15 MHz	20 MHz
Échantillons par symbole	128	256	512	1024	1536	2048
Symbole #0 CP	10	20	40	80	120	160
Symbole #1-6 CP	9	18	36	72	108	144
Durée d'une SF	1 ms					
Espacement entre les SC	15 kHz					

SC-FDMA et OFDMA

OFDMA est une technique utilisée dans les systèmes de télécommunication où la bande passante disponible est divisée en sous-bandes en fréquences qui peuvent transporter de multiples flux de données codées en parallèle, cette technique est une amélioration de OFDM qui est principalement utilisée dans beaucoup de protocoles de couche physique moderne comme les réseaux DSL et autres technologies sans-fil. Les avantages d'OFDMA proviennent du fait qu'elle utilise plusieurs bandes étroites et lentes au lieu d'une bande large à modulation rapide, ce qui améliore les communications duplex intégrales ainsi que l'efficacité spectrale. Le principal changement entre OFDM et OFDMA est que les blocs de ressources appartenant à la même sous-trame qui peuvent être partagés entre plusieurs utilisateurs dans OFDMA ne sont affectés qu'à un seul utilisateur dans OFDM, comme illustré dans la figure 1.7.



(a) OFDMA (b) OFDM

Figure 1.7: OFDMA et OFDM

Puisque LTE peut utiliser une bande passante jusqu'à 20 MHz en largeur, le réseau LTE peut avoir au plus 1200 sous-porteuses utilisées pour la communication. Pour utiliser un tel flux de données en parallèle, un système de cette envergure nécessiterait 1200 oscillateurs pour ces sous-porteuses au niveau de l'émetteur et du récepteur, ce qui le rendrait très difficile à implémenter, mais grâce aux processeurs dédiés aux signaux numériques (DSP), le calcul rapide comme les opérations

de IFFT sur le signal du domaine fréquentiel et la transformée en échantillons dans le domaine temporel devient plus rapide et cela rend beaucoup plus facile la génération et la transmission du signal.

Le seul inconvénient avec OFDMA est que la puissance requise pour générer ces signaux est très élevée en raison du ratio énergétique PAPR (Peak to Average Power Ratio) (Myung et Goodman, 2008). L'énergie des UE est très limitée (principalement des téléphones cellulaires avec batterie), donc pour réduire le coût énergétique, les UE génèrent des signaux en mode *Uplink* utilisant la technique SC-FDMA, qui fait du mappage de symbole de constellation sur plusieurs sous-porteuses en utilisant seulement une fraction de l'intervalle du temps alloué, cela réduit considérablement l'énergie nécessaire pour envoyer un symbole. OFDMA par contre, associe ces données à une seule sous-porteuse, mais les diffuse sur un intervalle de temps plus large, ce qui rend le PAPR proportionnel au carré du nombre de sous-porteuses utilisées, donc utiliser deux techniques différentes pour envoyer le même signal est une conséquence de l'utilisation d'une architecture de trame en grille très large (Maximum de $1200 \times 15KHz$). La figure 1.8 présente la différence entre OFDMA et SC-FDMA pour envoyer des points en modulation QPSK.

TDD et FDD

La communication TD-LTE (ou TDD) utilise une seule fréquence, mais le temps de transmission et le temps de réception sont différents, ce qui lui permet d'imiter une communication duplex intégrale en utilisant un lien semi-duplex. TDD a l'avantage dans l'estimation des canaux pour la formation de faisceau (Beamforming) puisqu'ils sont séparés par un intervalle de garde. FDD-LTE comme décrit précédemment utilise des porteuses différentes, il permet une communication duplex pleine et robuste contre les interférences avec son efficacité spectrale. Ces

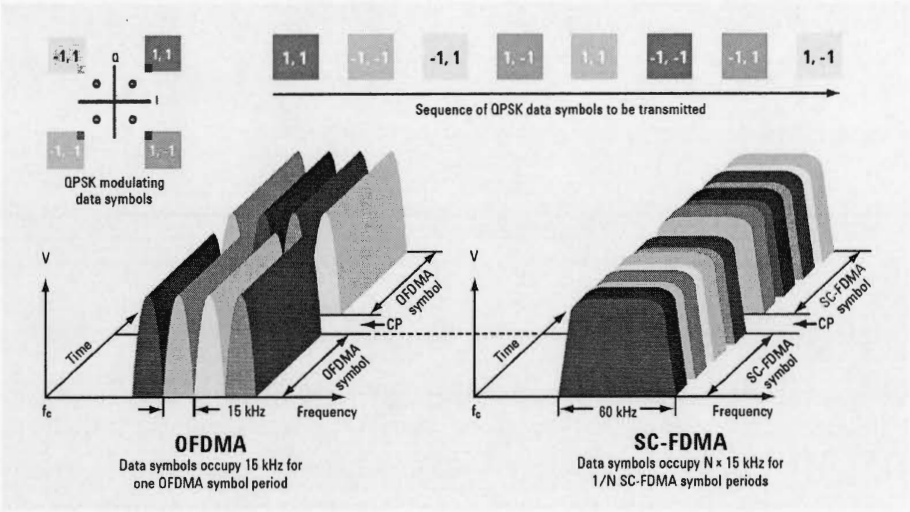


Figure 1.8: OFDMA et SC-FDMA (Rumney, 2008)

différences entre TDD et FDD sont uniquement présentes dans la couche physique, donc transparentes au niveau des couches supérieures, les figures 1.9 et 1.10 illustre la durée des deux types de trames.

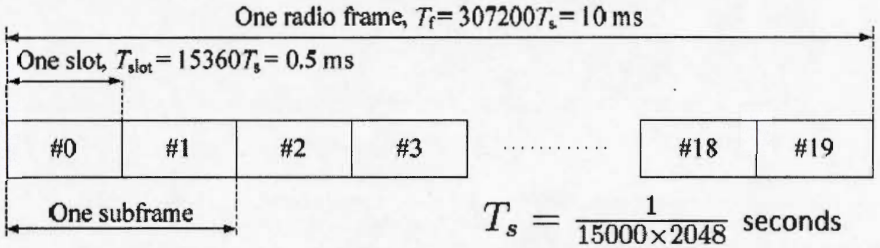


Figure 1.9: FDD LTE frame (TS-36.211, 2015)

Systèmes multi-antennes

L'approche MIMO (Multiple Input Multiple Output) est devenue très populaire au cours des dernières années, en particulier avec son utilisation dans le protocole WiFi (standard 802.11ad) et les réseaux LTE. Cette technique permet l'utilisation de plusieurs antennes pour la transmission ou la réception d'ondes radio, ce qui

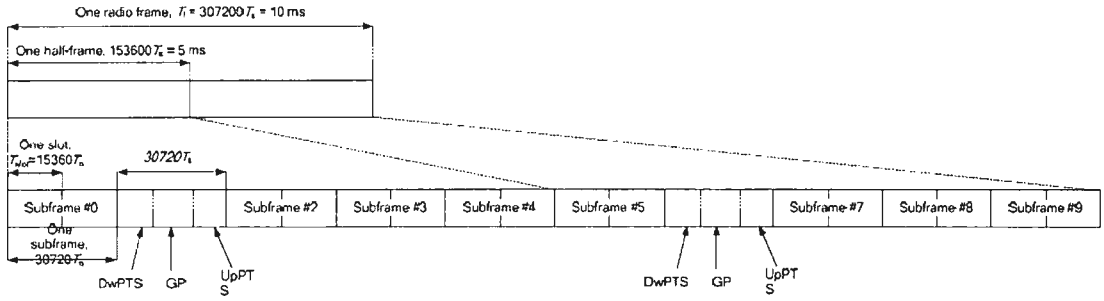


Figure 1.10: Trame TDD-LTE (TS-36.211, 2015)

améliore considérablement le débit et la qualité des liens en consommant la même énergie. Cette technique peut être utilisée sous différentes formes, comme envoyer plus de données (multiplexage spatial), ou les mêmes données plus efficacement (diversité de transmission). La norme LTE comporte des modes de multiplexage fixe qui peuvent être utilisés librement par les entreprises de communication dans le mode *Downlink*.

Pour les futures versions du protocole, les stations de base LTE et les UE vont pouvoir utiliser plus d'antennes pour transmettre et recevoir les données. MIMO peut améliorer la fiabilité de la transmission en envoyant la même copie du signal sur différentes antennes, ce qui améliorera sa résistance au bruit. Le récepteur peut combiner ces copies du signal, le seul inconvénient avec cette technique est que nous utilisons 2 ou 4 fois plus d'énergie pour envoyer la même grille de ressources, donc la complexité du traitement des trames à la réception augmente considérablement. D'un point de vue théorique, la figure 1.11 présente l'approche MIMO pour transmettre et recevoir des données sur plusieurs antennes.

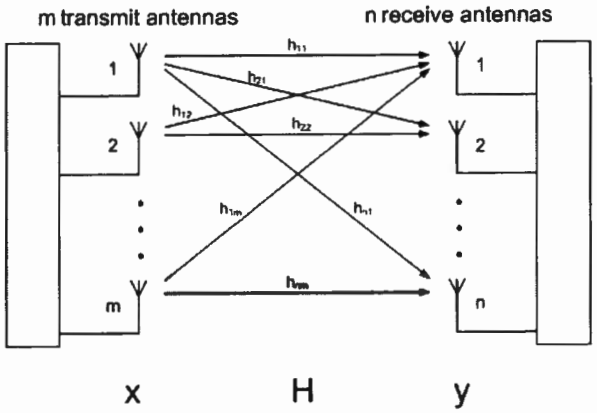


Figure 1.11: Système MIMO avec $m \cdot TX$ et $n \cdot RX$ antennes (Schulz, 2015)

1.3 Plateforme CUDA

1.3.1 Approche par GPU

Le traitement d'information utilisant les processeurs graphiques (GPGPU) est une technique qui permet l'envoi de certains calculs, généralement traités par le CPU, à une carte graphique (GPU) qui jouera le rôle de coprocesseur sur demande. Cette technique a été rendu célèbre à la fin des années 90 puisqu'elle permettait aux entreprises de faire du calcul parallèle complexe sur leur carte graphique. Le seul inconvénient à l'époque était la complexité de concevoir un tel système et de le réaliser. Les programmeurs devaient traduire un problème mathématique ou un algorithme complexe en polygones pour pouvoir exploiter ces ressources supplémentaires. Cette restriction est due au fait que les seuls librairies de programme et outils disponibles pour être utilisés sur les GPU avaient été conçus uniquement pour les opérations graphiques, ce qui a contraint les entreprises œuvrant dans le domaine financier par exemple d'embaucher des développeurs de jeux vidéos pour développer des systèmes capables d'effectuer un traitement parallèle pour l'analyse des données.

Pour résoudre ce problème, certaines chercheurs se sont penchés sur le problème pour transformer le GPU en périphérique de calcul parallèle, cela a donné naissance à un nouveau paradigme informatique : le traitement de flux (stream processing). Le premier environnement de développement exploitant ce paradigme sur les GPU était « Brook » (Buck *et al.*, 2004), l'ancêtre de CUDA. *Brook* avait une syntaxe simple (C étendu) et permettait d'effectuer toutes sortes d'opérations matricielles sur les cartes graphiques. Il permettait de développer des applications s'exécutant sur GPU sans connaissances préalables dans le domaine de l'infographie.

Lors des premiers tests, les performances des GPU étaient 2 à 5 fois supérieures comparés au CPU. En utilisant le concept de flux combiné avec le compilateur *Brook*, on pouvait transformer un GPU en CPU. Des programmeurs, sans connaissance préalable du calcul parallèle, ont pu l'utiliser avec différents types d'algorithmes parallélisable comme la DFT (Transformée de Fourier discrète) ou le traitement d'images complexes.

1.3.2 Architecture matérielle

La quantité des données à traiter a considérablement augmenté au cours des 20 dernières années, mais le nombre de cœurs des CPU est toujours bas, dû principalement à l'accroissement rapide du volume de stockage des ordinateurs de bureau et des serveurs Web. Le nombre de cœurs physiques dans un CPU peine à dépasser 24 cœurs, contrairement aux GPU qui ont atteint 2000 cœurs par puce juste au cours des dernières années, dû au jeu d'instruction reduit et specialisé des noyaux de GPU, ce qui permet d'en augmenter leur densité sur puces. De plus, la possibilité de travailler avec plusieurs cartes graphiques sur la même machine rend l'aternative GPU idéale pour opérer sur des données volumineuses.

Cette technologie peut être est simple à utiliser : dans un programme séquen-

tiel on prépare un flux de données et on l'envoie au GPU pour être traité en parallèle tandis que le reste de l'application continue son exécution sur le CPU. Du point de vue de l'utilisateur, les programmes basés sur des calculs mathématiques fonctionnent beaucoup plus rapidement. Une façon simple de différencier un GPU d'un CPU est de comparer leurs façons de traiter les tâches. Un CPU est composé de quelques noyaux complexes optimisés pour le traitement séquentiel tandis qu'un GPU possède une architecture massivement parallèle composée de milliers de noyaux simples et plus petits conçus pour exécuter plusieurs opérations simultanément, comme le montre la figure 1.12.

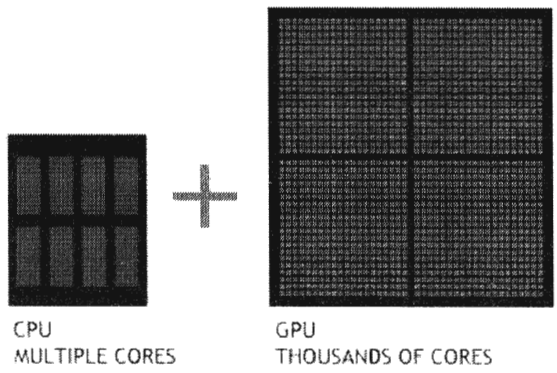


Figure 1.12: CPU et GPU (NVIDIA, 2007)

Puisque les CPU ont été conçus pour exécuter des systèmes d'exploitation et des programmes à l'aide d'un grand jeu d'instructions, une grande quantité de mémoire cache et de circuits de contrôle étaient nécessaires pour obtenir les meilleurs résultats pour des opérations SISD. Les GPU ont été créés pour exécuter un petit jeu d'instructions (opérations graphiques) sur de grandes quantités de données (primitives géométriques) avec une approche SIMD principalement pour assembler l'image finale à afficher sur l'écran, cette simplicité a fait en sorte que le circuit de contrôle et la cache étaient beaucoup plus petits, comme illustré dans la figure 1.13.

Figure 1.13: Architecture interne CPU et GPU (CudaDev, 2007)

Au moment où ce travail a été réalisé, l'architecture *Kepler* de NVIDIA était utilisée dans toutes les nouvelles cartes graphiques, conçues avec une architecture modulaire. Elle est basée sur des blocs appelés SMX (Streaming Multiprocessor) avec six contrôleurs de mémoire et une grande mémoire cache partagée par tous ces blocs, comme illustré dans la figure 1.14.

Figure 1.14: Architecture interne d'un GPU CUDA (Nvidia, 2007)

Les SMX sont la puissance de calcul principale du GPU, chacun d'eux contient

192 noyaux CUDA point flottant simple précision, 64 unités à double précision, 32 unités à fonction spéciale (SFU) (pour les opérations mathématiques) et 32 unités de lecture/écriture (pour l'accès à la mémoire globale). Ils y'a également une mémoire pour les textures, qui n'est pas utilisée avec CUDA, comme on peut le voir dans la figure 1.15. Chaque noyau est capable d'exécuter des instructions indépendamment des autres en utilisant directement la source d'horloge principale de la puce (fréquence GPU 900 MHz).

Figure 1.15: Architecture interne d'un module SMX (Nvidia, 2007)

1.3.3 Architecture logicielle

CUDA est basé sur le standard de l'industrie C/C ++ , étendu avec un ensemble d'extensions permettant la programmation ainsi que le contrôle GPU et sa mé-

moire volatile. Le partitionnement logiciel/matériel , comme le montre la figure 1.16, est organisé comme suit :

— Compute Engines : c'est l'implémentation matérielle de l'architecture CUDA, donc c'est un ensemble de processeurs, mémoires et contrôleurs qui exécutent les instructions de calcul.

— CUDA Driver and OS support : la bibliothèque permettant la communication avec le GPU et son BIOS (microprogramme contrôlant la carte) qui ne peut exécuter que les programmes binaires CUDA « cubin ». Ces derniers viennent sous la forme d'un code assembleur (PTX) pouvant être exécuté uniquement par le GPU.

— Application APIs : Une interface de programmation avec laquelle l'application utilise le pilote CUDA pour configurer le GPU, lancer le calcul et lire les résultats. Ces bibliothèques viennent sous la forme d'un code CUDA précompilé pour accélérer les opérations mathématiques matricielles et les algorithmes de traitement de signal et d'images.

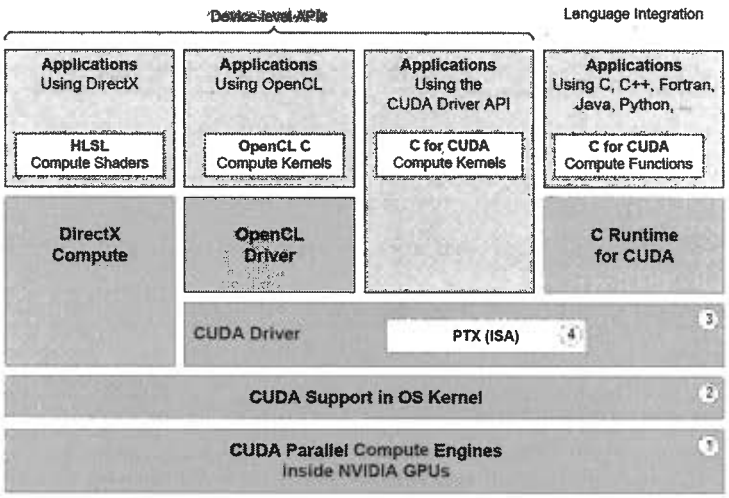


Figure 1.16: Environnement de développement (Arch, 2007)

1.3.4 Flux de traitement

Une fois qu'un code CUDA est passé par le processus de compilation, il est séparé en deux types d'instructions : les instructions CPU (Host code) et GPU (Device code). Les instructions CPU passent par le processus de compilation habituel comme tout autre programme C compilé avec un compilateur populaire comme GCC ou Visual C qui produit un code compatible avec le jeu d'instruction du processeur utilisé, les instructions du GPU par contre passe par le compilateur de NVIDA « nvcc » qui produit le code PTX, ce dernier peut être exécuté uniquement par un GPU compatible CUDA.

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Figure 1.17: Fonction CUDA (CudaDev, 2007)

Le code PTX est généré à partir des mots clés étendus dans le programme CUDA, par exemple le mot-clé « __global__ » indique qu'une fonction qui s'exécute sur le GPU est appelée à partir du code hôte, cette dernière est appelé « Kernel ». La figure 1.17, qui illustre un exemple d'appel à une fonction CUDA, les crochets triples indiquent un appel de fonction du code hôte vers un code GPU pour lancer un *Kernel* avec des paramètres spécifiques, comme la taille des données ciblée.

Comme on peut le constater, l'hôte et le périphérique sont traités comme des

entités distinctes, les pointeurs du périphérique pointent vers la mémoire GPU et les pointeurs de l'hôte pointent vers la mémoire CPU (la RAM gérée par le système d'exploitation). Pour utiliser la puissance du calcul, les données à traiter doivent être copiées dans la mémoire GPU avant d'être traitées puis le résultat recopié dans la mémoire RAM, ceci est dû au fait que le code du périphérique et le code de l'hôte ne peuvent pas communiquer directement.

La librairie de CUDA offre des fonctions d'opérations sur la mémoire, comme « cudaMalloc », « cudaFree », « cudaMemcpy » qui sont similaire aux fonctions standard C (Kernighan, 1988) « malloc », « free » et « memcpy », mais les opérations n'acceptent que les pointeurs de mémoire GPU.

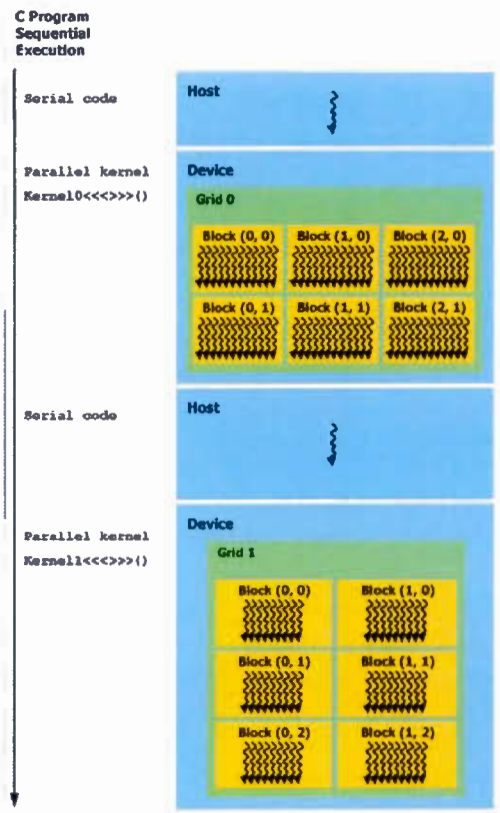


Figure 1.18: Execution d'un programme CUDA (CudaDev, 2007)

Les SMX de CUDA exécutent des *Threads* (une instance d'un *Kernel*) organisés en matrice, leur comportement est similaire aux processus fils du CPU. Ces *Threads* sont organisés en blocs (1024 *Threads* par bloc au maximum) et les blocs en grille (Grid). Tous les appels aux *Kernel* sont asynchrones, de sorte que le contrôle retourne au CPU immédiatement après un lancement de ces fonctions (lancer l'exécution du GPU). Cette approche permet à plusieurs processeurs ou processus de partager le même périphérique (le même GPU) et effectuer des appels de *Kernel* indépendamment. Ces exécutions de *Kernel* se font dans un contexte appelé « Stream » qui sépare logiquement deux programmes qui utilisent le même GPU par exemple. Dans les figures 1.18 et 1.19 on peut voir l'organisation des *Thread* en blocs et l'exécution d'un programme CUDA dans un contexte.

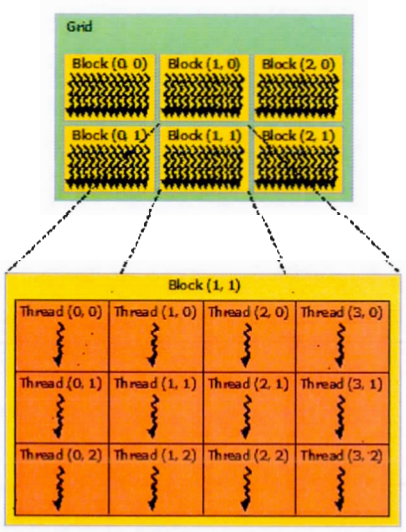


Figure 1.19: CUDA *Grid*, *Blocks* et *Threads* (CudaDev, 2007)

Les *Kernel* sont exécutés en groupe de 32 *Threads* parallèles appelés « Warp ». Chaque bloc de *Threads* doit être complété avant de traiter le bloc suivant, chaque bloc est exécuté par un seul SMX qui lui est assigné par les ordonnanceurs de la carte. Les SMX dispose de quatre des ces derniers pour organiser l'exécutions des *Warp* sur les SMX de la carte graphique ; ceci est illustré dans la figure 1.20.

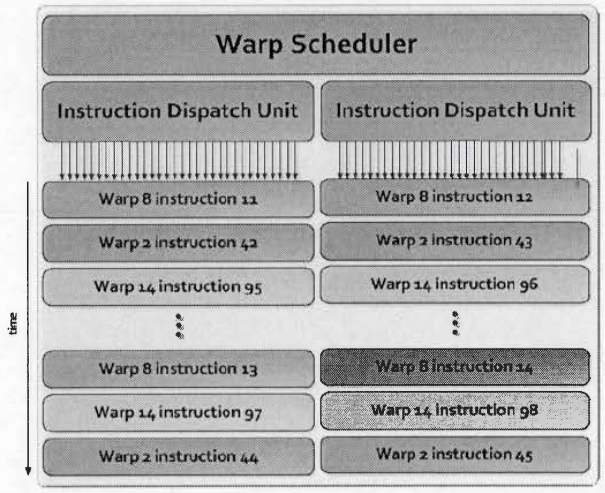


Figure 1.20: Ordonnanceur des *Warp* (Nvidia, 2007)

Le traitement des données par carte graphique peut être utilisé dans n'importe quel domaine scientifique tant que le calcul effectué est parallélisable. Dans les chapitres suivants, on montrera que le traitement des 140 symboles d'une trame LTE peut être parallélisé, ce qui permettra la virtualisation de la couche physique de ce protocole.

CHAPITRE II

TRAITEMENT DES TRAMES LTE

2.1 Le mode *Uplink*

La transmission *Uplink* (UE à eNodeB) utilise SC-FDMA, ceci a pour but de diminuer le ratio entre l'énergie maximale et l'énergie moyenne lors de la transmission d'une trame LTE. Ce ratio est appelé (PARP) et une bonne valeur de ce dernier permettrait une meilleure affectation de la bande passante. La différence principale entre les modes *Uplink* et *Downlink* est le mappage des symboles de modulation dans la grille des ressources RG. En plus, le mode *Uplink* ne supporte pas la diversité d'émission et le multiplexage spatial comme le mode *Downlink*.

En raison de l'architecture à faible consommation d'énergie, les UE n'ont qu'une antenne primaire pour transmettre et recevoir toutes les données échangées avec la station de base et une antenne secondaire appelé « Diversity Cellular Antenna », conçue pour faire de l'échantillonnage indépendant des signaux puis utiliser cette information pour choisir le bon signal (plus fort ou avec moins de bruit) ou pour combiner les données reçues sur les deux antennes. La prochaine génération du protocole LTE spécifiera plus d'antennes pour la transmission de données au niveau de l'UE.

Une trame envoyée par l'UE sur un canal aérien contient une combinaison de canaux et de signaux. Dans cette thèse, nous ne sommes intéressés que par le canal

PUSCH et son traitement par le GPU, car il contient toutes les données utilisateur transmises par les couches supérieures du protocole. La figure 2.1 montre la conversion des données binaires en entrée sous forme de « codeword » en signaux transmis par les antennes à la sortie.

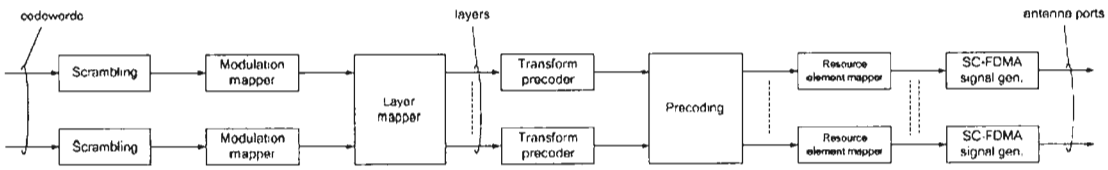


Figure 2.1: Transmetteur *Uplink* (TS-36.211, 2015)

Les modules principaux du mode *Uplink* en émission/réception appartiennent à deux blocs de calcul : un pour traiter les symboles et un autre pour le codage/décodage des canaux. Puisque le récepteur *Uplink* effectue l'inverse de toutes les opérations réalisées sur le côté émetteur avec l'émetteur et le récepteur ayant presque la même architecture au niveau de la couche physique, analysons davantage les opérations de l'émetteur.

Le mode *Uplink* au niveau de la couche physique est presque identique au mode *Downlink* à l'exception de certaines différences dans certains modules, la grille de ressource par exemple est identique, mais le mappage des RE est un peu différent en raison de l'utilisation de SC-FDMA, les modules affectés sont : le pré-codage, le mappage des RE dans la grille et le générateur du signal physique SC-FDMA.

Les données reçues au niveau de la couche MAC s'appellent « Transport Block » (TB). Une fois passées à la couche physique, ces données passent par plusieurs modules de traitement avant d'être envoyées par l'antenne. Dans les prochaines sections, on analysera en profondeur tous ces modules dont les spécifications sont décrites dans le standard TLE (TS-36.212, 2015).

2.2 Codage du canal partagé *ULSCH*

Le codage du canal partagé dans le mode *Uplink* et l'adaptation de débit sont identiques au mode *Downlink*, les étapes suivantes sont effectuées sur les données d'entrée (à partir de la couche MAC) avant de traiter la grille de ressources et envoyer la trame physique.

Ajout du CRC

Des bits de contrôle de redondance cyclique (CRC) (Miller *et al.*, 2009) sont calculés sur les blocs de données *TB* qui arrivent de la couche MAC à la couche physique et ceci afin d'être utilisé dans la détection d'erreur une fois arrivé au destinataire. Le calcul du CRC est une étape essentielle, mais si la longueur du *TB* dépasse une certaine valeur (6144 bits, y compris le CRC), le *TB* est segmenté en blocs appelés « Code Blocks »(CB), qui ont leurs propres CRC de 24 bits, comme on peut le voir dans la figure 2.2.

Le module CRC effectue une ou deux opérations :

TB-CRC (essentiel) : basé sur une génération cyclique polynomiale qui utilise le CRC24A (TS-36.212, 2015) pour calculer le résultat qui est ensuite concaténé au TB.

— CB-CRC (optionnel) : une fois que le TB a passé par une étape de segmentation, un autre calcul de CRC est effectué pour chaque CB généré, le générateur cyclique est appelé CRC24B (TS-36.212, 2015).

Encodeur turbo

Les CB traversent un codeur turbo (Turbo encoder), pour améliorer l'exploitation du canal en ajoutant des informations redondantes à l'aide d'un PCCC (Parallel

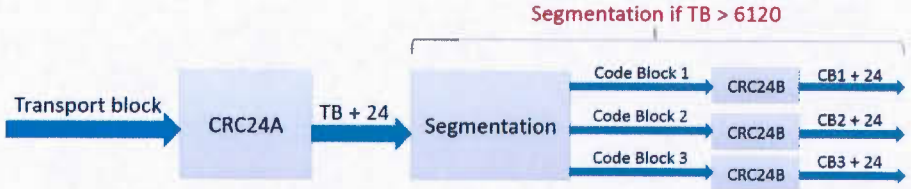


Figure 2.2: Concaténation du CRC

Concatenated Convolutional Code) comme on peut le voir dans la figures 2.3 et 2.4. Le PCCC est constitué de :

— Constituent encoders : deux codeurs convolutionnels récursifs (Recursive Convolutional code). On dit qu'ils sont récursifs parce que la sortie de l'encodeur est réutilisée par l'entrée et chacun donne comme résultat le bit x avec son bit de parité z.

— QPP (contention-free Quadratic interleaver) : ce module fait une permutation des bits à l'entrée du codeur turbo pour les copier à l'entrée du deuxième codeur convolutionnel (Chi et Kuo, 2012). Les valeurs de ces permutations de bits sont définies dans le standard (tableau 5.1.3-3 (TS-36.212, 2015)).

La sortie du codeur turbo de la figure 2.4 est sous la forme :

$$X_k, Z_k, X_{k+1}, Z_{k+1}, X_{k+2}, Z_{k+2}, X'_k, Z'_k, X'_{k+1}, Z_{k+1}, X'_{k+2}, Z'_{k+2}. \qquad (2.1)$$

Figure 2.3: Codeur Turbo

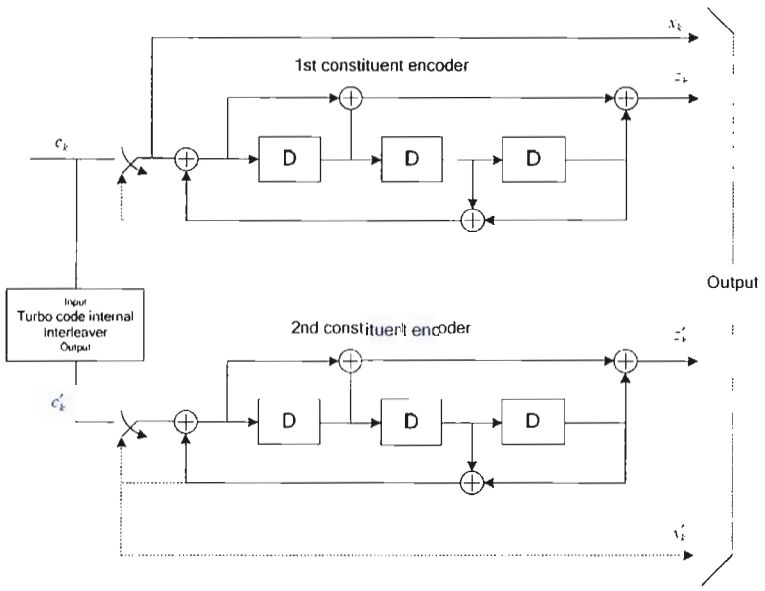


Figure 2.4: 1/3 Codeur turbo (TS-36.212, 2015)

Adaptation de débit

Ce module crée un débit de code souhaité, les sorties du turbo-codeur sont traitées en trois étapes consécutives, comme on peut le voir dans la figure 2.5 :

— Entrelaceur de sous-blocs (Sub-Block interleaver) : ce sont des modules identiques qui transforment le flux de données à l'entrée en une matrice de 32 colonnes, effectuent des permutations basées sur des modèles fixes puis chacun délivre un flux de bits entrelacés. L'objectif principal de l'entrelaceur est de réduire l'impact des erreurs rafale sur un signal (Cheng *et al.*, 2008). Les sorties x_k, z_k et z'_k du codeur turbo de la figure 2.4 correspondent aux entrées $d_k^{(0)}$, $d_k^{(1)}$ et $d_k^{(2)}$ de ces sous-blocs dans la figure 2.5.

— Collecte de bits (Bit Collection) : Cette étape collecte les bits et crée un tampon circulaire virtuel de longueur $K_w = 3K_\pi$, avec K_π tel que défini

par le standard (TS-36.211, 2015) représente la longueur des bits de chaque flux à l'entrée du module. En combinant ces trois flux, les bits entrelacés à la sortie sont indexés comme suit :

$$\left\{\begin{array}{ll} w_{[k]} = v^{(0)}_{[k]} & \text{pour } k = [0 \ldots K_\pi - 1] \\ w_{[K_\pi + 2k]} = v^{(1)}_{[k]} & \text{pour } k = [0 \ldots K_\pi - 1] \\ w_{[K_\pi + 2k + 1]} = v^{(2)}_{[k]} & \text{pour } k = [0 \ldots K_\pi - 1] \end{array}\right\}$$

La figure 2.6 nous montre comment les données de sortie de ce module sont organisées.

— Sélection de bits et groupage (Bit selection and pruning) : les bits e_k à la sortie de ce module sont sélectionnés et groupés à partir du tampon circulaire w_k pour créer une sortie dont la longueur correspond au débit du code désiré.

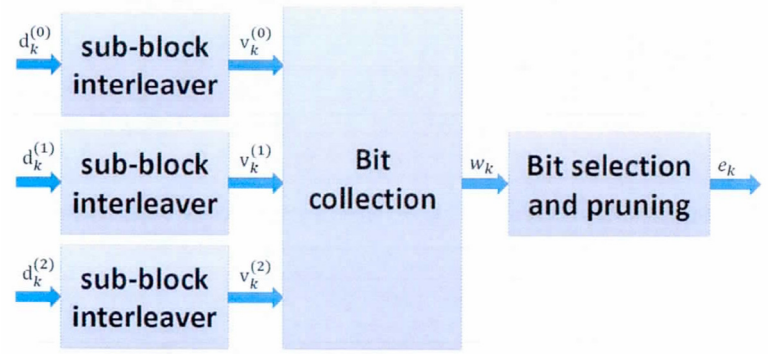


Figure 2.5: Modules du l'adaptateur de débit

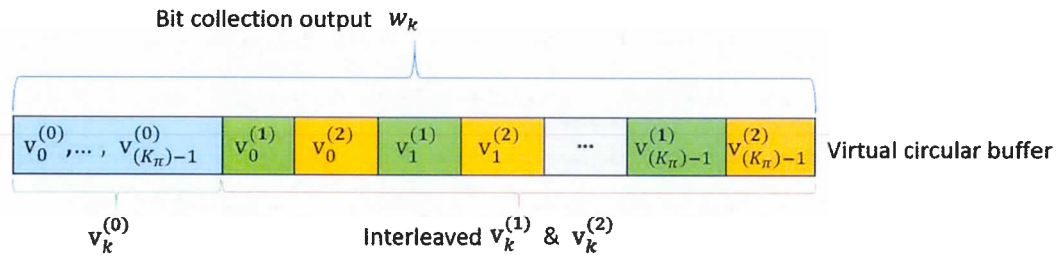


Figure 2.6: Mémoire à tampon circulaire

Concaténation des CB

C'est le dernier bloc du codage de canal des données de l'utilisateur, les multiples lignes sont combinées pour créer un flux de bits long qui crée le bloc de transport codé.

Multiplexage des données

Cette étape ajoute l'indicateur de qualité de canal (CQI) au bloc de transport codé qui est mappé aux RE qui se trouvent autour des signaux pilotes utilisés pour la démodulation (DRS). Le CQI a une valeur entre [0..30] avec 30 indiquant la meilleure qualité de canal. Le eNobeB utilise cette information pour contrôler la taille du bloc de transport : plus la qualité est élevée plus le TB augmente en taille aussi. La proximité du CQI du DRS permet à l'estimateur du canal de l'eNodeB de faire une meilleure approximation de sa valeur (Les RE proches du DRS sont mieux estimés).

Entrelaceur de canal

Cette étape mappe l'indicateur de rang (RI) dans le RG, qui donne une estimation de l'interférence (ou bruit), en plus de l'ACK / NACK qui donne une indication si le décodage des données PDSCH a réussi ou non (réponse renvoyée à l'eNodeB), le tout dans une seule sous-trame(SF).

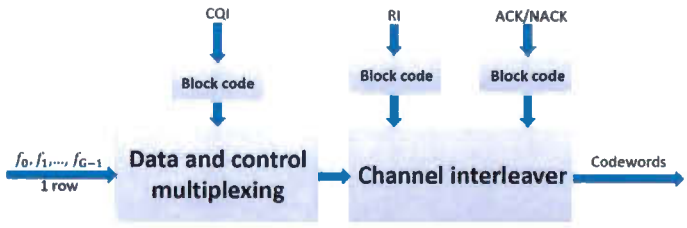


Figure 2.7: Canal de contrôle *PUCCH*

2.3 Traitement du canal partagé PUSCH

Brouilleur

Appelé « Scrambler » par le standard, il brouille les bits d'entrée codés afin de randomiser les données, de sorte que chaque *Codeword* q (deux codewords par sous-trame) est brouillé avec une séquence spécifique de l'UE (TS-36.211, 2015). Par exemple, les bits codés b sont brouillés en $\tilde{b}$ suivant la formule 2.2.

$$\tilde{b}^{(q)}(i) = (b^{(q)}(i) + c^{(q)}(i))mod2 \tag{2.2}$$

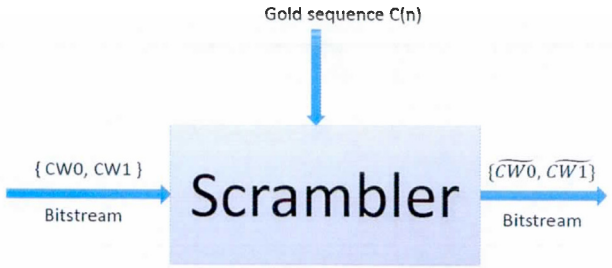


Figure 2.8: Module de brouillage

$C(n)$ de la figure 2.8 représente une séquence pseudo-aléatoire définie par le standard (TS-36.211, 2015).

Modulation QAM

Ce module sert à convertir le flux de bits à l'entrée en symboles à valeur complexe I/Q suivant le nombre de points de la constellation fixé par la station de base. Selon la qualité du canal, QPSK, 16QAM ou 64QAM peuvent être utilisés, de sorte que pour chaque *Codeword* q, le nombre de bits convertis en un point est de 2 (QPSK), 4 (16QAM) ou 6 (64QAM), donnant comme résultat un flux de valeurs complexes. Le mappage exact de la modulation QPSK est donné dans le

tableau 2.1.

Tableau 2.1: Mappage pour la modulation QPSK

b(i), b(i+1)	I	Q
00	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$
01	$\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$
10	$-\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$
11	$-\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$

Tableau 2.2: Scalaires des modulation LTE (TS-36.211, 2015)

Modulation	K_{norm}
QPSK	$\frac{1}{\sqrt{2}}$
16-QAM	$\frac{1}{\sqrt{10}}$
64-QAM	$\frac{1}{\sqrt{42}}$

Figure 2.9: Mappage binaire à complexe (TS-36.211, 2015)

Mappage sur les couches de transmissions

Appelé « Layer mapping », c'est le module qui s'occupe du mappage des symboles de modulation à valeur complexe dans une ou plusieurs matrices pour qu'elles soient envoyées par les antennes de transmission (codeword-to-layer mapping), mais comme en pratique la plupart des UE utilisent une seule antenne pour la transmission, l'entrée est égale à la sortie dans cette étape.

Le standard LTE a fixé la façon dont les constellations sont mappées sur les couches afin de faire du multiplexage spatial tel qu'illustré dans le tableau 2.10 qui est tiré du document de spécification technique de LTE. Par exemple dans la figure 2.11 on trouve 2 *Codeword* mappés sur 4 couches de transmission.

Figure 2.10: Mappage des *Codeword* sur la grille (TS-36.211, 2015).

Number of layers	Number of codewords	Codeword-to-layer mapping $i = 0,1,\ldots, M_{symb}^{layer} - 1$
1	1	$x^{(0)}(i) = d^{(0)}(i)$ $\quad M_{symb}^{layer} = M_{symb}^{(0)}$
2	1	$x^{(0)}(i) = d^{(0)}(2i)$ $x^{(1)}(i) = d^{(0)}(2i+1)$ $\quad M_{symb}^{layer} = M_{symb}^{(0)}/2$
2	2	$x^{(0)}(i) = d^{(0)}(i)$ $x^{(1)}(i) = d^{(1)}(i)$ $\quad M_{symb}^{layer} = M_{symb}^{(0)} = M_{symb}^{(1)}$
3	2	$x^{(0)}(i) = d^{(0)}(i)$ $x^{(1)}(i) = d^{(1)}(2i)$ $x^{(2)}(i) = d^{(1)}(2i+1)$ $\quad M_{symb}^{layer} = M_{symb}^{(0)} = M_{symb}^{(1)}/2$
4	2	$x^{(0)}(i) = d^{(0)}(2i)$ $x^{(1)}(i) = d^{(0)}(2i+1)$ $x^{(2)}(i) = d^{(1)}(2i)$ $x^{(3)}(i) = d^{(1)}(2i+1)$ $\quad M_{symb}^{layer} = M_{symb}^{(0)}/2 = M_{symb}^{(1)}/2$

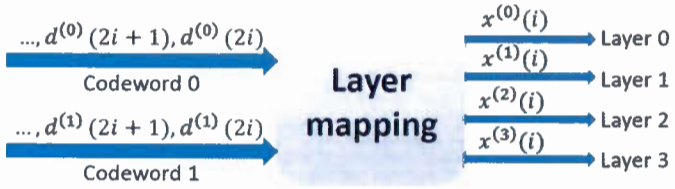


Figure 2.11: 2 *CW* mappés sur 4 couches

Précodage par transformée

Le précodage par transformée correspond à faire une opération DFT (ou FFT). La taille de cette dernière est égale au nombre de sous-porteuses utilisées pour la transmission. La taille de la DFT M_{sc}^{PUSCH} (nombre de Sc dans le PUSCH) doit respecter la Forme $2^i \times 3^j \times 5^k \times 7^l$ fixée par la norme LTE.

Chaque utilisateur dispose de certains groupes de blocs de ressources dédiés à son UE appelé *RBG*. Par exemple si le terminal utilisateur dispose de 5 *RBG* pour transmettre ses données dans la bande 1,4 MHz, alors $M_{sc}^{PUSCH} = 5 \times 12 = 60$ sous-porteuses. L'utilisateur ne peut utiliser que les SC qui lui sont attribuées par la station de base.

Tableau 2.3: Taille des RBG

Bande passante(MHz)	**Taille RBG**
1.4	1
3	2
5	2
10	3
15	4
20	4

L'eNodeB informe l'UE dans le PDCCH (champ DCI) sur le nombre de RBG attribué à l'utilisateur dans une représentation de type *Bitmap*. Dans le cas illustré dans la figure 2.12 utilisant la bande 20 MHz, 25 RBG sont disponibles pour être affectés aux utilisateurs, un UE peut alors utilisé les *RBG* dont l'index correspond à 1 dans la matrice d'allocation dans le DCI, comme on peut le voir dans la figure 2.12.

Le signal dans le domaine de fréquence résultant de cette assignation suit la formule tirée du standard (voir section 5.3.3 (TS-36.211, 2015)) suivante :

$$z(l \cdot M_{sc}^{PUSCH} + k) = \frac{1}{\sqrt{M_{sc}^{PUSCH}}} \sum_{i=0}^{M_{sc}^{PUSCH}-1} d(l \cdot M_{sc}^{PUSCH} + i) e^{-j \frac{2\pi ik}{M_{sc}^{PUSCH}}} \quad (2.3)$$

avec $k = [0 \dots M_{sc}^{PUSCH} - 1]$ et $l = [0 \dots \frac{M_{symb}}{M_{sc}^{PUSCH}-1}]$.

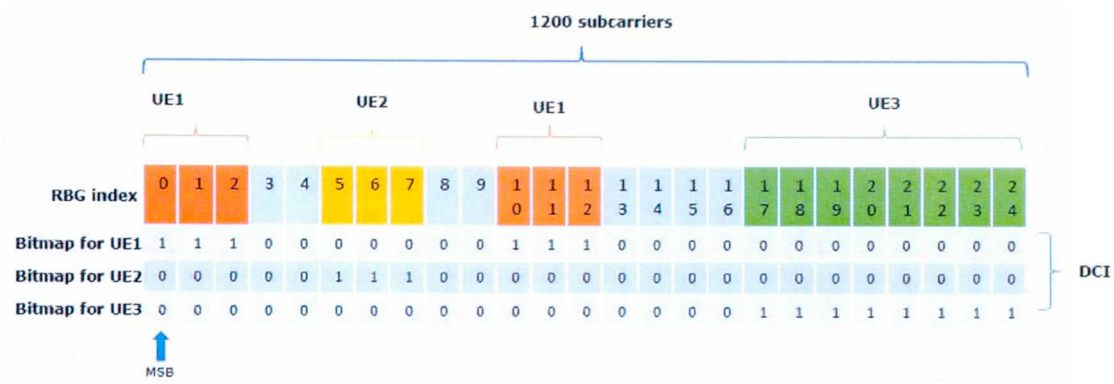


Figure 2.12: Allocation des RBG

z est le numéro de la matrice de données à transmettre, d le numéro du bloc actuel, k le nombre de sous-porteuses utilisées et l le nombre de symboles utilisés.

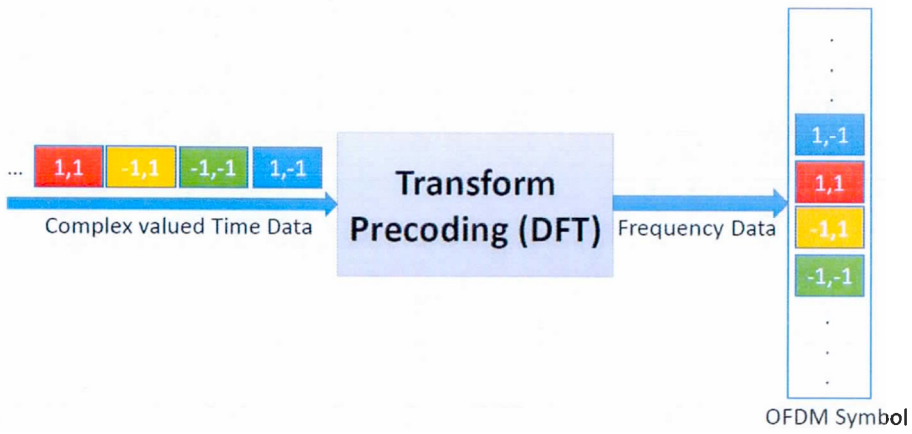


Figure 2.13: Précodage par transformée (DFT)

Précodage

Le précodeur mappe les blocs de données en entrée dans la grille de ressource pour chaque port d'antenne. Même si en pratique l'UE utilise une seule antenne pour transmettre, la norme LTE a spécifié le mappage des RE sur plusieurs antennes si le multiplexage spatial est utilisé et cela suivant les formules suivantes :

Transmission par antenne unique :

$$z^{(0)}(i) = y^{(0)}(i). \tag{2.4}$$

Transmission par plusieurs antennes :

$$[z^{(0)}(i) \dots z^{(P-1)}(i)] = W[y^{(0)}(i) \dots y^{(y-1)}(i)], \tag{2.5}$$

Avec $i = 0, 1, ..., M^{ap}_{symb} - 1$ et $M^{ap}_{symb} = M^{layer}_{symb}$.

La matrice utilisé pour le précodage W est appelé « Codebook », elle contient des valeurs fixes définies par le standard LTE (voir section 5.3.3A.2 du (TS-36.211, 2015)).

Mappage des ressources

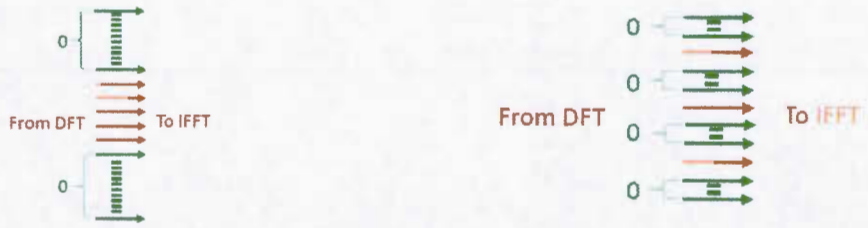
Cette étape mappe les symboles de valeur complexe précodés aux RE dans la grille actuellement assignés à l'utilisateur en partant du RE se trouvant au coin supérieur gauche d'un RB et en finissant par le RE inférieur droit utilisant un pas de 2, 3 ou 5 RE. Les données peuvent être mappées n'importe où sur la grille de ressources tant que ces RE ne sont pas affectés à un signal de référence (un signal pilote) ou à des données du canal de contrôle.

A cette fin, deux types de techniques sont utilisés en pratique : localisés et distribués.

— La technique localisée mappe les RE sur des sous-porteuses consécutives sur une fraction de la bande passante et attribue des zéros à tous les autres RE afin que nous puissions obtenir une meilleure efficacité en fréquence quand plusieurs utilisateurs sont connectés.

— La technique distribuée mappe les éléments de ressource sur toute la bande passante allouée sans données consécutives dans une approche entrelacée

afin que nous puissions obtenir un UE avec une meilleure transmission pour le canal de contrôle.



(a) Mappage localisé (b) Mappage distribué

Figure 2.14: Mappage localisé et distribué

Générateur de signal

Ce module génère un signal physique SC-FDMA dans le domaine temporel à valeur complexe pour chaque port d'antenne en effectuant la transformée de Fourier inverse (IFFT) avec un décalage d'une demi sous-porteuse $(k+\frac{1}{2})$ sur les données en entrée (TS-36.211, 2015) et enfin ajouter le préfixe cyclique à la trame avant de l'envoyer sur le canal physique via un convertisseur numérique-analogique (CNA). Le signal $s_l^{(p)}(t)$ envoyé sur le port de l'antenne p pour un symbole spécifique l est défini par :

$$s_l^{(p)}(t) = \sum_{k=-[\frac{N_{RB}^{UL} N_{sc}^{RB}}{2}]}^{[\frac{N_{RB}^{UL} N_{sc}^{RB}}{2}]-1} a_{k^*,l}^{(p)} \cdot e^{j2\pi(k+\frac{1}{2})\triangle f(t-N_{CP,l}T_s)} \tag{2.6}$$

Pour $0 \leq t < (N_{CP}+N)\times T_s$ avec $k^* = k+[N_{RB}^{UL}N_{sc}^{RB}/2]$, $N = 2048$, $\triangle f = 15$ kHz, $a_{k^*,l}^{(p)}$ est le contenu du RE (k,l) sur le port d'antenne p et la longueur du préfixe cyclique $N_{CP,l}$ pour le symbole en cours l et T_s est l'unité de temps minimale fixe définie dans la figure 1.9.

Comme on a pu le constater, le standard a défini les opérations à appliquer sur le flux des données pour la transmission et la réception du signal, on remarque que chaque module a un seul port d'entrée et de sortie, ce qui nous donne un calcul séquentiel avec une complexité différente d'un module à l'autre. Dans le chapitre suivant, on analysera l'aspect computationnel des modules les plus complexes en matière d'opérations nécessaires pour donner le bon résultat à la sortie ainsi que les endroits où une optimisation par parallélisation du calcul est possible.

CHAPITRE III

ANALYSE COMPUTATIONNELLE ET PARALLÉLISATION

3.1 Flux de données à la réception

Le récepteur *Uplink* effectue l'opération inverse de tous les modules de l'envoyeur. La seule information inconnue que le destinataire doit estimer est le canal physique sur lequel la trame a été envoyée.

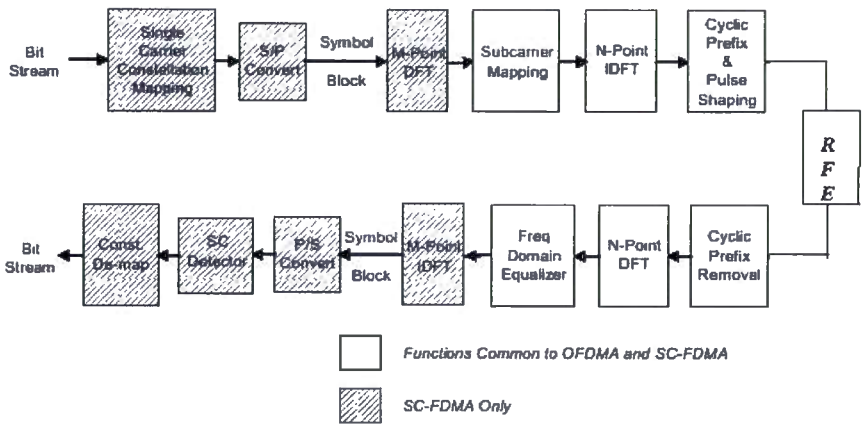


Figure 3.1: Émetteur/Récepteur LTE (Zyren, 2007)

La procédure de traitement au niveau de la station de base LTE pour traiter la trame reçue est la suivante :

— **CAN :** l'eNodeB reçoit le signal radio sur une ou plusieurs antennes et le convertit en flux binaire.

— **Suppression du CP :** pour tous les 140 symboles reçus, ce module supprime ou ignore les échantillons qui font partie du préfixe cyclique.

— **FFT :** on effectue une opération FFT pour chaque symbole reçu pour le convertir en un signal dans le domaine fréquentiel.

— **Estimation du canal :** on estime la matrice qui correspond au canal physique sur lequel le signal a été envoyé en utilisant les symboles des pilotes (DRS).

— **Estimation du bruit :** une fois que la matrice du canal a été estimée, le bruit peut être déduit à l'aide des RE aux positions pilotes appartenant à une même sous-porteuse.

— **Égalisation du canal :** on utilise la matrice de canal estimée H et la valeur de bruit estimée n pour déduire la matrice des données envoyée par les utilisateurs.

— **IFFT :** on effectue une opération IFFT sur les RBG appartenant à chaque utilisateur groupé dans les symboles, ce qui donne en résultat plusieurs signaux dans le domaine temporel, un par utilisateur.

— **Démodulation QAM :** selon le mode de modulation choisi par l'eNodeB, ce module convertit les RE à valeurs complexes de chaque utilisateur en un flux de bits (2, 4 ou 6 bits par RE).

Décodage de canal : dans cette phase, l'opération inverse de tous les modules de codage de canal est effectuée, ce qui implique le dé-brouillage, le dé-entrelacement, le dé-multiplexage, l'extraction des *Codeword*, l'adaptation de débit, le décodage turbo, la validation et la suppression des CRC des *Codeword* et des TB.

3.2 Parallélisme dans la couche physique

Étant donné que toutes les données reçues par la station de base sont organisées sous forme matricielle, la plupart des modules de traitement peuvent être exécutés en parallèle. Les sous-trames LTE disposent de toutes les informations nécessaires pour traiter les données des utilisateurs, car au moins deux RB dans le domaine temporel sont alloués pour chaque utilisateur, de sorte qu'une station de base peut traiter indépendamment toutes les sous-trames reçues, mais le problème avec cette approche est que la valeur instantanée du bruit ne peut pas être estimée avec une seule sous-trame et une meilleure estimation serait plus efficace si nous disposons de toute la matrice estimée H. C'est pourquoi il serait plus efficace de commencer le traitement une fois que toute la trame a été capturée.

Le signal reçu peut être traité en parallèle de plusieurs façons :

Niveau antenne

Les antennes sont physiquement liées aux convertisseurs analogiques et numériques (CNA), de sorte que chaque antenne capture une trame indépendante qui doit être traitée. Si cette trame ne dépend pas des autres (MIMO multi-utilisateur), elle peut être traitée jusqu'à ce qu'elle atteigne la couche MAC, sinon la trame peut être traitée indépendamment jusqu'à ce qu'elle atteigne le décodeur par transformée.

Niveau Symbole

À ce niveau, l'entrée de données se distingue par les symboles séquentiels reçus, de sorte que des opérations telles que FFT et IFFT peuvent être effectuées indépendamment sur chaque symbole. Les RE au niveau des pilotes dans les trame *Uplink* sont envoyés dans le symbole numéro 4 et 11 de chaque sous-trame, de sorte que

l'estimation du canal et l'estimation du bruit dans les positions des pilotes peuvent être parallélisées au niveau symbole également.

Niveau sous-porteuses

Les données des utilisateurs sont organisées en paires (symbole/sous-porteuse), de sorte que l'extraction des données indépendantes des utilisateurs peut être effectuée en parallèle. Certains algorithmes dans l'estimation du canal sont effectués sur les RE appartenant à la même sous-porteuse, comme l'interpolation linéaire ou autres formes d'interpolation ainsi que l'estimation du bruit.

Niveau algorithme

OFDM et SC-FDMA sont utilisés dans de nombreux protocoles de communication et grâce à cela, de nombreuses recherches ont été faites pour améliorer la fiabilité, le débit et l'efficacité de la communication. A cette fin, la plupart des algorithmes développés ont des calculs indépendants sur les données qui peuvent être parallélisées, en particulier dans les modules d'estimation et d'égalisation du canal.

Niveau utilisateur

Étant donné que toutes les données des utilisateurs dans le *PUSCH* sont indépendantes les unes des autres, l'extraction des données de l'UE, IFFT et le décodage des canaux peuvent être parallélisés.

3.3 Complexité computationnelle

Chaque module de traitement à la réception de l'eNoveB a une complexité de calcul différente des autres, mais puisque toutes les opérations sont sur des données à valeurs entières ou réelles, analysons la complexité de chaque module.

Suppression du préfixe cyclique

Avant de commencer la suppression du préfixe cyclique, une synchronisation dans le temps peut être effectuée par eNodeB pour s'aligner sur la bonne position de la trame reçue, lorsqu'un UE se connecte à une station de base, il utilise les signaux PSS et SSS envoyés par l'enodeB pour se synchroniser avec elle. Dans certains cas, par exemple si aucune donnée n'est reçue par eNodeB pendant une certaine durée de temps, la station de base peut utiliser le signal SRS envoyé par l'UE, qui est présent dans le dernier symbole d'une sous-trames pour se synchroniser avec l'UE, cela pourrait arriver très souvent avec des utilisateurs à mobilité rapide. Finalement, enlever le préfixe cyclique après la synchronisation optionnelle est une simple opération qui consiste à copier tous les échantillons utiles et ignorer les CP.

FFT/IFFT

La FFT est un algorithme déduit de la DFT pour transformer un signal du domaine temporel vers le domaine fréquentiel qui réduit le nombre d'opérations de $O(N^2)$ à $O(NlogN)$.

La sortie d'une opération DFT sur un signal de taille N est :

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{\frac{(-j2\pi nk)}{N}}, \tag{3.1}$$

Avec k = 0, 1, 2, ..., N-1

L'opération FFT est la fonction la plus complexe du côté récepteur. Pou une taille N de symbole qui est une puissance de 2, elle peut être calculée avec l'algorithme « Cooley–Tukey radix-2 FFT » (Ibrahim *et al.*, 2016). Comme illustré dans le

tableau 3.1, tous les symboles sont compatibles avec Base-2, à l'exception des symboles de longueur 1536 points. L'algorithme DIT (in-place decimation-in-time) doit être alors utilisé puisque 1536 n'est pas une puissance de 2 (Cho *et al.*, 2013) qui calcule la FFT dans sa mémoire d'origine. Le tableau 3.1 illustre la complexité d'un base-r pour une FFT à n échantillons.

Tableau 3.1: Complexité d'une FFT base-r n-échantillons (Blahut, 1985)

	Multiplications réelles			*Additions réelles*		
Taile du bloc	**Base-2**	**Base-4**	**Base-8**	**Base-2**	**Base-4**	**Base-8**
128	712	-	-	2 054	-	-
256	1 800	1 392	-	5 896	5 488	-
512	4 360	-	3 204	13 566	-	12 420
1 024	10 248	7 856	-	30 728	28 336	-
2 048	45 056	-	-	67 584	-	-

L'algorithme Radix-2 divise un signal à N-échantillons en deux signaux à $N/2$ échantillons (signaux d'indice paire et impair) suivi de plusieurs DFT de taille 2 réduite. Cet algorithme correspond le mieux à presque toute les configurations de la bande passante (le nombre d'échantillons par symbole sont des multiples de 2 sauf dans la bande 15 MHz). La figure 3.2 illustre un exemple d'une opération FFT Base-2 à 8 échantillons.

Les RBG des utilisateurs extraites du PUSCH traversent un module IFFT dont la taille des blocs pour une operations IFFT en nombre de sous-porteuses par symbole respecte la formule $2^i \times 3^j \times 5^k \times 7^l$ sous-porteuses (avec i, j, k et l des entiers non négatifs). Une taille de blocs fixes pour effectuer plusieurs opérations en parallèle peut être pré-calculée, la taille de ces blocs (M_{sc}^{PUSCH}) est indiqué dans le tableau 3.2 (100 RB est le maximum alloué par le standard).

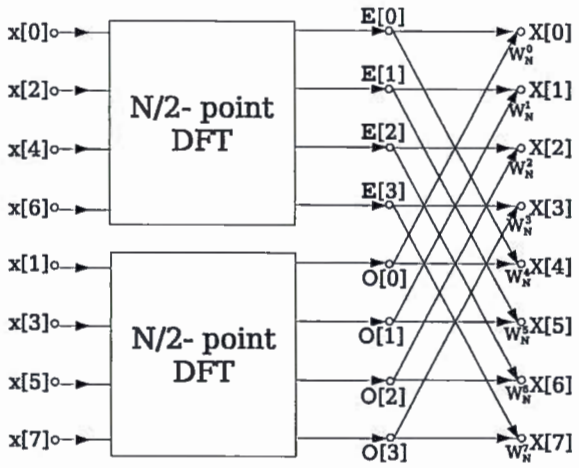


Figure 3.2: Radix-2 FFT à 8 échantillons (Ibrahim *et al.*, 2016)

$$M_{sc}^{PUSCH} = N_{sc}^{RB} \times 2^i \times 3^j \times 5^k \times 7^l \leq N_{sc}^{RB} \times N_{RB}^{UL} \tag{3.2}$$

Tableau 3.2: Taille des blocs disponibles pour IFFT

RB	*1*	*2*	*3*	*4*	*5*	*6*	*8*	*9*	*10*
M_{sc}^{PUSCH}	12	24	36	48	60	72	96	108	120
RB	*12*	*15*	*16*	*18*	*20*	*24*	*25*	*27*	*30*
M_{sc}^{PUSCH}	144	180	192	216	240	288	300	324	360
RB	*32*	*36*	*40*	*45*	*48*	*50*	*54*	*60*	*64*
M_{sc}^{PUSCH}	384	432	480	540	576	600	648	720	768
RB	*72*	*75*	*80*	*81*	*90*	*96*	*100*	-	-
M_{sc}^{PUSCH}	864	900	960	972	1080	1152	1200	-	-

Estimation et égalisation de canal

Ce module est consideré le plus important dans le traitement des trames LTE, l'objectif principal est de calculer le canal H sur lequel le signal a été envoyé.

Pour rendre l'estimation plus facile par le récepteur, un signal de référence connu par l'émetteur et le récepteur appelé « Demodulation Reference Signal » (DRS) est inséré dans chaque RB du PUSCH par l'UE juste avant de générer le signal physique, le DRS prend presque un symbole complet dans le domaine de fréquence, comme illustré dans la figure 3.3.

La génération du signal de référence $r_{u,v}^{(\alpha)}$ de longueur n qui est une séquence générée à partir de différentes valeurs de α et d'une séquence de base $\bar{r}_{u,v}(n)$ est la suivante (voir section 5.5.1 (TS-36.211, 2015) pour les paramètres) :

$$r_{u,v}^{(\alpha)} = e^{j\alpha n} \cdot \bar{r}_{u,v}(n), 0 \leq n < M_{sc}^{RS} \tag{3.3}$$

Avec $M_{sc}^{RS} = mN_{sc}^{RB}$ la longueur du signal de référence, $u = [0..29]$ qui représente l'index du RB et v le numéro de la séquence de base à l'intérieur du groupe.

Si $M_{sc}^{RS} \geq 3 \cdot N_{sc}^{RB}$ alors :

$$\bar{r}_{u,v}(n) = x_q(n \cdot modulo(N_{ZC}^{RS})), 0 \leq n < M_{sc}^{RS} \tag{3.4}$$

Avec la racine d'une séquence « Zadoff-Chu » de longueur $1 \leq m \leq N_{RB}^{PUSCH}$ définie par :

$$x_q(m) = e^{-j\frac{\pi qm(m+1)}{N_{ZC}^{RS}}}, 0 \leq m \leq N_{ZC}^{RS} - 1 \tag{3.5}$$

et q défini par :

$$q = [\bar{q} + \frac{1}{2}] + v \cdot \ (-1)^{[2\bar{q}]} \tag{3.6}$$

$$\bar{q} = N_{ZC}^{RS} \cdot (u+1)/31 \tag{3.7}$$

— Si $M_{sc}^{RS} < 3 \cdot N_{sc}^{RB}$ alors :

$$\bar{r}_{u,v}(n) = e^{\frac{\phi(n)\pi}{4}}, 0 \leq n < M_{sc}^{RS} - 1 \tag{3.8}$$

Les valeurs de ϕ, u et v sont fixées par le standard (section 5.5.1(TS-36.211, 2015)).

La longueur de la séquence « Zadoff-Chu » est égale au plus grand nombre premier inférieur à M_{sc}^{RS}.

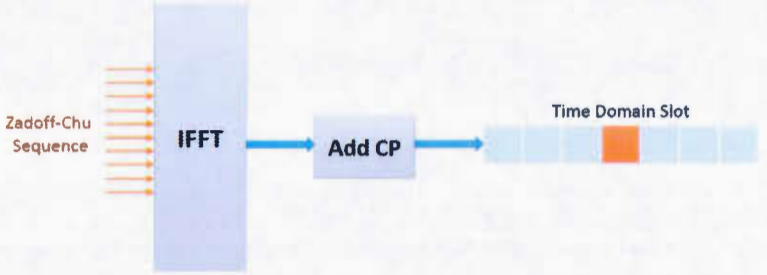


Figure 3.3: Insertion du signal DRS par l'UE

Cette technique est appelée « Pilot Assisted Channel Estimation », car une fois le signal est transformé dans le domaine fréquentiel via l'opération FFT, le récepteur peut extraire un ensemble de RE qui appartiennent au signal pilote (4ème symbole de chaque RB dans le PUSCH), comme nous pouvons le voir dans la figure 3.4.

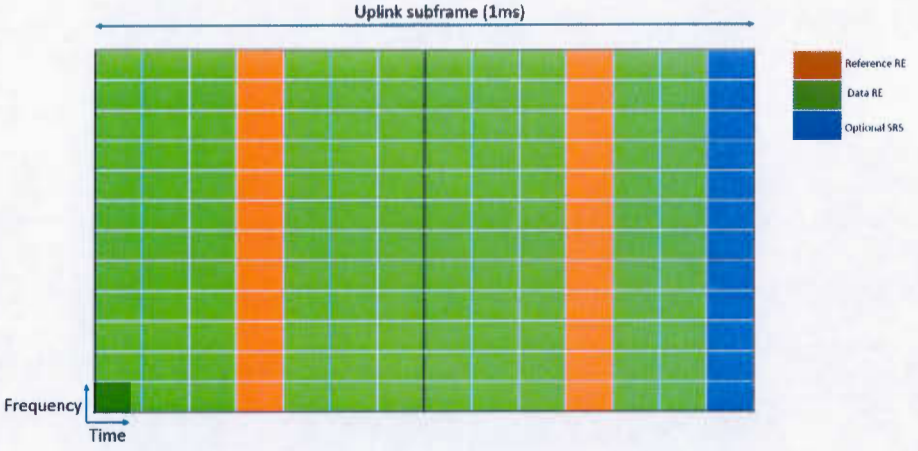


Figure 3.4: Position du signal DRS dans la grille

De nombreux algorithmes sont utilisés pour l'estimation et l'égalisation du canal

basée sur un signal pilote. L'estimé de la matrice du canal H se calcule par une interpolation qui peut être linéaire, de second ordre, cubique ou tout simplement par copie du plus proche voisin. Tout le traitement est basé sur des opérations matricielles de complexité différente. Les plus utilisées dans LTE sont :

— Approximation des moindres carrés : qui minimise la distance quadratique entre le signal reçu et le signal transmis. Elle a une très faible complexité (Laraki *et al.*, 2013).

— Approximation à erreur quadratique moyenne minimale (MMSE) : qui emploi des statistiques de second ordre du canal pour minimiser l'erreur moyenne quadratique. Cet algorithme a une plus grande complexité (Ketonen *et al.*, 2012).

La qualité du signal récupéré dépend entièrement de la qualité du canal estimé H et du bruit e, qui forme le médium physique utilisé pour transmettre les informations de l'UE à l'eNodeB. Supposons que nous ayons une grille de RE reçue Y, le résultat d'une grille de RE transmise X via un canal H avec un bruit gaussien blanc e ayant une variance δ_n^2, l'identifiant d'un RE par son n-ième symbole temporel et sa k-ième sous-porteuse, on trouve :

$$Y(n,k) = X(n,k) \cdot H(n,k) + e. \tag{3.9}$$

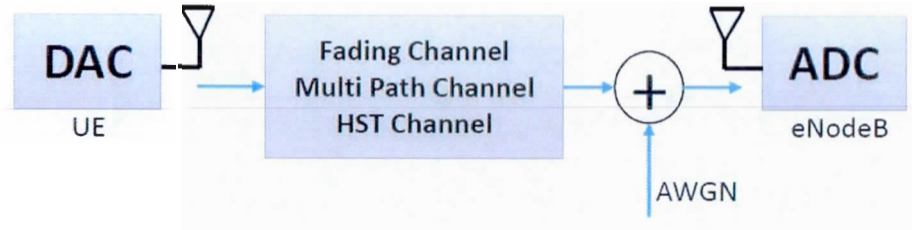


Figure 3.5: Transmission LTE via le médium physique

Dans la figure 3.4, les valeurs des RE dans les positions pilotes sont connues par l'émetteur et le receveur, donc on peut en déduire :

$$Y(k) = X(k)H(k) + e \tag{3.10}$$

Avec k représentant l'index d'un RE.

Appelons $\hat{H}$ l'estimation instantanées des moindres carrés du canal dans le domaine de fréquence aux positions pilotes (le canal H + bruit e), alors on aura :

$$\hat{H_p}(k) = \frac{Y_p(k)}{X_p(k)} = H_p(k) + e \tag{3.11}$$

Ceci nous donne comme résultat un estimé du canal dans les position pilotes sans compter le bruit (Kewen et XingKe, 2010).

Une fois le calcul de $\hat{H_p}$ terminé, $\hat{H}$ aux positions des données utilisateurs peuvent être calculées via une interpolation matricielle comme suit :

— Interpolation linéaire :

$$\hat{H_n} = \hat{H}(n_{k-1}) + \frac{H_p(n_k) - H_p(n_{k-1})}{n_k - n_{k-1}} \times (n - n_{k-1}) \tag{3.12}$$

Avec n le nombre de RE et k l'index du pilote à la position pilote (symbole numéro 4 et 11 de chaque sous-trames).

Puisque n_k et n_{k-1} (k étant le numéro du symbole pilote dans la trame LTE) sont toujours espacés de 7 symboles, alors on peut simplifier l'équation précédente comme suit :

$$\hat{H_n} = \hat{H}(n_{k-1}) + \frac{H_p(n_k) - H_p(n_{k-1})}{7} \cdot (n - n_{k-1}) \tag{3.13}$$

— Interpolation par proches à voisins : cet algorithme suppose que le canal n'a pas beaucoup changé dans une sous-trame, donc la valeur du canal dans les position pilotes est juste recopié vers les RE voisins les plus proches

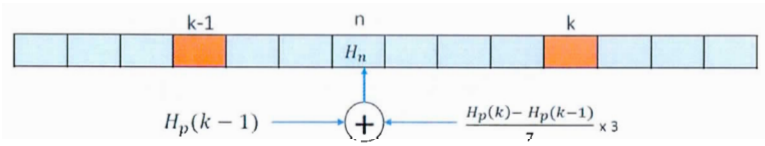


Figure 3.6: Interpolation linéaire dans LTE (Beek *et al.*, 1995a)

à l'exception des RE qui se trouvent sur la bordure de la sous-trame. On remplit ces dernier avec la moyenne des 7 RE les plus proches afin de lisser la valeur du canal entre les sous-trames comme on peut le voir dans la figure 3.7.

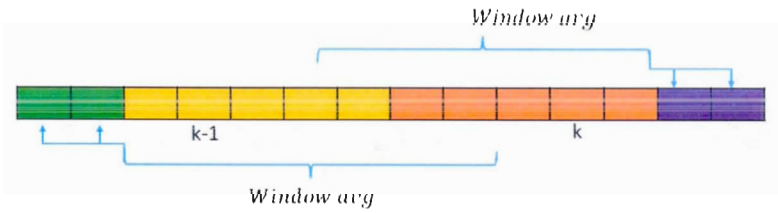


Figure 3.7: Interpolation par proches voisins LTE

L'estimation du niveau de bruit instantané utilise également la valeur du canal aux positions des pilotes (Beek *et al.*, 1995b). Puisque ces valeurs sont déjà connues, la moyenne des estimations du canal dans l'ensemble de la trame nous donne :

$$\hat{H}_p^{avg} = \frac{1}{|s|} \sum_{m \in s} \hat{H}_p(m), \tag{3.14}$$

Avec s un ensemble de RE aux positions pilotes appartenant à la même sous-porteuse et ceci pour toute la trame. $|s|$ est leur nombre (20 par antenne pour la version en cours de TLE), donc ceci nous donne un estimé de la valeur instantanée du bruit dans les positions pilotes comme suit (Beek *et al.*, 1995a) et (TS-36.141, 2016) :

$$e = \hat{H}_p - \hat{H}_p^{avg}, \tag{3.15}$$

On peut calculer la variance du bruit :

$$\delta^2 = \frac{1}{|s|} \sum |Y_p - \hat{H}_p X_p|^2. \tag{3.16}$$

Il n'est pas possible d'éliminer tout le bruit en utilisant la moyenne, puisque le but est de réduire le bruit, seule une estimation de la puissance du bruit peut être calculé (MathWorks, 2015).

L'égaliseur MMSE (Ketonen *et al.*, 2012) produit un filtre qui minimise l'erreur entre le signal transmis et le signal calculé à la réception. Ce filtre est utilisé comme indicateur de la qualité de l'estimateur. Une fois le canal H obtenu, la matrice filtre MMSE peut être calculée comme suit :

$$W = (H^H H + \delta^2)^{-1} H^H \tag{3.17}$$

Avec δ^2 la variance du bruit et $()^H$ la transposée conjuguée complexe.
Une fois le filtre calculé, on peut calculer le signal des données en multipliant la grille reçue par ce filtre comme suit :

$$R_{eq} = W \times R_x \tag{3.18}$$

avec R_{eq} qui représente la grille égalisée et R_x représentant la grille reçue.

Sachant que toutes les opérations sont effectuées sur des données complexes (deux réels par RE), nous pouvons estimer la complexité du module d'estimation et d'égalisation du canal du récepteur LTE (deux valeurs à virgule flottante simple précision). Le tableau ci-dessous donne une estimation du nombre d'opérations pour une antenne en réception (les opérations de copie ne sont pas prises en considération).

Tableau 3.3: Complexité de l'estimation et l'égalisation du canal

Modules	Données	Add/Sous	Mul/Div	OR
Estimation linéaire	48 000	0	1	48 000
Estimation moindres carrées	288 000	2	2	1 152 000
Interpolation (proche voisin)	48 000	7	1	384 000
Variance du bruit	48 000	1	3	192 000
Égalisation MMSE	288 000	1	4	1 728 000

Démodulation QAM

Le signal reçu égalisé est une matrice à valeur complexe, donc afin d'extraire la ces données associées à chaque constellation, deux techniques peuvent être utilisées : une démodulation à décision douce ou dure.

Un démodulateur à décision dure a une faible complexité. La norme LTE a déjà fixé les valeurs de bit pour chaque point de constellation (TS-36.211. 2015), donc ce module renverra la séquence de bits du point de constellation le plus proche. La figure 3.8 illustre la valeur d'un RE en entrée du module qui renverra « 00 » comme résultat.

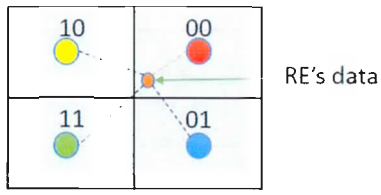


Figure 3.8: Décodage à décision dure QPSK

Étant donné que le démodulateur à décision dure est basé sur la comparaison de la valeur de la donnée reçue avec des seuils, en conséquence des informations peuvent être perdues. Par exemple 0,50001 et 0.999 volts donnent « 1 logique » si

le seuil est de $0,5 >= S < 1$ et cela même si 0,999 est beaucoup plus proche de 1 volt que 0,5 volt, avec S la valeur reçue .

Le démodulateur à décision douce offre une meilleure précision mais avec une complexité plus élevée. Il génère des informations douces en calculant un LLR (Log-Likelihood Ratio), qui est une valeur réelle représentant la sortie du démodulateur, qui est ensuite injectée vers l'entrée de ce module de démodulation (Kerner et Amrani, 2008).

3.4 Parallélisation du récepteur LTE

Les deux entités principales utilisées par le récepteur *UPLINK* sont : l'état actuel de l'eNodeB et la trame physique reçue sur un port d'antenne. Pour que le traitement soit indépendant, deux structures principales d'un programme CUDA ont été créées :

— Trame LTE : une seule instance de trame LTE par bande passante est créée, qui contient toutes les informations sur le signal reçu sur une antenne particulière.

— enodeB : est la structure qui stocke l'état actuel de la station de base, qui est un espace mémoire qui contient la configuration actuelle ainsi que les données, pour chaque bande passante une instance doit être créée et transférée vers la mémoire du GPU. Par exemple le tableau 3.5 affiche la taille de la mémoire réservée pour le mode de fonctionnement 20 MHz.

Tableau 3.4: Structure de la trame reçue et sa taille

Structure	Emplacement	Nbre complexe	Description
rxwave	GPU/CPU	2 457 600	La trame reçue
batch	GPU	2 293 760	Le signal sans le CP
halfShiftedSignal	GPU	2 293 760	Signal décalé d'une demi SC
signalTransform	GPU	2 293 760	Signal transformé après FFT
phaseCorrected	GPU	2 293 760	Signal avec phase corrigée
shiftedSignal	GPU	2 293 760	Signal décalé

Opérations sur le signal

En pratique, à cause du délai de propagation du canal, il est préférable d'ignorer une fraction du préfixe cyclique. Un index que nous pouvons appeler c_f peut être utilisé pour extraire le symbole parce que les positions des échantillons ignorés sont inférieures à la taille du préfixe cyclique. L'index du bloc utile de données dans chaque symbole peut être pré-calculé comme suit :

$$Data_{index} = Symbol_{offset} + c_f, \tag{3.19}$$

Avec $cf = 0$ à 100% du CP_{length} pour le symbole actuel.

Puisque les échantillons de trame capturés sont décalés de n échantillons, alors pour chaque configuration de la bande passante. il faut préparer deux vecteurs de correction de phase (Ling et Proakis, 2017) qui sont enregistrés dans la mémoire du GPU (peuvent être pré-calculer afin d'accélérer de calcul). un pour le premier symbole et l'autre pour le reste des symboles dans une tranche (le préfixe cyclique a une longueur différente pour le premier symbole seulement).

Tableau 3.5: Structure conservant l'état actuel

Structure eNodeB	Lieu	Taille (octet)	Description
symbol_offset	GPU	560	index du CP.
modulationMapperQPSK	GPU	8	LUT pour décodeur QPSK.
modulationMapper16QAM	GPU	16	LUT pour décodeur 16QAM.
modulationMapper64QAM	GPU	32	LUT pour décodeur 64QAM.
phaseCorrection_0	GPU	16 384	Vecteur correction de phase pour symbole #0
phaseCorrection_1	GPU	16 384	Vecteur correction de phase pour symbole #1-6
halfsc_0	GPU	16 384	Vecteur de décalage pour symbole #0
halfsc_1	GPU	16 384	Vecteur de décalage pour symbole #1-6
pilots_grid	GPU	1 344 000	Grille des pilotes
rxgrid	GPU	1 344 000	Grille avant égalisation
hest	GPU	1 344 000	Grille du canal estimé H
egrid	GPU	1 344 000	Grille égalisée.
parallelData	GPU	168 000	Grille après démodulation QAM.
dataOut	CPU	168 000	Sortie binaire

Ce vecteur est sous la forme :

$$CorrectedSignal[k] = T[k] \times e^{\frac{-j2\pi sk}{N}}, \tag{3.20}$$

Avec T le signal transformé après l'opération FFT, N la taille du symbole, $0 \leq k < N$ et s l'échantillon de départ pour le symbole en cours ($Data_{index}$) excluant le CP.

Tableau 3.6: Vecteur de correction de phase pour le mode 20MHz

k	**0**	**1**	**...**	**2047**
Vecteur de correction	$e^{const \times 0}$	$e^{const \times 1}$	...	$e^{const \times 2047}$
Échantillon	$T[0]$	$T[1]$	...	$T[2047]$
Signal corrigé	$T[0]$	$T[1] \times e^{const \times 0}$	...	$T[2047] \times e^{const \times 2047}$

Avec $const = \frac{(-j2\pi \times 160)}{2048}$ à utiliser pour le symbole #0 et $\frac{(-j2\pi \times 144)}{2048}$ à utiliser pour les symboles #1-6 pour la configuration 20 MHz.

La même technique est appliquée pour le décalage fréquentiel du signal d'une demi sous-porteuse, puisque la norme LTE exige que les sous-porteuses en mode « Uplink » soient espacées de chaque côté du DC par un espacement de longueur 7.5 kHz. L'opération est effectuée juste avant l'opération FFT, donc directement sur les échantillons utiles après la suppression du CP :

$$HalfScShiftedSignal[k] = R_x[k] \times e^{\frac{j\pi sk}{N}}. \tag{3.21}$$

Avec $R_X[k]$ le signal reçu excluant le CP.

Étant donné que la sous-porteuse à fréquence nulle est au milieu du spectre, l'opération de décalage est juste une opération qui échange l'emplacement de la

première tranche du symbole avec sa deuxième tranche pour faciliter l'extraction des sous-porteuses utiles (sous-porteuses actives). L'index de la première sous-porteuse utile est donné par :

$$FirstActiveSc = (N/2) - (N_{RB}^{UL} * 6) + 1 \tag{3.22}$$

La figure 3.9 illustre l'opération d'échange et d'extraction des sous-porteuses actives utilisant le mode 20 MHz (Ling et Proakis, 2017).

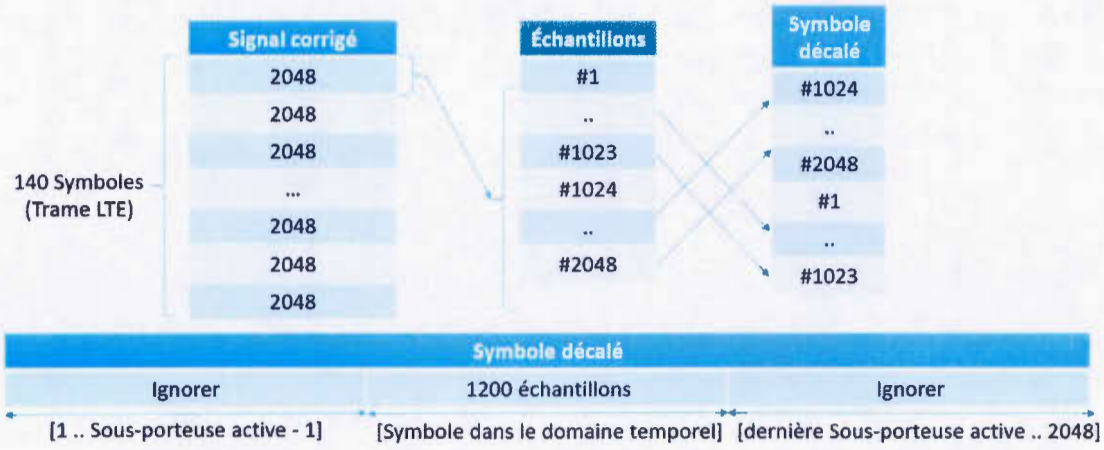


Figure 3.9: Décalage et extraction des sous-porteuses actives

Comme nous avons pu le constater, la plupart des opérations dans les modules de la couche physique sont parallélisables, que ce soit par symbole, par sous-porteuses ou par utilisateur. L'exploitation de la force du calcul des GPU peut être exploitée en implémentant ces opérations en Kernel CUDA. Dans le chapitre suivant, les résultats de cette conversion seront présentés sous la forme de temps d'exécution de chaque module.

CHAPITRE IV

RÉSULTATS ET DISCUSSION

4.1 Implémentation

Préalables

Matlab LTE System Toolbox

L'outil de simulation fourni par MATLAB « LTE Toolbox » peut nous générer des ondes radio compatibles avec le standard LTE, simuler le comportement d'une station de base réceptrice et tester le résultat de notre programme.

Pour la génération de l'onde LTE en mode *Uplink*, « lteRMCULTool » a été utilisé avec les paramètres de la figure 4.1 pour simuler la scénario le plus complexe, qui est :

- 1200 sous-porteuses par symbole (BW à 20 MHz).
- Mode FDD.
- Modulation 64-QAM.
- Génération de 10 sous-trames (Trame LTE complète).

Un autre moyen plus rapide pour générer ces ondes est un script MATLAB qui consiste à utiliser la fonction « lteRMCUL ». Notre code générateur d'onde est représenté dans la liste 4.1, qui est le scénario le plus complexe, mais pour le simpli-

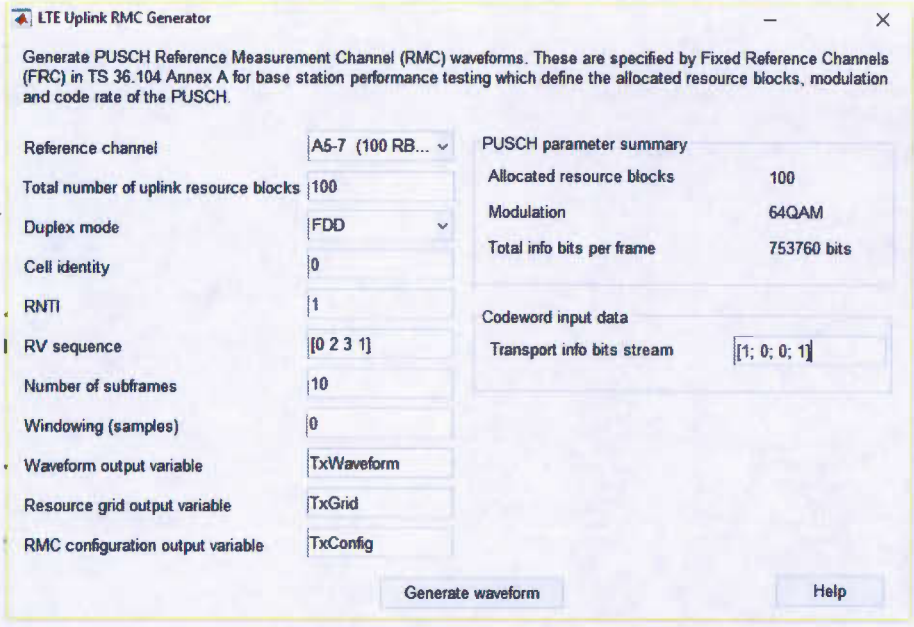


Figure 4.1: Générateur de trame LTE

fier nous supposons qu'un seul UE utilise le spectre, donc il n'y a pas d'extraction d'utilisateurs avant l'IFFT. Le programme MATLAB effectue les opérations :

— Génère une onde de 20 MHz dans le mode *Uplink*.

— Envoie le signal via un canal multi-voies « Rayleigh fading » (Patzold, 2003).

— Démodule le signal reçu (suppression de CP, correction de phase et FFT).

— Estimation du canal et bruit.

— Calcule le filtre MMSE et l'appliquer sur la grille de ressources.

— Effectue une opération IFFT sur tous les échantillons de symboles.

— Décode tous les RE avec un démodulateur 64QAM à décision dure (le seuil est de 50%).

Listing 4.1: Générateur de trames LTE Uplink

```
ue = lteRMCUL( A5-7 );
[TxWaveform,TxGrid,TxConfig]  = lteRMCULTool(ue,[1;0;0;1]);
channel.ModelType = 'GMEDS'; %Rayleigh fading channel
channel.DelayProfile = 'EVA'; % EVA delay spread
channel.DopplerFreq = 70; % 70Hz Doppler frequency
channel.MIMOCorrelation = 'Medium'; %Correlation between UE and eNodeB antennas
channel.NRxAnts =1;  % 1 receive antenna
channel.InitTime = 0; % Initialize at time zero
channel.InitPhase = 'Random'; % Random initial phases
channel.Seed = 17; % Random channel seed
channel.NormalizePathGains = 'On'; % Normalize delay profile power
channel.NormalizeTxAnts = 'On'; % Normalize for transmit antennas
channel.SamplingRate = info.SamplingRate;
channel.NTerms = 16; % Oscillators used in fading model
rxWaveform = lteFadingChannel(channel,[txWaveform;zeros(25,1)]);

rxGrid = lteSCFDMADemodulate(ue,rxWaveform);
cec = struct('FreqWindow',1, TimeWindow',7,'InterpType', nearest','Window','Left );% channel
    estimation config structure with nearest neighbor algorithm using 7 RE averaging window
[hest, noise_est] = lteULChannelEstimate(ue,ue.PUSCH,cec,rxGrid);
eqgrid = lteEqualizeMMSE(rxGrid,hest,noise_est);
for symbol = 1:140
  temp=ifft(eqgrid(:,symbol),1200,'nonsymmetric');
  out(:,symbol) = lteSymbolDemodulate(temp,'64QAM','Hard');
end
```

Étant donné que les symboles des pilotes sont des données connues à la fois par l'émetteur et le récepteur, la fonction « ltePUSCHDRS » a été utilisée pour générer les symboles DRS avec les paramètres globaux actuels. Les symboles ont été sauvegardés dans un fichier binaire avec l'onde générée pour être utilisés dans le programme CUDA.

Nsight pour Microsoft Visual Studio

En ce qui concerne l'environnement de développement, nous avons utilisé l'outil CUDA de Nvidia appelé « Nsight », qui est un complément de l'environnement « Microsoft Visual Studio ». Il comprend le compilateur de NVIDIA « nvcc » avec les bibliothèques de base nécessaires. Le programme LTE a été compilé avec la version 5.2 de Nsight installé sur Visual Studio 2013.

Le matériel informatique utilisé pour exécuter le programme CUDA est un ordina-

teur de bureau Intel I5-4690K CPU. Le CPU fonctionne à 3,5 GHz avec 16 Go de mémoire DDR3-1600 et une carte graphique NVIDIA GTX-780 (3,91 TFLOPS) avec Windows 10 Pro 64bits comme système d'exploitation.

4.2 Temps d'exécution

Le code Matlab précédent a pris 4 secondes pour s'exécuter, dont 3 pour terminer l'égalisation MMSE. Même si cette fonction est déjà compilée, le programme MATLAB était à sa performance maximale. On peut voir que le CPU ne peut pas traiter les trames LTE en temps réel. Nous avons testé la performance du programme équivalent à l'aide de CUDA en mode SISO, car il n'y a pas de mappage ou de précodage de couche de transmission lorsqu'une seule antenne est utilisée pour la transmission et la réception. 100000 trames LTE ont été traitées et leur temps de calcul en millisecondes a été sauvegardé . Tous les modules ont été exécutés séquentiellement comme indiqué dans la figure 4.2.

La séquence de traitement est :

- Lire les deux fichiers binaires contenant la trame LTE à partir du disque (l'un pour la partie réel et l'autre pour la partie imaginaire des échantillons) et les mettre dans la RAM du CPU. Cette étape n'a pas été incluse dans le calcul du temps car elle n'a rien à voir avec le protocole lui-même.
- La trame est ensuite transférée vers la RAM du GPU pour le traitement, comme indiqué sur la figure 4.3.
- Démarrer le traitement de la trame en supprimant le CP à l'aide d'un tableau contenant les positions des symboles et appliquer le décalage d'une demi sous-porteuse, comme indiqué dans la figure 4.4.
- L'opération FFT est effectuée sur tous les 140 symboles en utilisant la librairie précompilée de CUDA « CuFFT », la figure 4.5 montre le temps d'exécution.

— Une correction de phase (Li *et al.*, 2016) est appliquée sur tous les échantillons en utilisant un tableau de correction de phase statique. Le résultat est indiqué dans la figure 4.6.

— Les sous-porteuses actives sont extraites pour créer la grille de ressources, qui est une matrice 2D. Le temps d'exécution est illustré dans la figure 4.7.

— La canal H et le bruit sont estimés en utilisant l'algorithme du plus proche voisin puisqu'il est plus faible en complexité que l'interpolation linéaire. Le temps de traitement est représenté dans la figure 4.8. Cette estimation est fait en plusieurs étapes successive comme vu dans le chapitre 3. Le temps d'exécution de chacune a été calculer à part, comme le calcul de H_p dans les positions pilotes, opération d'interpolation suivant l'algorithme du plus proche voisin, calcul de la moyenne des RE appartenant à la même sous-trames, remplissage du contour de la trame (Watson, 1992) et estimation de la valeur instantanée et moyenne du bruit.

— L'égalisation MMSE commence par la création d'un filtre qui représente le canal sur lequel le signal a été envoyé. Ensuite la grille de ressource est calculée en utilisant la grille reçue et le filtre MMSE déduit. Le temps de traitement pour ce module est illustré dans la figure 4.9.

— L'opération IFFT s'est faite à l'aide de la librairie « CuFFT » aussi, mais la bibliothèque ne divise pas par N le résultat, de sorte que le *Kernel* « PostIFFT » a été ajouté pour effectuer l'opération de division. Son temps de traitement est indiqué dans la figure 4.10.

— Le démodulateur QAM transforme la grille de ressources à valeur complexe vers une à valeur binaire, cette transformation est effectué via une démodulation à décision dure puisque c'est l'algorithme le moins complexe. Son temps de traitement est représenté dans la figure 4.11.

— La grille résultante est ensuite copiée vers le programme hôte pour être

mise en mémoire RAM du CPU. Le temps de traitement est affiché dans la figure 4.12.

Le temps de traitement de tous les modules est combiné dans la figure 4.13 qui nous montre le temps de traitement global écoulé pour traiter les signaux *Uplink* à la réception.

Le tableau 4.1 montre que les opérations FFT / IFFT sont clairement les goulets d'étranglement des modules de traitement des signaux de la station de base LTE.

Tableau 4.1: Temps d'exécution moyen total

	Temps moyen (ms)
Copie CPU/GPU	0,436
Extraction & décalage	0,087
FFT	1,345
Correction de phase & décalage	0,088
Extractopn de Sous-porteuses actives	0,050
Estimation de canal et du bruit	0,248
Égalisation MMSE	0,074
IFFT	1,301
Démodulateur 64-QAM	0,079
Copie GPU/CPU	0,052
Total	5,054

4.3 Comparaison

En obtenant le temps de calcul moyen des opérations sur les trames LTE, on peut comparer nos résultats avec l'un des travaux antérieurs le plus intéressant (Zheng *et al.*, 2013). Le temps de calcul des quelques modules LTE dans l'article est donné pour le traitement non temps réel d'une seule sous-trame LTE (Utilisant

une carte graphique NVIDIA GTX680). En estimant que le temps de traitement reste stable pendant l'exécution et en multipliant par dix leurs résultats, voici quelques différences avec les nôtres du tableau 4.1 :

— Modules FFT/IFFT : la bibliothèque cuFFT a été utilisée par l'équipe pour les opérations de transformation de Fourier. Ils ont traité le module FFT en 0.6 ms et IFFT en 1 ms, ce qui est un peu plus rapide que nos résultats. Ceci peut être expliqué par le fait qu'ils ont augmenté manuellement la fréquence d'horloge du GPU à 1 GHz, alors que dans notre cas, on a utilisé la configuration par défaut qui est de 862 MHz.

— Estimation de canal : les auteurs ont utilisé la technique « Least square estimate » (Coleri *et al.*, 2002) et ont obtenu 0.2 ms comme temps d'exécution (Estimation du bruit n'est pas précisée), ce qui est un peu similaire à nos résultats de 0.248 ms. Même en ayant un GPU qui a plus de cœurs, beaucoup d'opérations dans l'estimation de canal sont séquentielles, comme les opérations d'interpolation et de moyenne, ce qui explique le gain minime en performance.

— Égalisation : les auteurs ont utilisé l'égalisation du type MMSE et ont obtenu un temps de traitement de 0.2 ms, alors que dans nos tests on a obtenu 0.074 ms, cela peut être expliqué par la différence dans la force du calcul, 2304 cœurs dans notre GPU et 1536 dans le leur. Les opérations d'égalisations sont indépendantes dans ce module pour tous les RE, donc plus on a des cœurs disponibles plus on réduit le temps de calcul.

— Démodulateur 64QAM : les auteurs ont obtenu 4.7 ms comme temps de traitement pour ce module, alors que le nôtre a donné 0.079 ms. Le démodulateur qu'ils ont utilisé est à décision douce et nous avons implémenté un à décision dure. Ce dernier a beaucoup moins de complexité puisque les valeurs résultats sont stockées dans une table de correspondance. Cette différence

nous donne un aperçu du temps de traitement des deux implémentations.

4.4 Optimisation

La librairie logicielle CuFFT donne un temps de calcul assez bon pour traiter toute la trame LTE qui est de 1.345 ms pour 140 symboles de 2048 échantillons. D'autres librairies existent sur le marché pour faire des opérations de transformation de Fourier, parmi eux, on trouve MKL de Intel (Intel, 2015).

Étant donné que nous utilisons CuFFT qui est une bibliothèque CUDA compilée pour les FFT, le traitement du module de démodulation de la trame reçue avec la bibliothèque MKL d'Intel a été testé pour comparer les performances des deux bibliothèques et identifier le temps de traitement et les latences supplémentaire si elles existent.

La bibliothèque Intel MKL fonctionne uniquement avec le compilateur de Intel *ICC* , ce qui est regrettable car nous ne pouvons pas l'utiliser directement dans notre programme CUDA, c'est pourquoi un autre programme a été compilé avec *ICC* qui fonctionne parallèlement avec le programme principal CUDA pour tester si l'envoi des symboles vers le 2e programme a une valeur ajoutée dans notre calcul. Le problème avec cette méthode est qu'il faut transférer les symboles du GPU vers le CPU pour effectuer les opérations FFT, puisque les processus ne peuvent pas lire ou modifier les données dans la mémoire du GPU (seulement un Kernel CUDA peut opérer sur les données) pour que l'autre processus les traite.

Étant donné qu'on a déjà transféré la trame au GPU pour les opérations avant la FFT, cette copie ajoute un temps de traitement supplémentaire, comme on peut le voir dans le tableau 4.2. Pour la communication entre les deux processus, la mémoire partagée du système d'exploitation a été utilisée pour les données d'entrée et le tableau de résultats. Un Mutex, illustrer dans la figure 4.15, permet de

Tableau 4.2: Temps de Transfert avec MKL

BW(20Mhz)	Transfert CPU/GPU (us)		
	Min	Max	Moyenne
CPU à GPU	268	381	276,5
GPU à CPU	394	568	408,4
Total	662	949	684,9

Tableau 4.3: Temps d'exécution du processus Intel

BW(20Mhz)	Temps (us)		
	Min	Max	Moyenne
FFT avec MKL	495	546	499,2

synchroniser les deux afin d'obtenir une exclusion mutuelle efficace. Le temps de traitement des opérations FFT avec le processus utilisant Intel MKL est illustré dans le tableau 4.3, on peut voir aussi dans la figure 4.14 que le système d'exploitation ne cause pas un retard problématique puisque le temps de calcul de la FFT n'a jamais dépassé 546 microsecondes pour le traitement des 30 000 trames générées pour tester ce genre d'opération.

4.5 Décodage des canaux sur CPU

Une fois le décodage des signaux terminé, les données qui sont devenus numériques doivent être décodés pour obtenir l'information utile à la couche MAC. a plupart du traitement dans cette phase est séquentiel, on a décidé de l'exécuter directement dans la partie hôte du programme, qui est exécuté par le CPU.

La séquence de traitement sur les données binaires obtenues du décodeur QAM est la suivante :

— De-brouilleur (Descrambler) : dé-brouillage du canal avec une séquence pseudo aléatoire.

— Désentrelaceur (De-interleaver) : réorganise les données, le RI et le ACK des symboles reçus.

— Dé-multiplexage (Demultiplexing) : extraction des données utilisateurs et de contrôle.

— Dé-concaténation (Deconcatenation) : dé-concatenation des CB.

— Adaptateur de débit inverse (reverse rate matching) : prépare les flux de données à partir des données encodées par le turbo codeur.

— Décodeur turbo (Turbo decoder) : décoder les flux pour les transformer en données utiles comme les CB ou TB (voir section 5.1.3.2 (TS-36.211, 2015)).

— De-segmentation : c'est une étape qui identifie les CB et les extrait pour pouvoir calculer leurs CRC.

— CB CRC : c'est une étape optionnelle, qui permet de calculer le CRC des CB si le TB a été segmenté, la réponse ACK/NACK dépend de ce calcul.

— TB CRC : Calculer le CRC du TB, une deuxième réponse ACK/NACK dépend de ce calcul.

4.6 Discussion

Dans la figure 4.13, nous avons remarqué que pendant le traitement des 100 000 trames (5 millisecondes en moyenne pour chacune), le récepteur LTE était stable (le temps de traitement varie peu) à l'exception de deux trames, ceci est dû au ralentissement causé par le système d'exploitation. Ce retard indique que la fiabilité (garantir un temps maximal de traitement d'une trame LTE) d'un logiciel eNodeB ne peut pas atteindre les 100% tant que nous utilisons un système d'exploitation non temps réel. Même lorsque nous avons essayé de traiter quatre trames à la fois en utilisant le même processus, ces retards sont devenus plus fréquents.

Tableau 4.4: Décodage de canaux sur CPU

Décodage de canaux sur CPU	Max(ms)	Min(ms)	Moyenne(ms)
De-brouilleur	0,406	0.098	0,121
Désentrelaceur	3,964	1.345	1,413
Dé-multiplexage	1,459	0.502	0,555
Dé-concaténation	1,413	0.42	0,454
Adaptateur de débit inverse	4,314	1.414	1,497
Décodeur turbo	2,139	0.217	0,231
De-segmentation	1,506	0.478	0,514
CB CRC	3,236	0.734	0,787
TB CRC	3,215	0.741	0,811
Total	21,652	5.949	6,383

Comme on peut le voir à partir des tableaux 4.2 et 4.3, la performance MKL est supérieure à CuFFT en plus d'avoir une meilleure stabilité lors des opérations FFT, car ce module n'utilise pas la carte graphique (pas d'appel au pilote de la carte). Le retard causé par le transfert des symboles supplémentaire rend cette solution moins pratique, car il n'y a pas de gain en performance, en plus cette solution augmente les appels au système d'exploitation puisqu'il utilise la mémoire partagée en vérifiant constamment le mutex.

L'usage du CPU pendant tout le traitement du programme utilisant la bibliothèque MKL était à 100%, donc tous les cœurs étaient occupés, en plus d'être dans le mode SISO, le temps de traitement utilisant Intel MKL va doubler ou quadrupler si on passe au mode MIMO. En ce qui concerne les GPU, on peut mettre jusqu'à quatre cartes graphiques dans la plupart des ordinateurs de bureau (ou plus dans ceux de type serveurs), ce qui donnera le même temps de traitement

des trames pour le mode SISO et MIMO (1.345 ms pour quatre trames LTE).

En regardant la couche physique LTE dans la figure 3.1, nous avons réussi à traiter presque tous les modules dans l'enodeB récepteur en mode *Uplink* en seulement la moitié du temps alloué, puisque la longueur de la trame est de 10 ms et cela a été effectué sur une carte graphique capable de faire 3.97 TFLOPS. L'occupation des noyaux du GPU était de 100 % (tous les SMX étaient occupés pendant l'exécution) pour tous les modules (pas de SMX inactif) à l'exception des modules FFT / IFFT qui avaient 25% d'occupation. Cette baisse peut être expliquée par la nécessité qu'un stage de la transformation de Fourier doit se terminer avant que le prochain commence son calcul, ce qui introduit un aspect séquentiel à cette étape.

La raison pour laquelle le décodeur de canaux n'a pas été traité en CUDA, est que les modules concernés font du calcul binaire séquentiel, donc le traitement se fait bit à bit à l'aide d'opérateurs logiques. En pratique, ce genre de module est implémenté en FPGA puisque les techniques comme le décodeur turbo ou le calcul du CRC sont communes à beaucoup de technologies. Pendant nos test, le calcul du CRC par exemple a été quatre fois plus rapide sur le CPU qu'en GPU et peut être traité beaucoup plus rapidement si on l'exporte vers un périphérique FPGA.

Les résultats obtenus peuvent nous donner un aperçu des avancements actuels dans le domaine des solutions logicielles qui remplacent graduellement leurs équivalents matériels dans le domaine de la télécommunication.

Dans le chapitre suivant, on discutera de potentiels travaux futurs dans le domaine de la virtualisation de la couche physique LTE et nous conclurons ce mémoire.

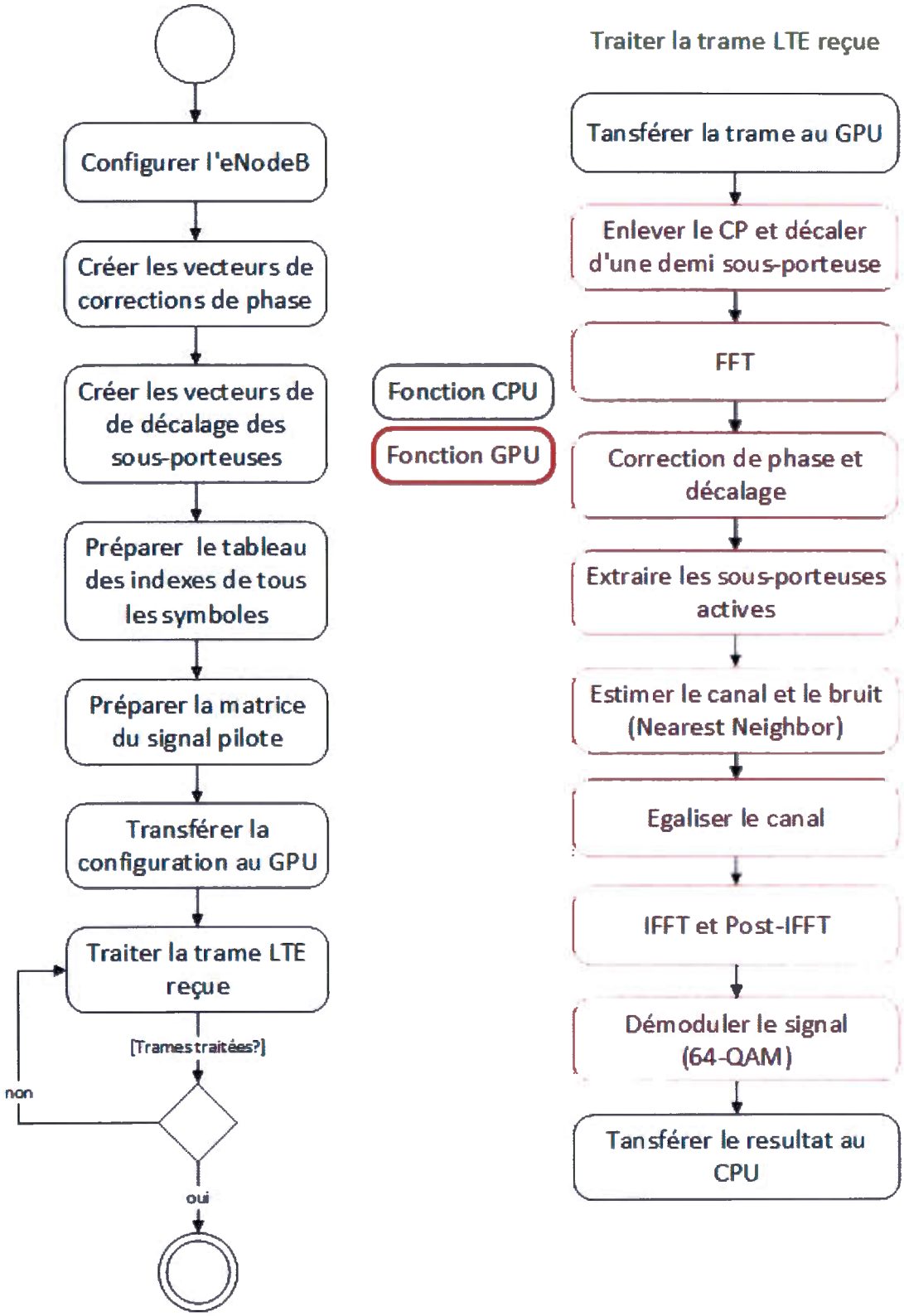


Figure 4.2: Exécution du récepteur *Uplink* avec CUDA.

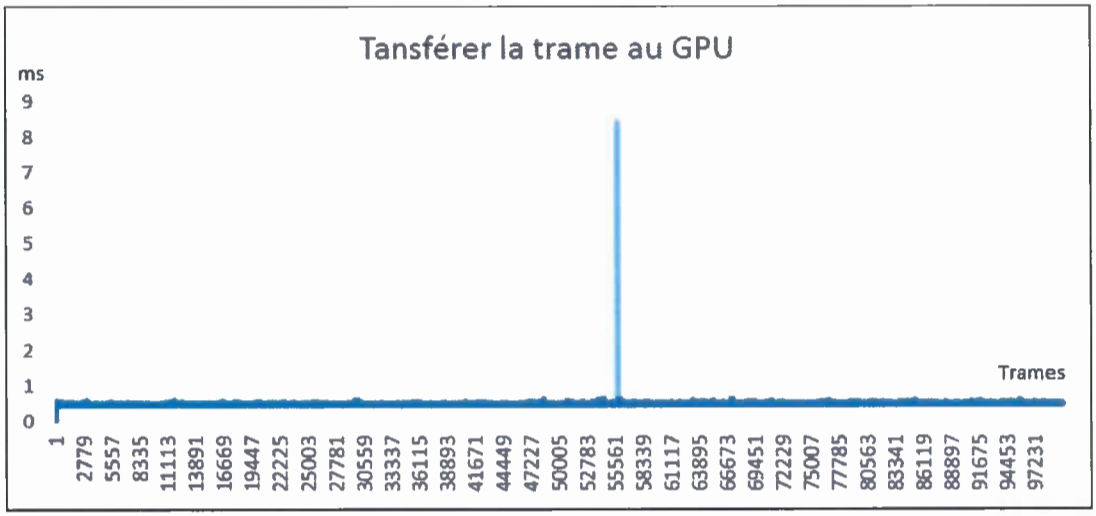


Figure 4.3: Copier la trame vers la RAM du GPU

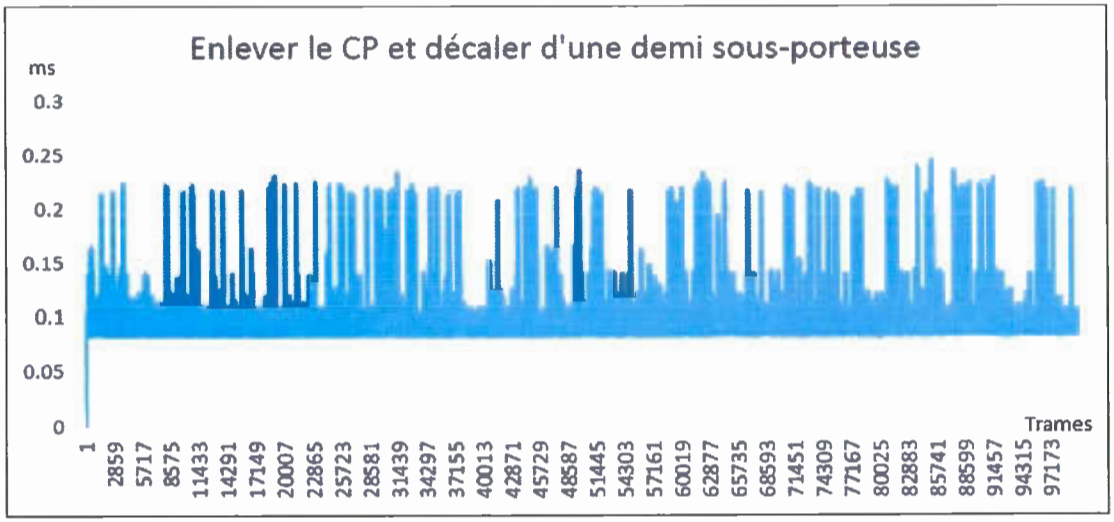


Figure 4.4: Ignorer le CP et décaler le signal

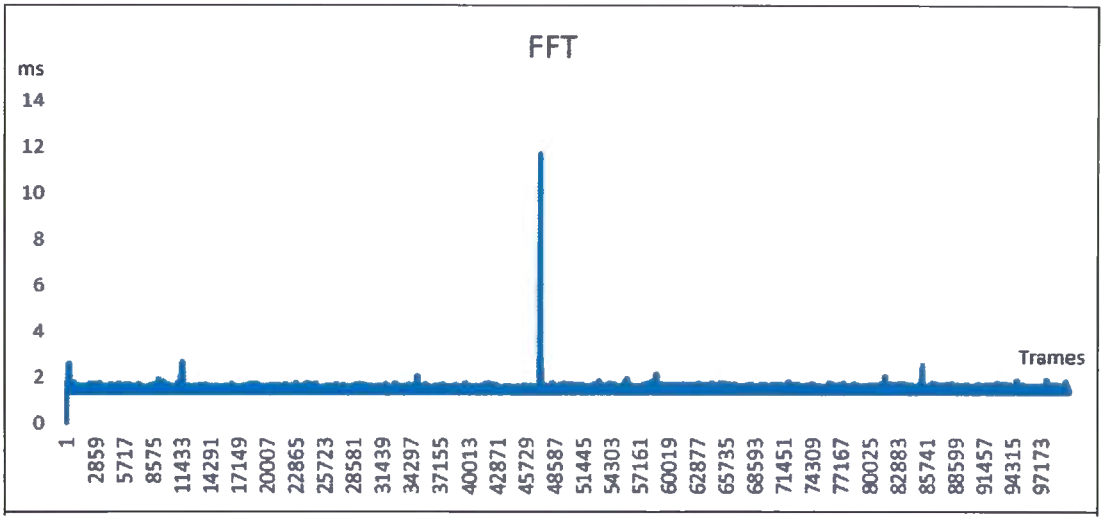


Figure 4.5: FFT

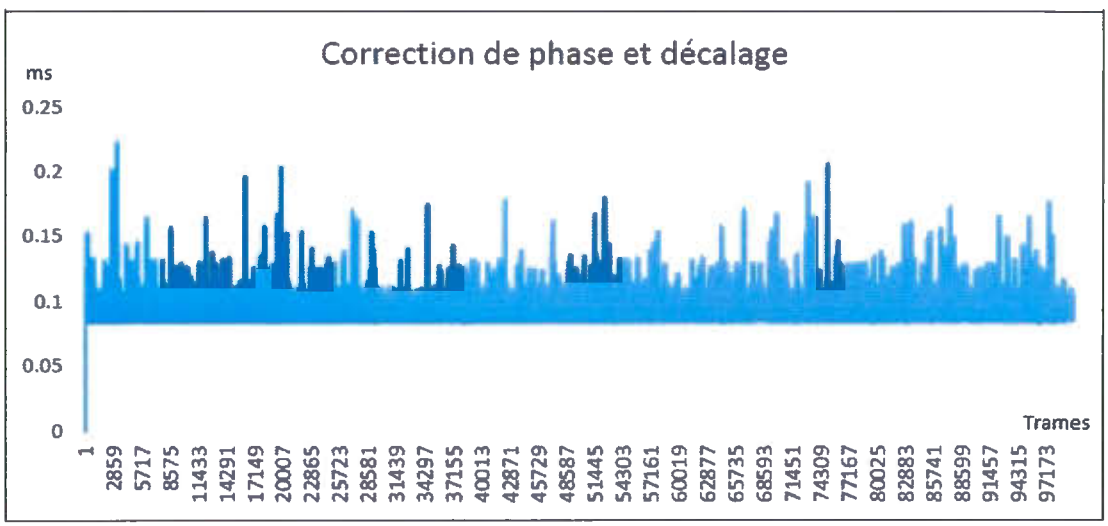


Figure 4.6: Correction de phase et décalage

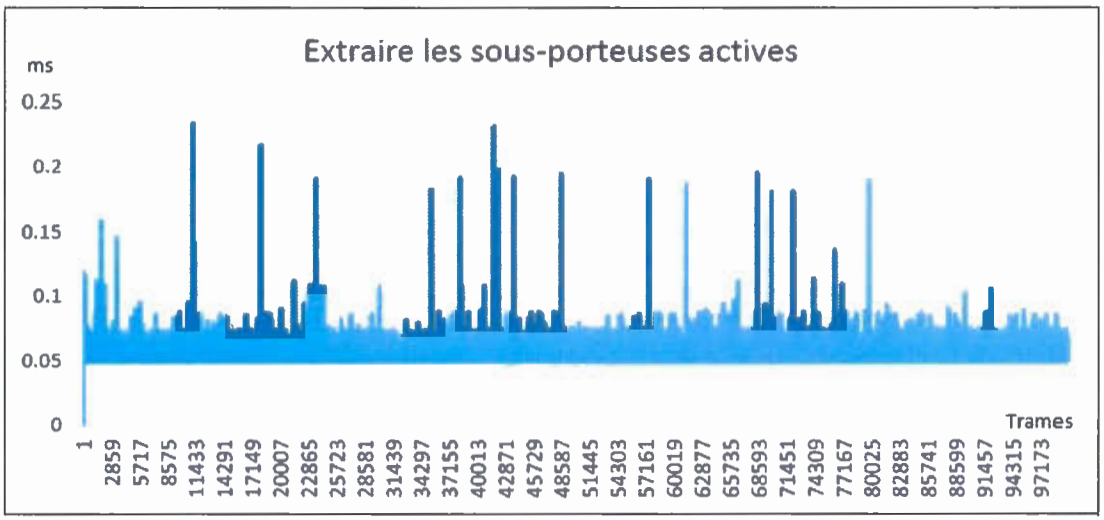


Figure 4.7: Extraction des sous-porteuses actives

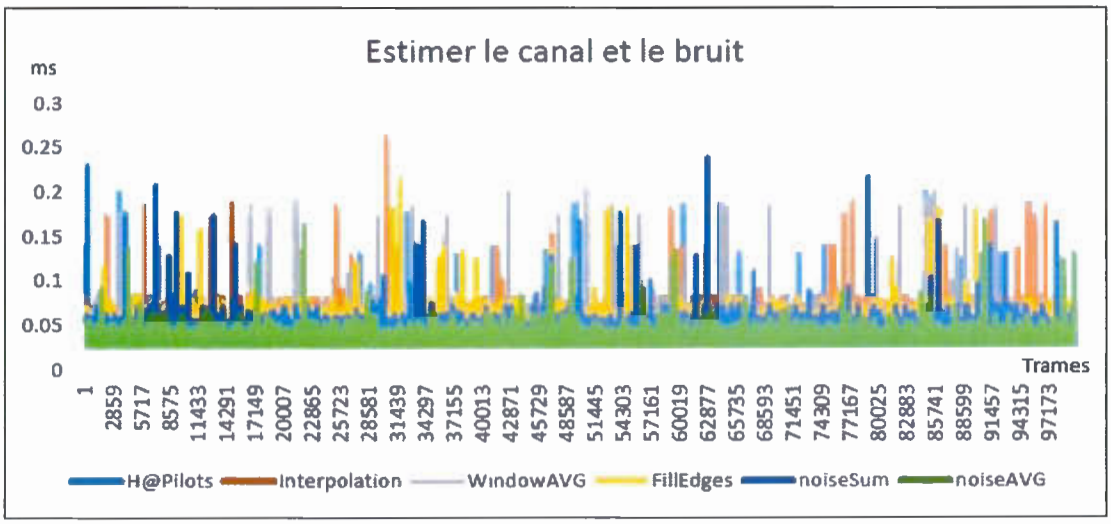


Figure 4.8: Estimation du canal H avec bruit

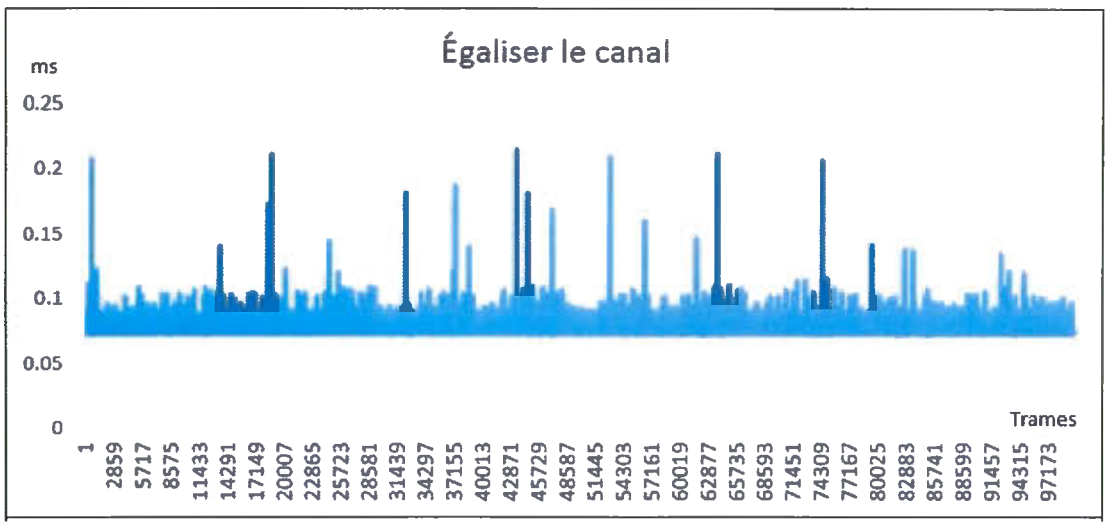


Figure 4.9: Égalisation MMSE

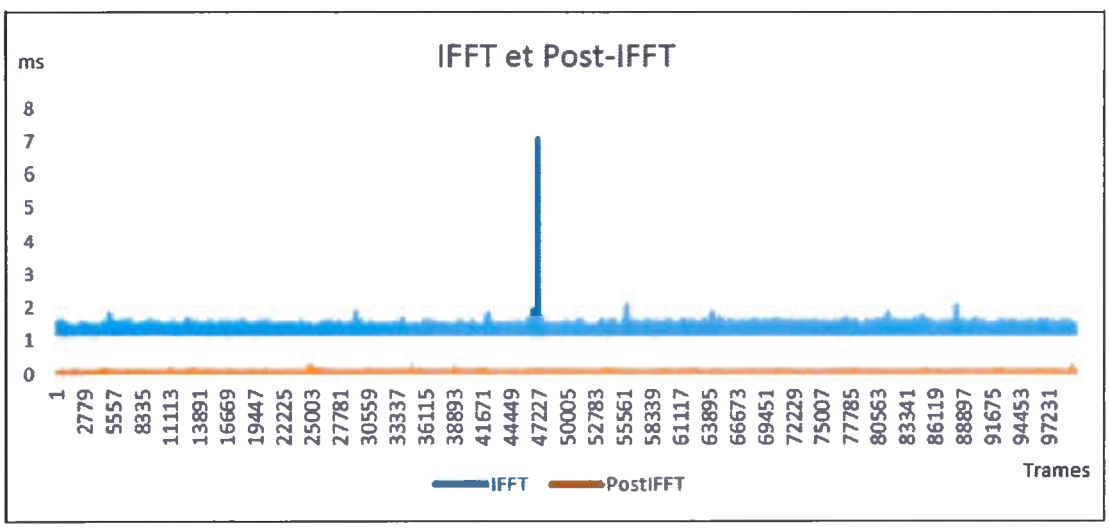


Figure 4.10: Opération IFFT pour tous les symboles

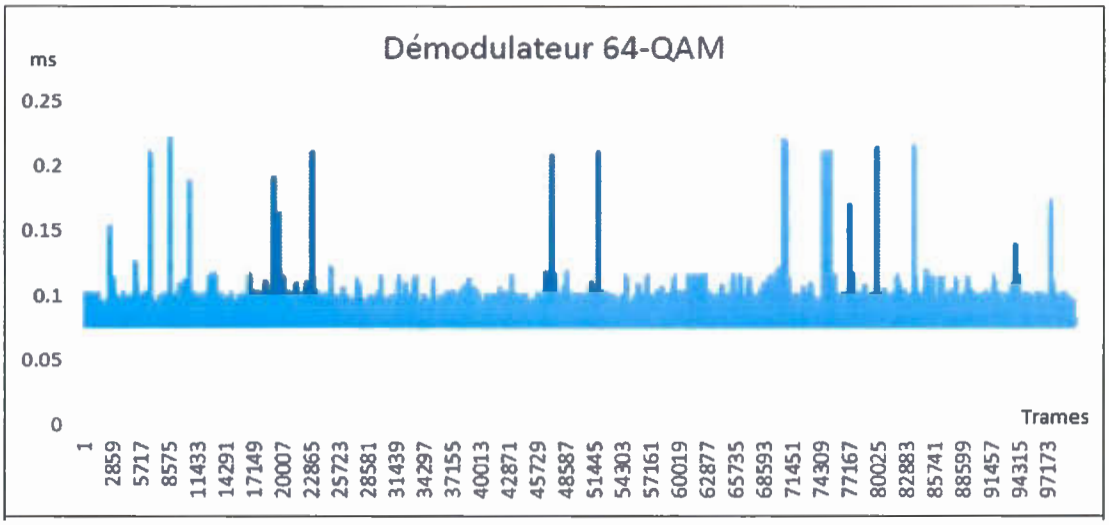


Figure 4.11: Démodulateur 64-QAM

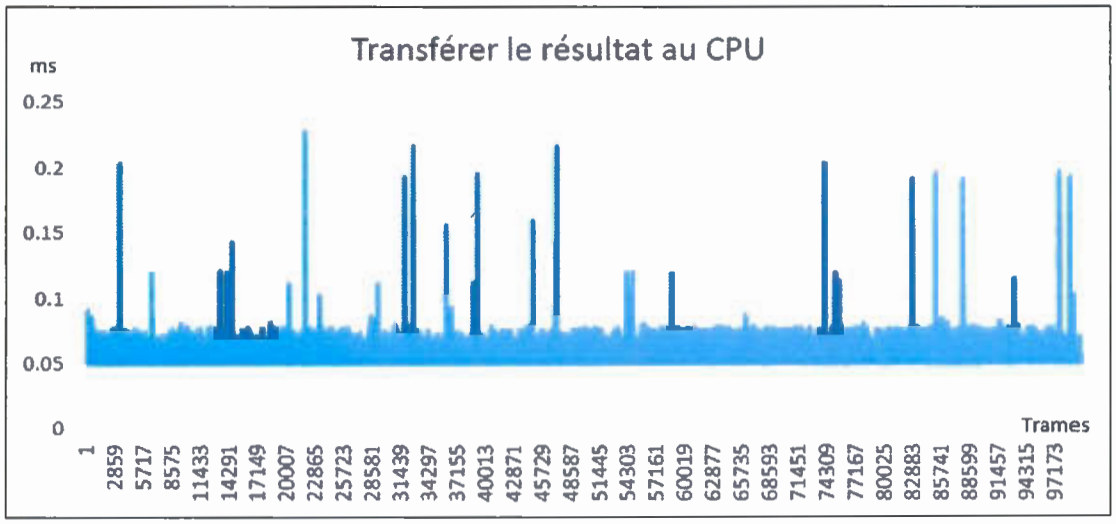


Figure 4.12: Transférer les résultats vers le CPU

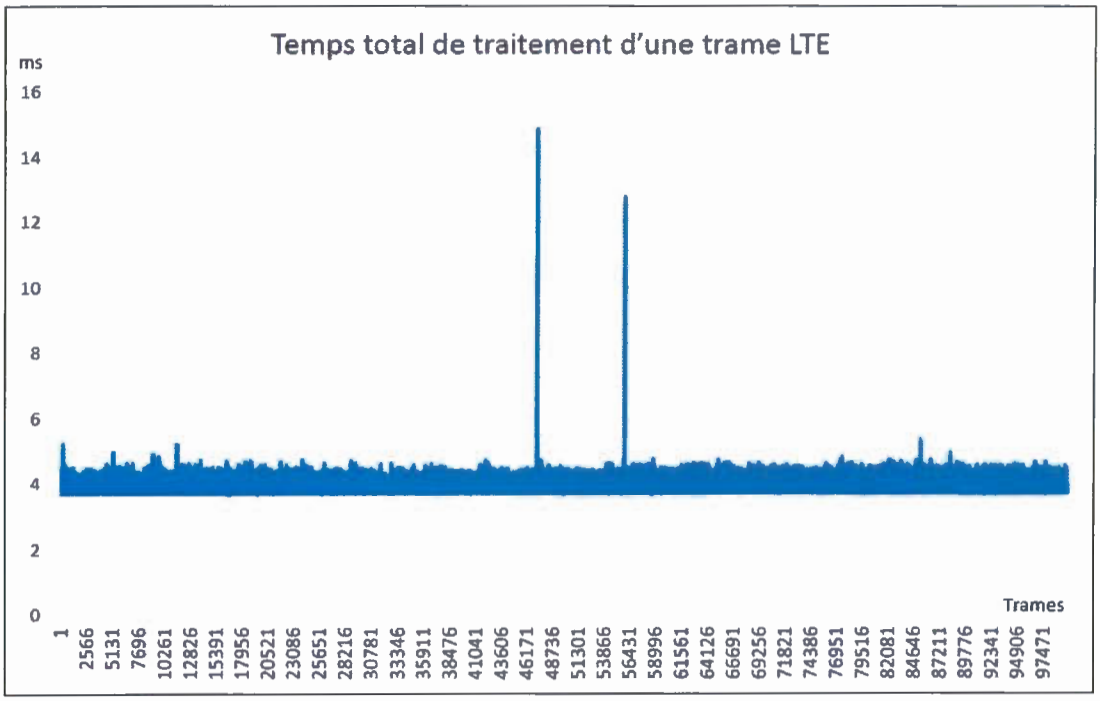


Figure 4.13: Temps de traitement d'une trame avec GPU

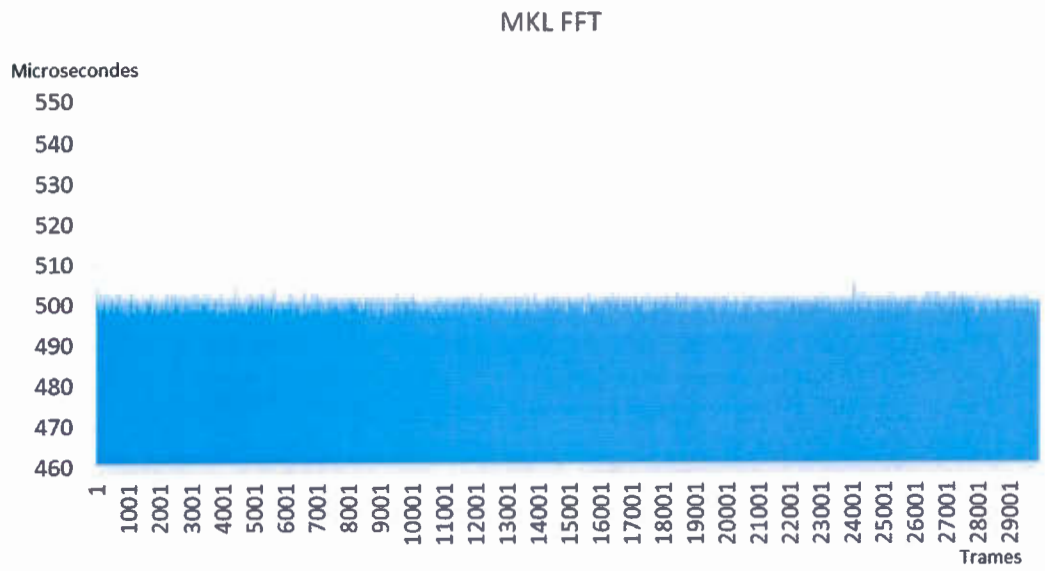


Figure 4.14: Temps d'exécution de la FFT avec Intel-MKL

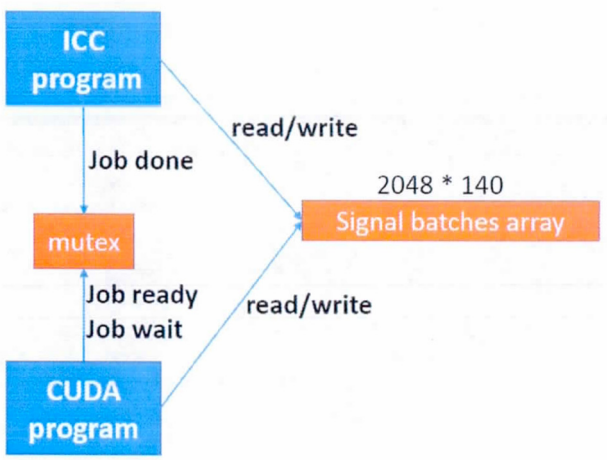


Figure 4.15: Communication inter-processus CUDA/MKL

CONCLUSION

Une étude plus approfondie peut être effectuée dans la virtualisation de la couche physique des protocoles sans fil, par exemple la lecture/écriture des trames qui communique avec une antenne n'a pas été implémentée dans cette recherche, car elle n'appartient pas à LTE lui-même, mais cela peut ajouter d'autres goulets d'étranglement liés à l'interface réseau contrôlés par le système d'exploitation.

Le décodage des canaux n'a pas été optimisé dans cette recherche puisque ces opérations sont de nature séquentielle et logique. C'est pourquoi même si le CPU est beaucoup plus rapide que le GPU dans ce genre d'opération, ces performances ne sont pas satisfaisantes (et cela pour une seule antenne), donc une solution par périphérique externe peut être plus fiable.

Dans les versions futures de notre station de base, nous pouvons ajouter par exemple :

— L'utilisation d'un système d'exploitation en temps réel pour éliminer les pointes que nous avons vues dans le temps de traitement global dans la figure 4.13.

— Implémenter le module de décodage dans un périphérique externe comme un FPGA ou un DSP.

— Utiliser une carte graphique plus puissante (10+ TFLOPS GPU) pour tester le temps de traitement du mode MIMO multi-utilisateur (trames séparées par utilisateur) et le MIMO de diversité Rx (1 trame /2 antennes).

— Développer l'interface communicant avec une antenne pour faire des tests

dans un environnement réel.

Cette étude montre qu'un modem purement logiciel pour le protocole temps réel LTE est possible, un processeur d'une carte graphique utilisé pour faire du calcul parallèle remplaçant ainsi les solutions actuelles peut s'avérer profitable pour les fournisseurs de services de télécommunication. Un tel logiciel informatique agit d'une manière similaire aux coprocesseurs mathématiques des premiers CPU Intel. Les données obtenues prouvent que les GPU peuvent remplacer la solution ASIC coûteuse ou la consommation élevée d'un FPGA dans un avenir proche. En même temps, nous avons vu que le protocole LTE est très complexe et utilise des techniques récentes, ce qui rend cette étude utile comme base de comparaison avec d'autres protocoles au niveau de la couche physique qui partage certains modules avec LTE.

Les futurs protocoles de communication utiliseront plus d'antennes, ce qui rendra une synchronisation des flux de données en matériels plus difficiles, comme l'utilisation de huit antennes pour l'émission et la réception, alors que ceci est géré automatiquement avec une approche logicielle telle que CUDA. Avec les changements très fréquents du standard, le coût du développement, maintenance et mises à jour des stations de base LTE sera considérablement réduit.

BIBLIOGRAPHIE

Agilent (2010). *Introducing LTE-Advanced, Application Note.* www.3g4g.co.uk/LteA/LteA_Overview_1011_Agilent.pdf.

Arch, C. (2007). *CUDA Architecture Overview.* http://developer.download.nvidia.com/compute/cuda/docs/CUDA_Architecture_Overview.pdf.

Artizanetworks. (2016). *Physical Channel Structure.* http://artizanetworks.com/lte_resources/lte_tut_phy_cha.html.

Beek, J.-J. V. D., Edfors, O., S, M., Wilson, S. K., Brjesson, P. O., Kate, S., Per, W. I. et Brjesson, O. (1995a). On channel estimation in ofdm systems. Dans *in Proc. of the IEEE Vehicular Technology Conference, VTC '95*, 815–819.

Beek, J. J. V. D., Edfors, O., Sandell, M., Wilson, S. K. et Borjesson, P. O. (1995b). On channel estimation in ofdm systems. Dans *1995 IEEE 45th Vehicular Technology Conference. Countdown to the Wireless Twenty-First Century*, volume 2, 815–819 vol.2.

Bhattacharjee, S., Yadav, S. et Patra, S. (2014). Lte physical layer implementation using gpu based high performance computing.

Blahut, R. E. (1985). *Fast Algorithms for Digital Signal Processing* (1st éd.). Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc.

Buck, Ian, Foley, Tim, Horn, Daniel, Sugerman, Jeremy, Fatahalian, Kayvon, Houston, Mike, Hanrahan et Pat (2004). Brook for gpus : Stream computing on graphics hardware. *ACM Trans. Graph.*, *23*(3), 777–786.

Cheng, J. F., Nimbalker, A., Blankenship, Y., Classon, B. et Blankenship, T. K. (2008). Analysis of circular buffer rate matching for lte turbo code. Dans *2008 IEEE 68th Vehicular Technology Conference*, 1–5. http://dx.doi.org/10.1109/VETECF.2008.162

Chi, C.-L. et Kuo, C.-H. (2012). Quadratic permutation polynomial interleaver for lte turbo coding. *2012 International Conference on Information Security and Intelligent Control*, 313–316.

Cho, I., Patyk, T., Guevorkian, D., Takala, J. et Bhattacharyya", S. ("2013"). *"Pipelined FFT for Wireless Communications Supporting 128-2048 / 1536 - Point Transforms"*, Dans *"1st IEEE Global Conference on Signal and Information Processing, GlobalSIP 2013, December 3-5, 2013, Austin, TX, USA"*, (p. "1242–1245"). "IEEE".

Coleri, S., Ergen, M., Puri, A. et Bahai, A. (2002). Channel estimation techniques based on pilot arrangement in ofdm systems. *IEEE Transactions on Broadcasting*, *48*(3), 223–229.

CudaDev (2007). *CUDA Programming Guide.* `https://docs.nvidia.com/cuda/cuda-c-programming-guide/`.

Davis, C. R. (2001). *Ipsec : Securing Vpns.* McGraw-Hill Professional.

Ibrahim, M., Kamal, M., Khan, O. et Ullah, K. (2016). Analysis of radix-2 decimation in time algorithm for fpga co-processors. Dans *2016 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube)*, 154–157. `http://dx.doi.org/10.1109/ICECUBE.2016.7495214`

Intel. (2015). *Intel Math Kernel Library.* `https://software.intel.com/en-us/intel-mkl/`.

Kerner, M. et Amrani, O. (2008). Soft-input hard-output based decoder for turbo codes. Dans *2008 IEEE 25th Convention of Electrical and Electronics Engineers in Israel*, 789–791. `http://dx.doi.org/10.1109/EEEI.2008.4736644`

Kernighan, B. W. (1988). *The C Programming Language* (2nd éd.). Prentice Hall Professional Technical Reference.

Ketonen, J., Juntti, M., Ylioinas, J. et Cavallaro, J. R. (2012). Implementation of ls, mmse and sage channel estimators for mobile mimo-ofdm. Dans *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, 1092–1096.

Kewen, L. et XingKe (2010). Research of mmse and ls channel estimation in ofdm systems. Dans *The 2nd International Conference on Information Science and Engineering*, 2308–2311. `http://dx.doi.org/10.1109/ICISE.2010.5688562`

Laraki, A., El Ouadghiri, D. et Jamali, A. (2013). Channel estimation algorithms for mimo-ofdm systems (lte standard). Dans *Proceedings of International Conference on Advances in Mobile Computing & Multimedia*, MoMM '13, 104 :104–104 :111., New York, NY, USA. ACM.

Li, K., Wu, M., Wang, G. et Cavallaro, J. R. (2014). A high performance GPU-based software-defined base station. Dans *48th Asilomar Conference on Si-*

gnals, Systems and Computers, ACSSC 2014, Pacific Grove, CA, USA, November 2-5, 2014, 2060–2064.

Li, P., Jia, Y.-j., Chen, F., He, P.-h. et Fan, H.-f. (2016). The synchronization design and implementation of lte-advanced real-time test platform based on software defined radio. *International Journal of Smart Home, 10*(5), 149–158.

Ling, F. et Proakis, J. (2017). *Synchronization in Digital Communication Systems.* Cambridge University Press. `http://dx.doi.org/10.1017/9781316335444`

MathWorks. (2015). *Channel Estimation.* `https://www.mathworks.com/help/lte/ug/channel-estimation.html`.

Matlab (2016). *SC-FDMA modulation.* `https://www.mathworks.com/help/lte/ref/ltescfdmamodulate.html`.

Miller, F. P., Vandome, A. F. et McBrewster, J. (2009). *Cyclic Redundancy Check : Computation of CRC, Mathematics of CRC, Error Detection and Correction, Cyclic Code, List of Hash Functions, Parity Bit, Information ... Cksum, Adler-32, Fletcher's Checksum.* Alpha Press.

Myung, H. G. et Goodman, D. J. (2008). *Single Carrier FDMA : A New Air Interface for Long Term Evolution.* Wiley Publishing.

NI (2016). *Introduction to LTE Device Testing. From Theory to Transmitter and Receiver Measurements.* `download.ni.com/evaluation/rf/Introduction_to_LTE_Device_Testing.pdf`.

NVIDIA (2007). *GPU Computing.* `www.nvidia.ca/object/what-is-gpu-computing.html`.

Nvidia (2007). *Kepler Architecture white paper.* `http://www.nvidia.ca/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf`.

Patzold, M. (2003). *Mobile Fading Channels.* New York, NY, USA : John Wiley & Sons, Inc.

Rumney, M. (2008). *Introducing Single-Carrier FDMA.* `https://www.tmit.bme.hu/sites/default/files/LTE_SCFDMA.pdf`.

S., B., C., A., Y, J. et S., C. (2014). Implementation of LTE system on an SDR platform using CUDA and UHD. *Analog Integr. Circuits Signal Process., 78(3)*, 599–610.

Schulz, B. (2015). *LTE Transmission Modes and Beamforming*

TS-23.401 (2017). *3GPP TS 23.401 Version V14.5.0 : Technical Specification Group Services and System Aspects.*

TS-36.141 (2016). *3GPP TS-36.141 : Base Station (BS) conformance testing.*

TS-36.211 (2015). *3GPP TS 36.211 Version 12.7.0 : Physical channels and modulation.*

TS-36.212 (2015). *3GPP TS 36.212 Version 12.6.0 : Multiplexing and channel coding.*

Vieira, J. (2011). Implementation of LTE mini receiver on GPUs. *Institute of Telecommunications.* `http://publik.tuwien.ac.at/files/PubDat_196229.pdf`.

Watson, D. F. (1992). *Contouring : A Guide to the Analysis and Display of Spatial Data.* Oxford : Pergamon Press.

Yang, S. (2010). *OFDMA System Analysis and Design.* Artech House telecommunications library. Artech House. Récupéré de `\url{https://books.google.ca/books?id=TiYxUy7LnRsC}`

Zheng, Q., Chen, Y., Dreslinski, R., Chakrabarti, C.nastasopoulos, A., Mahlke, S. et Mudge, T. (2013). Architecting an LTE base station with graphics processing units. Dans *2013 IEEE Workshop on Signal Processing Systems (SiPS)*, 219–224.

Zyren, J. (2007). Overview of the long term evolution physical layer. `http://www.freescale.com/files/wireless_comm/doc/white_paper/3GPPEVOLUTIONWP.pdf`.