Exploration d'architectures génériques sur FPGA pour des algorithmes d'imagerie multispectrale

Junyan Tan

THESE

Présentée par

Junyan TAN

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE JEAN MONNET

Discipline : microélectronique

Laboratoire Hubert Curien UMR CNRS 5516

Département Informatique Image

Exploration d'architectures génériques sur FPGA pour des algorithmes d'imagerie multispectrale

Soutenance le 12 Juin 2012 devant la commission d'examen

JURY

M. Frédéric Rousseau	Professeur TIMA-UJF	Directeur de thèse
Mme. Virginie Fresse	Maitre de Conférences LaHC-UJM	Co-directrice de thèse
M. Yves Ledru	Professeur LIG-UJF	Président
M. El-Bay Bourennane	Professeur LE2I-Dijon UB	Rapporteur
M. Mohamed Akil	Professeur ESIEE	Rapporteur
M. Fernando Gehm Moraes	Professeur PUC-Brésil	Examinateur
M. Érik Hochapfel	Fondateur/IngénieurADACSYS	Examinateur

Remerciements

Aucun travail ne s'accomplit dans la solitude.

J'ai tout au long de mon doctorat eu le soutien de plusieurs personnes et je tiens à leur exprimer ma gratitude.

Tout d'abord j'aimerais remercier mes deux directeurs de thèse qui m'ont encadrée pendant ces années et sans qui ce travail n'aurait pas pu se faire : monsieur Frédéric Rousseau et madame Virginie Fresse. Ils se sont montrés disponibles, attentifs, ont toujours su orienter mon travail et m'ont prodigué des conseils avisés.

J'aimerais également remercier les membres du jury qui ont accepté de prendre de leur temps pour valider mes travaux et plus particulièrement monsieur El-Bay Bourennane et monsieur Mohamed Akil qui m'ont fait l'honneur d'assumer le rôle difficile de rapporteur.

Je tiens à remercier tous mes collègues du groupe System-Level-Synthesis au laboratoire TIMA à Grenoble, et du groupe Image Processing au laboratoire Hubert Curien à Saint Etienne.

De plus j'aimerais remercier la région Rhône-Alpes pour m'avoir accordé le financement de ses recherches.

Pour terminer, je remercie chaleureusement mes parents, mon mari, mes grands parents, mes amis pour m'avoir soutenue dans les bons et moins bons moments et pour m'avoir donné l'envie d'apprendre.

Résumé

Les architectures multiprocesseur sur puce (MPSoC) basées sur les réseaux sur puce (NoC) constituent une des solutions les plus appropriées pour les applications embarquées temps réel de traitement du signal et de l'image. De part l'augmentation constante de la complexité de ces algorithmes et du type et de la taille des données manipulées, des architectures MPSoC sont nécessaires pour répondre aux contraintes de performance et de portabilité. Mais l'exploration de l'espace de conception de telles architectures devient très coûteuse en temps. En effet, il faut définir principalement le type et le nombre des cœurs de calcul, l'architecture mémoire et le réseau de communication entre tous ces composants. La validation par simulation de haut niveau manque de précision, et la simulation de bas niveau est inadaptée au vu de la taille de l'architecture. L'émulation sur FPGA devient donc inévitable.

Dans le domaine de l'image, l'imagerie spectrale est de plus en plus utilisée car elle permet de multiplier les intervalles spectraux, améliorant la définition de la lumière d'une scène pour permettre un accès à des caractéristiques non visibles à l'œil nu. De nombreux paramètres modifient les caractéristiques de l'algorithme, ce qui influence l'architecture finale.

L'objectif de cette thèse est de proposer une méthode pour dimensionner au plus juste l'architecture matérielle et logicielle d'une application d'imagerie multispectrale. La première étape est le dimensionnement du NoC en fonction du trafic sur le réseau. Le développement automatique d'une plateforme d'émulation sur mono ou multi FPGA facilite cette étape et détermine le positionnement des composants de calcul. Ensuite, le dimensionnement des composants de calcul et leurs fonctionnalités sont validés à l'aide de plateformes de simulation existantes, avant la génération du modèle synthétisable sur FPGA. Le flot de conception est ouvert dans le sens qu'il accepte différents NoC à condition d'avoir le modèle source HDL de ce composant.

De nombreux résultats mettent en avant les paramètres importants qui ont une influence sur les performances des architectures et du NoC en particulier. Plusieurs solutions sont décrites, commentées et critiquées. Ces travaux nous permettent de poser les premiers jalons d'une plateforme d'émulation complète MPSoC à base de NoC.

Abstract

The Multiprocessor-System-On-Chip (MPSoC) architectures based on the Network-On-Chip (NoC) communication are the one of the most appropriate solution for image and signal processing applications under real time constraints. Due to the ever increasing complexity of these algorithms, the types and sizes of the data manipulated, the MPSoC architectures are necessary to meet the constraints of performance and portability. However exploring the design space of such architecture is time consuming. Indeed, many parameters should be defined such as the type and the number of processing cores, the memory architecture and the communication network between all these components. Validation by high-level simulations has the lack of the precision. Low-level simulation is inadequate for such big size of the architecture. Therefore, the emulation on FPGA becomes inevitable.

In image processing, spectral imaging is more and more used. This technology captures light from more frequencies than the human eye increasing the number of wavelengths. Invisible details can be extracted from a scene. The difference between all spectral imaging applications is the number of wavelengths and the precision. Many parameters affect the characteristics of the algorithm, having a huge impact on the final architecture.

The objective of this thesis is to propose a method for sizing one of the most accurate hardware and software architecture for multispectral imaging application. The first step is the design of the NoC based on the network traffic. The automatic development of an emulation platform on a single FPGA or multi-FPGAs simplifies this step and determines the positioning of the computational components. Then, the design of computational components and their functions are validated using existing simulation platforms. The synthesizable model of the architecture on FPGA is then generated. The design flow is open. Several NoC structures can be inserted using the source model of this component.

The set of results obtained points out the major parameters influencing the performances of architecture and the NoC itself. Several solutions are described and analyzed. These studies allow us to lay the groundwork for a complete MPSoC emulation platform based on NoC.

Sommaire

Prologue

Les travaux de recherche menés et présentés dans ce manuscrit ont été effectués dans le cadre d'une bourse région Rhône-Alpes, projet Semba (Systèmes Embarqués). Cette thèse est faite en collaboration entre le laboratoire Hubert Curien Saint Étienne, équipe Image Processing, unité mixte de recherche CNRS 5516 de l'université Jean Monnet, et le laboratoire Technique de l'Informatique et de la Microélectronique pour l'Architecture d'ordinateurs (TIMA) Grenoble, équipe System Level Synthesis (SLS), unité mixte de recherche CNRS 5159, de Grenoble INP (Institut Polytechnique de Grenoble) et de l'Université Joseph Fourier.

Liste des Figures

Liste des Tableaux

Glossaire

NoC	Network-on-Chip
SoC	System on Chip
FPGA	Field Programmable Gate Array
IP	Intellectual Property
CMOS	Complementary Metal Oxide Semiconductor
MPSoC	Multi-Processors System on Chip
GPU	Graphical Processing Unit
GPP	General Purpose Processor
SIMT	Single Instruction Multiple Threads
NI	Network Interface
NA	Network adaptor
FLIT	Flow control unIT
TR	Récepteur de trafic
TG	Générateur de trafic
SISD	Single Instruction, Single Data
MIMD	Multiple Instruction, Multiple Data

Chapitre 1. Introduction

Le nombre de transistors sur puce croissant de manière exponentielle a permis d'augmenter de manière considérable la complexité des systèmes embarqués disponibles. Cette complexité est accrue par l'augmentation de la diversité des composants intégrés, diversité qui devient de plus en plus importante. Les progrès significatifs en termes de conception de circuit permettent dorénavant d'intégrer des composants analogiques ou d'autres types (par exemple mécanique) avec des transistors CMOS. Les performances des circuits ne se limitent plus à la miniaturisation mais à la diversification des composants. Cette diversification est prise en compte dans la loi « More than Moore », loi remplaçant la loi de Moore présentée par l'ITRS et présentée en Figure 1-1[1][2].Cette loi montre que les systèmes embarqués deviennent de plus en plus complexes et hétérogènes. Le nombre de blocs s'accroit ainsi que leurs diversités, ce qui devient une difficulté majeure dans la spécification et conception de ces systèmes. Cette difficulté va s'accroitre de manière significative avec l'augmentation du nombre et de la complexité des cœurs d'un système embarqué.

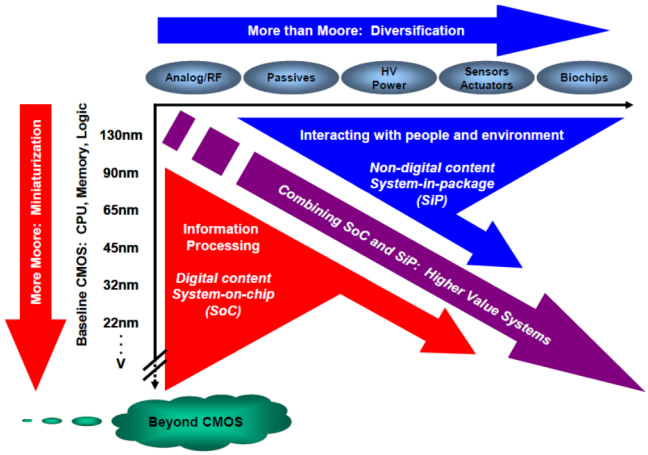


Figure 1-1 : Loi de Moore/ More than Moore : miniaturisation des fonctions numériques contre diversification fonctionnelle.

Les spécifications de la conception de systèmes sur puce (appelé SoC pour System On Chip) ainsi que leur validation sont à l'heure actuelle des challenges importants. L'espace de conception des architectures est très vaste, trop vaste pour qu'un concepteur système puisse définir des architectures adaptées parmi tous les composants de calcul, de mémorisation, de communication et de contrôle pour un ou des algorithmes de traitement du signal et de l'image. Le choix de l'architecture doit se faire dès les premières étapes du flot de conception afin de répondre aux contraintes de « Time To Market ». En effet, les premiers outils de codesign [3][4][5][6], apparus

dans les années 90, ont montré l'impossibilité d'un choix efficace des composants d'une architecture dans les dernières étapes du cycle de conception.

La simulation des systèmes multi composants hétérogènes devient également inadaptée car trop coûteuse en temps pour des simulations post-placement routage et trop loin du fonctionnement réel sur circuit pour les simulations fonctionnelles. La simulation est supplantée par le prototypage rapide et l'émulation qui permettent la vérification, l'exploration et la validation de système appropriées aux contraintes de « Time To Market » actuelles.

Les architectures embarquées conçues pour des applications de traitement de signal et d'images à contraintes temps réel fortes sont des architectures multiprocesseurs à plusieurs dizaines voire centaines de processeurs. On parle respectivement d'architectures « multicores » ou « manycores » (se référant respectivement à des dizaines et centaines de cœurs de calcul sur SoC). Ces cœurs bien souvent hétérogènes sont interconnectés via des architectures de communication de type réseau sur puce (NoC Network On Chip) afin de supporter un nombre élevé de cœurs pour des communications rapides et de faible consommation. L'architecture SoC qui utilise plusieurs processeurs connectés via le NoC est définie comme MPSoC(Multi-Processors System on Chip) [8]. Les circuits programmables notamment les FPGA (Field-Programmable Gate Array) sont utilisés pour l'émulation et le test des architectures, dans de nombreuses applications nécessitant le prototypage rapide d'application d'électronique numérique (télécommunication, traitement d'images, l'aéronautique…). Les circuits les plus évolués peuvent intégrer des cœurs de processeurs, logiciels ou matériels, des opérateurs câblés dédiés (appelés également blocs IP pour Intellectual Property) pour effectuer efficacement le prototypage de systèmes embarqués. A l'heure actuelle, un seul circuit FPGA ne possède pas assez de ressources pour supporter une architecture SoC complète. Ces architectures ont besoin d'être partitionnées sur plusieurs circuits reconfigurables, en général sur les plateformes multi-FPGA pour l'émulation et le test.

Dans le domaine de l'image, les applications en imagerie spectrale se substituent de plus en plus à l'imagerie couleur. L'imagerie spectrale permet de multiplier les intervalles spectraux d'intégration, améliorant la définition de la lumière d'une scène et un accès à des caractéristiques non visibles à l'œil nu. Ces algorithmes deviennent de plus en plus complexes, intégrant des données de type de plus en plus variés (avec des représentations et tailles variables) et augmentent le nombre de fonctions de calcul dans les dimensions spatiales et spectrales. Les contraintes temps réel de ces applications nécessitent des architectures dédiées telles des MPSoC à base de NoC présentées précédemment.

Il est donc nécessaire de pouvoir proposer des architectures embarquées hautes performances, dédiées aux algorithmes d'imagerie spectrale. Les architectures proposées doivent être paramétrables pour accueillir plusieurs applications d'imagerie spectrale sans repasser par le flot complet de la conception de l'architecture. Le choix des paramètres de l'architecture proposée doit se faire rapidement par le concepteur logiciel, sans pré-requis matériel et avec des solutions de vérification et d'émulation adaptées et disponibles immédiatement.

1.1 Contribution

La contribution de ce travail de thèse est de proposer un flot de conception, d'émulation et de validation d'architecture MPSoC basées sur un NoC générique. Cette architecture est dédiée aux applications d'imagerie spectrale, notamment pour des applications d'authentification d'œuvres d'art. Le flot de conception proposé permet de générer une plateforme mono ou multi-FPGA et facilite l'exploration de l'espace de conception et l'analyse des performances.

Il est apparu au cours de ces travaux qu'une des contraintes majeures provenait des échanges de données entre les éléments de calcul. Aussi un travail plus approfondi sur l'évaluation des performances et l'amélioration des caractéristiques d'un NoC générique est réalisé. Une contribution plus spécifique consiste à définir les blocs d'émulation adaptés à l'évaluation et à l'exploration d'architecture. Une bibliothèque de générateurs et de récepteurs de trafic développés en VHDL permet à l'utilisateur de sélectionner les blocs d'émulation adaptés à ses besoins. Les générateurs permettent de simuler tous types de trafic, d'un émetteur ou de plusieurs émetteurs vers un ou plusieurs récepteurs ciblés en utilisant des tailles de paquets variables. De la même manière, les récepteurs permettent de récupérer des informations de performances (telle la latence d'un flit, d'un paquet) ou d'extraire les données du paquet. Les mesures de performances tiennent compte des éléments additionnels nécessaires aux communications entre FPGA pour une plateforme multi-FPGA.

1.2 Plan du manuscrit

Ce manuscrit, organisé en quatre chapitres, présente l'étude et l'exploration d'une architecture générique sur FPGA pour des algorithmes d'imagerie multispectrale. Cette architecture est destinée à des équipements embarqués et portables, nécessitant de supporter des applications variées d'imagerie spectrale.

1.2.1 Applications et principes de base des architectures

Le premier chapitre de ce manuscrit présente les algorithmes d'imagerie multispectrale ainsi qu'une description succincte des architectures de communication pour les systèmes embarqués appelés réseaux de communication sur puce. Ce chapitre présente dans un premiers temps le principe de l'imagerie spectrale puis décrit une application dédiée à l'authentification d'œuvres d'art en imagerie multispectrale. Nous présentons ensuite deux solutions d'architectures parallèles standard permettant d'accélérer éventuellement les performances de calcul de cet algorithme : les architectures PC multicoeurs et les architectures GPU (Graphical Processing Unit). Nous analysons les performances c'est à dire les forces et faiblesses de chacune de ces architectures. Ensuite, nous effectuons une étude du réseau de communication sur les systèmes sur puce (SoC) et systèmes multiprocesseurs sur puce (MPSoC), ces architectures constituant les plateformes de développement du travail de thèse.

1.2.2 Proposition d'une méthodologie pour la conception et l'exploration de NoC

Après une étude des architectures d'émulation des réseaux de communication sur puce NoC, ce deuxième chapitre aborde la proposition de notre méthodologie pour la conception d'une plateforme d'émulation de NoC dédiée à la validation et l'exploration de l'espace de conception des NoCs. Cette nouvelle méthodologie, basée sur diverses bibliothèques développées et insérées dans le flot de conception, permet la génération des architectures adaptables à des plateformes mono-FPGA ou multi-FPGA afin d'évaluer efficacement les performances du NoC et d'explorer les architectures de NoC les plus adaptées aux besoins de l'application ciblée.

1.2.3 Implémentation et évaluation des NoC pour l'évaluation et l'exploration de l'espace de conception

Le troisième chapitre de ce manuscrit cible l'implémentation d'une plateforme générée par la méthodologie proposée dans le chapitre précédent. Cette plateforme d'émulation permet l'évaluation des performances mais également l'exploration de l'espace de conception de l'architecture, ceci grâce au développement de composants génériques paramétrables disponibles dans les bibliothèques associées au flot. Tout d'abord, nous proposons un ensemble de blocs d'émulation contenant les générateurs de trafic et les récepteurs de trafic. Nous présentons l'architecture du NoC Hermes utilisée dans la conception de la plateforme d'émulation. Cette dernière est implémentée sur FPGA Xilinx afin d'étudier des performances et définir les limitations d'une telle architecture. Ainsi un flot similaire ciblant des architectures multi-FPGA est proposé offrant ainsi une extensibilité de la plateforme d'émulation à des plateformes multi-composants. L'analyse des performances est étudiée en fin de chapitre.

1.2.4 Expérimentation : Implantation d'algorithmes d'imagerie multi-spectrale

Le dernier chapitre de ce manuscrit présente une solution d'une architecture MPSoC obtenue à l'aide de notre flot de conception. L'architecture est composée d'un ensemble de processeurs Plasma autour du NoC Hermes. Nous avons utilisé l'outil HeMPS, développé par l'université PUCRS qui permet de générer une plateforme de simulation ou d'émulation sur FPGA. Ceci nous permet de comparer différentes architectures et de valider nos travaux sur l'implantation MPSoC basée sur un NoC pour l'algorithme d'imagerie multi-spectrale.

Le manuscrit se termine par une conclusion générale ainsi que des perspectives.

Chapitre 2. Applications et principe de base des architectures

2.1 Imagerie couleur et multispectrale

2.1.1 Imagerie couleur

L'imagerie couleur est une image numérique qui contient des informations de couleur pour chaque pixel [8].Un espace de couleurs associe des nombres aux couleurs visibles. Compte tenu des limites de la vision humaine, ces nombres se présentent généralement sous la forme de triplets. Chaque couleur de lumière peut donc être caractérisée par un point dans un espace à trois dimensions. Il existe plusieurs espaces couleur (RGB, XYZ, L*a*b …), les espaces les plus utilisés étant présentés ci-dessous.

2.1.1.1 L'espace de couleur RGB

C'est l'espace de base, supporté nativement par la plupart des cartes vidéo. En combinant les trois primitives RGB (respectivement Red, Green et Blue pour rouge vert et bleu), il est possible d'obtenir, ou presque, toutes les couleurs du visible.

Le mélange des pigments de peinture permet d'obtenir toutes les couleurs possibles par filtrage de la lumière en retirant des composantes, et en ne réfléchissant que les couleurs désirées [10]. Ceci est appelé un système soustractif. Le système de couleur RGB est un système additif car la lumière est ajoutée. Ainsi dosées, les trois composantes permettent de représenter un grand nombre de couleurs, mais certaines couleurs visibles par l'œil n'ont pas de représentation dans cet espace.

Chaque axe du cube, de la Figure 2-1représente des valeurs de rouge, de vert, ou bleu dans l'intervalle [0.255]. L'axe rouge, étiqueté R, montre l'échelle de couleurs associée. Les axes vert et bleu sont illustrés de la même manière. La valeur 0 signifie que le moniteur n'émet pas la couleur primaire et la valeur 255 signifie qu'il émet le maximum. Les valeurs entre 0 et 255 représentent les gradations dans l'intensité de la couleur.

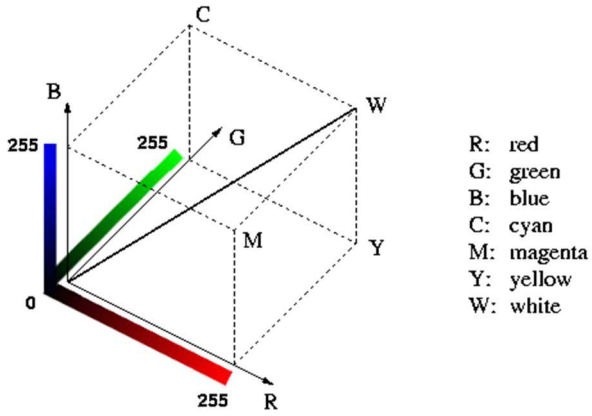


Figure 2-1 : Le cube de couleurs RGB [11]

Comme l'indique l'espace de couleur, les couleurs agissent comme des vecteurs. Ces couleurs peuvent être combinées par addition et soustraction pour obtenir d'autres couleurs dans le cube. Ainsi, l'origine du cube, ou 0^R 0^G 0^B, représente l'absence totale de couleur, c'est à dire le Noir. L'extrémité la plus éloignée de l'origine est la somme des intensités les plus élevées de rouge, de vert, et de bleu, ou 255^R 255^G 255^B. Ceci correspond à la couleur du Blanc (étiquetée par W sur la Figure 2-1). Les autres coins du cube représentent les diverses couleurs primaires et secondaires. Nous avons déjà rencontré le rouge, le vert, et le bleu. Les trois couleurs restantes sont cyan, magenta, et jaune.

2.1.1.2 L'espace de couleur XYZ

L'espace couleur CIE (Commission Internationales de l'Eclairage) XYZ a été défini afin de corriger certains défauts présents dans l'espace RGB. Cet espace résulte les travaux de Judd [12] et est constitué de trois primaires X, Y et Z, dites virtuelles. Ainsi, l'espace couleur XYZ présente les propriétés suivantes :

- Les triplets décrivant chaque couleur en fonction de ses primaires ont tous des valeurs positives pour les spectres visibles (Figure 2-2);
- La fonction Y(λ) représente approximativement la sensibilité de l'œil humain à la luminosité. Par conséquent, la composante Y est habituellement considérée comme la composante luminance du spectre incident ;
- Chaque spectre d'égale énergie est associé à un triplet dont toutes les composantes sont égales [12].

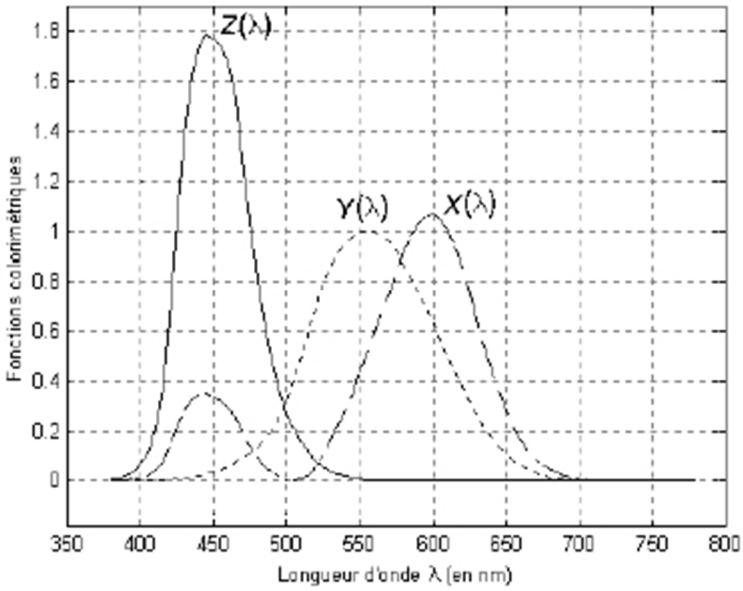


Figure 2-2 : Les fonctions colorimétriques X(λ), Y(λ) et Z(λ)

Le passage de l'espace RGB à l'espace XYZ s'effectue simplement grâce à une transformation linéaire pouvant être interprétée comme un changement de base. L'équation de transformation d'espace est la suivante [13] :

$$\begin{bmatrix} +0.431 & +0.342 & +0.178 \\ -0.222 & +0.707 & +0.071 \\ +0.020 & -0.130 & -0.939 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad \textbf{Équation 2-1}$$

L'équation de passage de l'espace XYZ à RGB est la suivante:

$$\begin{bmatrix} +3.059 & -1.393 & -0.475 \\ -0.968 & +1.875 & +0.042 \\ +0.069 & -0.230 & +1.069 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad \textbf{Équation 2-2}$$

*2.1.1.3 L'espace CIE L*a*b**

L'espace CIE L*a*b* est un modèle de représentation des couleurs développé en 1976 par la CIE. Comme tous les systèmes issus du système CIE XYZ, il caractérise une couleur à l'aide d'un paramètre d'intensité correspondant à la luminance et de deux paramètres de chrominance qui décrivent la couleur. Il a été spécialement étudié pour que les distances calculées entre couleurs correspondent aux différences perçues par l'œil humain.

- La combinaison **L*** est la clarté, allant de 0 (noir) à 100 (blanc).
- La composante **a*** représente la gamme de l'axe rouge (valeur positive) ->vert (négative) en passant par le blanc (0) si la clarté vaut 100.

- La composante **b*** représente la gamme de l'axe jaune (valeur positive) -> bleu (négative) en passant par le blanc (0) si la clarté vaut 100.

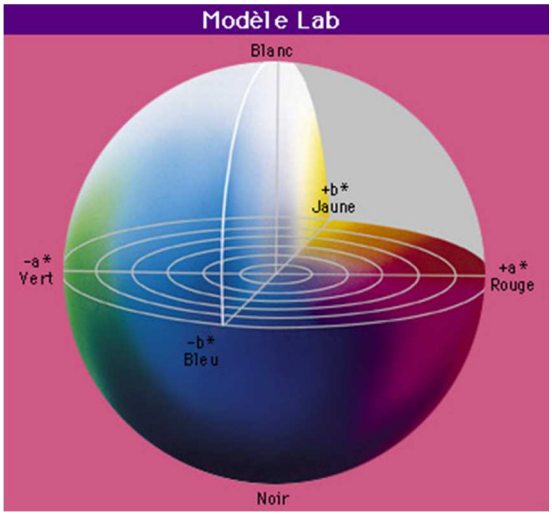


Figure 2-3 : Modèle de représentation dans l'espace couleur L*a*b* [14]

Une difficulté majeure de ce système est l'utilisation d'un système mixte de repérage des points de couleur. La saturation est mesurée de manière cartésienne, alors que la teinte et la luminosité sont mesurées de manière angulaire (Figure 2-3).

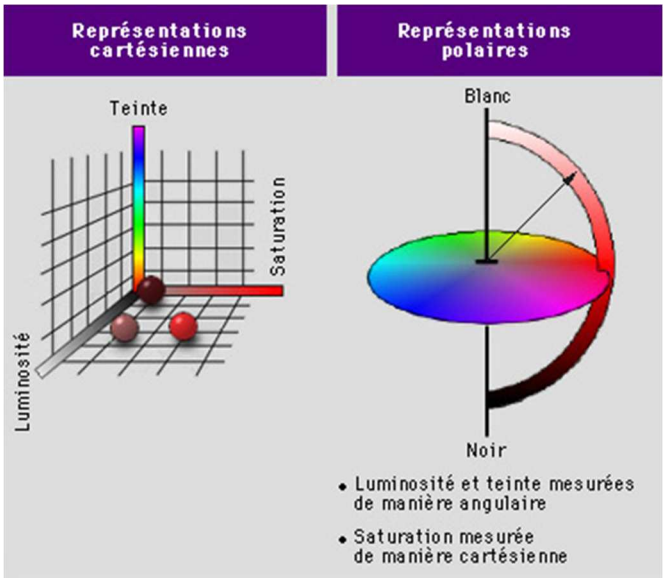


Figure 2-4 : Représentations cartésiennes et polaires [14]

La conversion de l'espace RGB vers l'espace $L^*a^*b^*$ nécessite une conversion intermédiaire dans l'espace de couleur XYZ. Les équations permettant de passer par l'espace intermédiaire sont les suivantes [14]:

$$x = x_{max}\left(p + \frac{a^*}{500}\right)^3 \qquad y = y_{max} * p^3 \quad z = z_{max}\left(p - \frac{b^*}{200}\right)^3 \qquad \text{Où } p = \frac{l^*+16}{116}$$

Équation 2-3

et l'inverse est donné par :

$$L = 116\left(\left(\frac{y}{y_{max}}\right)^{\frac{1}{3}}\right) - 16 \qquad a^* = 500\left(\left(\frac{x}{x_{max}}\right)^{\frac{1}{3}} - \left(\frac{y}{y_{max}}\right)^{\frac{1}{3}}\right) b^* = 200\left(\left(\frac{y}{y_{max}}\right)^{\frac{1}{3}} - \left(\frac{z}{z_{max}}\right)^{\frac{1}{3}}\right)$$

Équation 2-4

2.1.2 Imagerie Multispectrale

L'imagerie multispectrale est une technique développée à l'origine pour le traitement de l'image dans le spectre. En imagerie couleur classique, on acquiert 3 images. En imagerie multispectrale on acquiert beaucoup plus d'images, chaque image correspondant à une bande très étroite du spectre. On a ainsi une définition beaucoup plus précise de la lumière réfléchie par une surface et on peut ainsi accéder à des caractéristiques non visibles à l'œil nu. Les domaines d'utilisation de l'imagerie sont de plus en plus variés. De ce fait les applications traditionnellement en couleur s'orientent vers l'imagerie spectrale (multispectrale ou hyperspectrale).

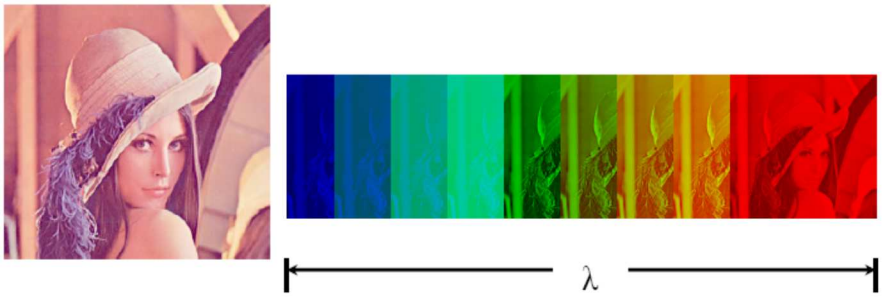


Figure2-5 : Imagerie multispectrale.

La Figure2-5représente une image multispectrale. Les images spectrales obtenues à différentes longueurs d'ondes sont extraites d'une scène, le nombre de longueurs d'onde pouvant couvrir le domaine du visible mais également au-delà. Ainsi le nombre d'images spectrales constituant l'image multispectrale peut être élevée. Les bandes spectrales peuvent être choisies en fonction de longueurs d'ondes caractéristiques des matières ou des produits à analyser.

2.2 Processus d'authentification en Imagerie Multispectrale

Une des applications possible des algorithmes d'imagerie multispectrale est l'authentification d'objets. Le domaine applicatif choisi concerne l'authentification d'œuvres d'art, notamment des

peintures. L'objectif est de déterminer si l'œuvre d'art présente est l'originale ou une copie. Pour le processus d'authentification d'œuvres d'art, nous comparons une œuvre mise à notre disposition avec un ensemble d'informations caractérisant l'œuvre de référence (œuvre originale). Les informations relatives à l'image originale correspondent à des régions significatives. Ainsi les données d'entrée sont :

- OR (Original Region) : sont les régions significatives de l'œuvre de référence dans les différentes longueurs d'ondes. Ces données sont enregistrées dans une base de données.
- CR (Compared Region) : sont les régions significatives de l'œuvre à authentifier dans les différentes longueurs d'ondes. Ces régions proviennent d'une caméra spectrale adaptée pour l'acquisition des régions de l'œuvre à différentes longueurs d'onde.

2.2.1 Principe de l'algorithme d'authentification pour les œuvres d'arts

L'algorithme d'authentification s'effectue dans des espaces couleurs puis en multispectral si nécessaire, plus précisément si le résultat de l'authentification indique que l'œuvre à authentifier est l'originale (dans le cas d'une fausse authentification, les analyses dans le domaine multispectrale ne sont pas conduites). L'algorigramme de l'application proposé est donné en Figure 2-6.

L'algorithme d'authentification se base sur deux étapes principales : une comparaison entre la OR et la CR dans certains espaces de couleur (espace spectral de base, l'espace RGB, l'espace XYZ, l'espace L*a*b, etc.) puis une comparaison entre la OR et la CR dans l'espace spectral. Le processus d'authentification se décompose en cinq étapes:

1) Moyenne spatiale des régions : les régions provenant de la base de données et de la caméra sont moyennées pour chaque région et pour chaque longueur d'onde.

2) Projection couleur : selon l'espace de couleurs de référence choisi, les données originales de spectres de la OR / CR sont transformées en données couleurs.

3) Authentification couleur : le résultat R1 correspondant à la distance couleur entre La région originale et celle à comparer. Ce résultat est obtenu en choisissant au préalable un calcul de distance approprié. Une comparaison entre le résultat R1 et la précision P1 (définie au préalable en fonction de l'expérimentation) est effectuée. Si le résultat de la comparaison correspond, le processus d'authentification est affiné par un passage dans l'espace multispectral (étape 3). Si R1 est différent de P1 (la comparaison ne correspond pas), l'OR et le CR sont différents, le processus s'arrête car l'objet à authentifier est faux.

4) Authentification multispectrale : le résultat R2 correspondant à la distance dans l'espace spectral est calculé en choisissant un calcul de distance spectral. Une comparaison entre le résultat R2 et la précision P2 (définie au préalable) est effectuée. Si le résultat de la comparaison correspond, le processus d'authentification multispectrale peut être affiné en augmentant le nombre de longueurs d'ondes utilisé. Si une plus grande précision n'est pas nécessaire, le résultat de l'authentification est fait dans l'étape 4 afin de définir si l'objet est authentifié ou non.

5) Résultat de l'authentification : si le résultat indique une différence significative entre une précision (P2, P3 …) et R2, cela signifie que l'œuvre d'art n'est pas l'originale. Sinon, l'œuvre d'art est bien l'originale.

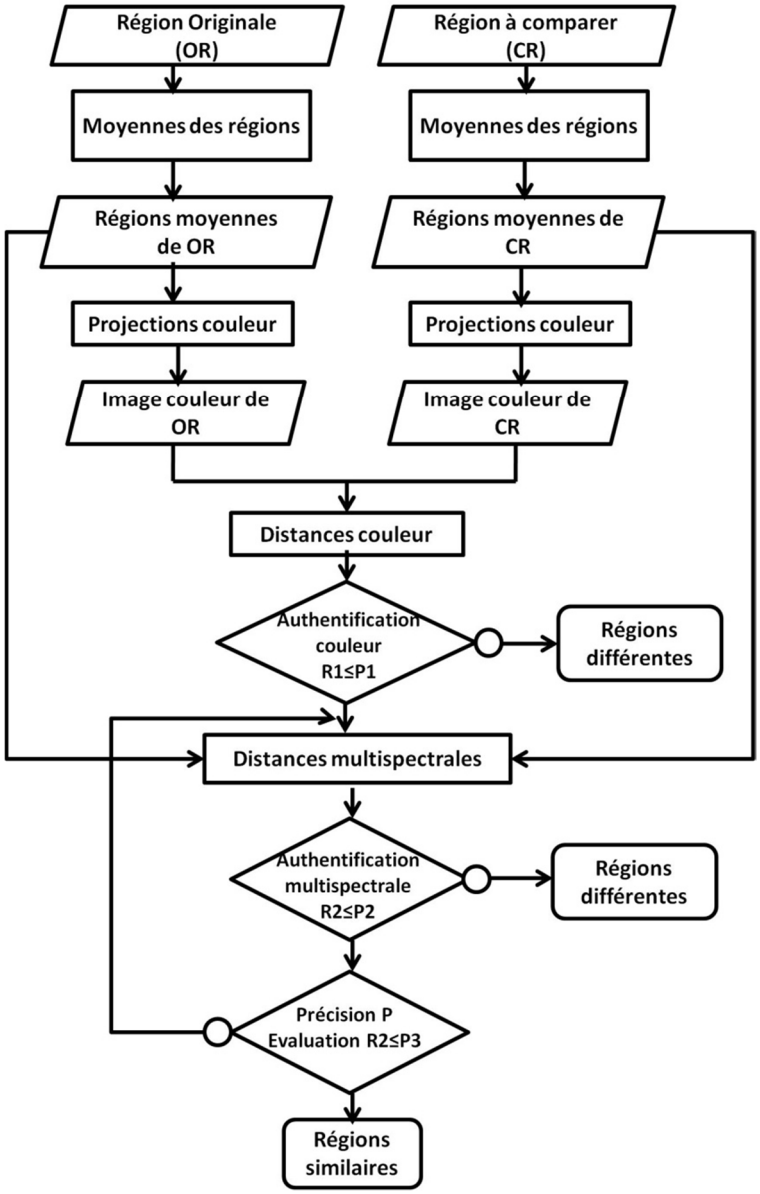


Figure 2-6 : Processus d'authentification d'œuvres d'art en imagerie spectrale.

Les transformations dans les différents espaces couleurs ayant été présentées dans le paragraphe précédent, la présentation de l'algorithme porte sur les calculs de distance couleur et spectral. Les calculs de distances choisis sont ceux les plus couramment utilisés dans les applications d'authentification et répertoriés en littérature.

2.2.2 Calculs de distance dans des espaces couleurs

2.2.2.1 Goodness of Fit Coefficient : GFC

Le calcul GFC est basé sur l'inégalité de Schwartz et est décrit par l'équation suivante [15] :

$$GFC = \frac{|\sum_j R_m(\lambda_j) R_e(\lambda_j)|}{\sqrt{\sum_j [R_m(\lambda_j)]^2} \sqrt{\sum_j [R_e(\lambda_j)]^2}} \qquad \textbf{Équation 2-5}$$

Où $R_m(\lambda_j)$est la donnée spectrale d'origine à la longueur d'onde λ_j, $R_e(\lambda_j)$est la donnée spectrale d'estimation à la longueur d'onde λ_j.

Le résultat GFC doit être compris entre 0,999 et 0,9999 pour considérer que l'on obtient respectivement une bonne et une excellente correspondance spectrale.

2.2.2.2 Viggiano Index Mv

Cette méthode est basée sur la somme des valeurs ΔE^* d'une longueur d'onde entre les deux spectres, dans lequel ils diffèrent seulement d'une longueur d'onde particulière. En pratique, une approximation linéaire de ces ΔE^* est utilisée. Le résultat est une somme pondérée des valeurs absolues des différences entre les deux spectres [16]. L'indice de comparaison spectrale [16]est calculé comme suit :

$$M_v = \sum_{\lambda=1}^{n} w(\lambda) \|\Delta\beta(\lambda)\| \qquad \textbf{Équation 2-6}$$

Où $\Delta\beta(\lambda)$ est la différence entre les deux spectres. $w(\lambda)$ est défini suivant l'équation :

$$w(\lambda) = \sqrt{\left(\frac{dL^*}{d\beta(\lambda)}\right)^2 + \left(\frac{da^*}{d\beta(\lambda)}\right)^2 + \left(\frac{db^*}{d\beta(\lambda)}\right)^2} \qquad \textbf{Équation 2-7}$$

2.2.3 Calculs de distance dans l'espace spectral

Les calculs de distances choisis pour des images multispectrales sont RMS et WRMS.

2.2.3.1 Root Mean Square Error: RMS

La fonction RMS, Root Mean Square Error est calculée au moyen de l'équation suivante :

$$RMS = \sqrt{\frac{1}{N}\sum_j \left\| S_{1\lambda j} - S_{2\lambda j} \right\|^2} \qquad \textbf{Équation 2-8}$$

Où S correspond au spectre à calculer, **λ** la longueur d'onde et **N** le nombre de longueur d'ondes.

Cette méthode de calcul de distance spectral est la plus utilisée, car facile à calculer. Cependant elle ne considère pas la vision humaine [17].

2.2.3.2 WRMS (Inverse Reflectance)

Le calcul WRMS considère un ensemble de poids associé à chaque longueur d'onde contrairement au calcul RMS. La distance spectrale est obtenue au moyen de l'équation :

$$wrms = \sqrt{\frac{\sum_{\lambda=1}^{n}\left(\sqrt{w(\lambda)\Delta\beta(\lambda)}\right)^2}{n}} \qquad \textbf{Équation 2-9}$$

Où $w(\lambda)$ est le poids, n est le nombre de longueurs d'ondes, $\Delta\beta(\lambda)$ est la différence entre les deux spectres [19].

Cette méthode utilise différents poids pour les couleurs foncées et les couleurs claires.

2.2.4 Conclusion

La description de l'algorithme et des fonctions utilisées montre la nécessité de concevoir une architecture matérielle dédiée qui réponde à plusieurs contraintes :

- Il est nécessaire d'avoir des puissances de calcul importantes pour répondre à des contraintes temps réel fortes,
- Les structures de ces blocs doivent s'adapter aux paramètres variables de l'application, à l'ajout de nouvelles fonctions, à la modification de la représentation du flot de donnée de l'application,
- Les capacités de communication doivent être performantes et adaptables,
- L'architecture doit pouvoir considérer des formats de données variables et multiples pour les communications et les opérations de calcul, pour répondre à la méthodologie AAA (Adéquation Algorithme Architecture).

2.3 Les architectures programmables embarquées pour l'algorithme d'authentification

Il est possible d'implanter l'algorithme précédemment présenté sur plusieurs architectures programmables comme un processeur classique (CPU) ou des processeurs graphiques(GPU). L'objectif est de montrer les limitations et les avantages de ces architectures par rapport aux architectures MPSoC sur FPGA.

Dans la famille des solutions programmables dans le domaine de l'embarqué, nous distinguons les processeurs généralistes et les processeurs spécialisés embarqués.

2.3.1 Processeur généraliste dans l'embarqué : AMD Athlon 64

Un processeur généraliste, appelé « General Purpose Processor » (GPP), permet la plus grande versatilité en termes de calcul. Dans le domaine de l'embarqué, ce type de processeur est généralement utilisé pour du contrôle global ou des applications de haut niveau.

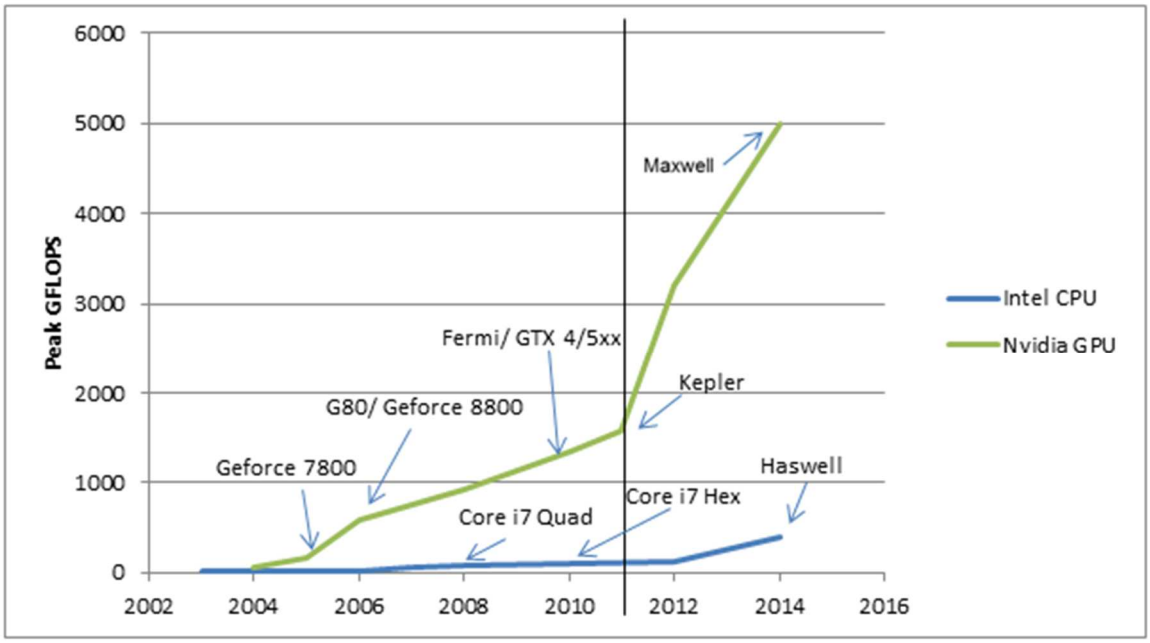
L'AMD Athlon 64 est le processeur généraliste utilisé pour évaluer l'algorithme d'authentification. C'est un microprocesseur 64 bits basé sur l'architecture K8 et conçu pour les ordinateurs de bureau. Ces microprocesseurs comprennent toutes les fonctions de base de micro-architecture K8. Nous pouvons citer le découpage 128 Ko de cache niveau 1, le cache niveau 2 exclusif 512 Ko ou 1 Mo, la technologie AMD64, Enhanced Virus Protection, et le jeu d'instructions spécifiques Intel SSE2 (Streaming SIMD Extensions). Le jeu d'instructions SSE3 a été ajouté sur tous les processeurs Athlon 64 bits.

La stratégie des architectures CPU, présentée en Figure 2-9, est d'exécuter une tâche la plus rapidement possible. Ces stratégies utilisent la mémoire cache, le prefetching des instructions et l'exécution spéculative. Les architectures sont de type SISD ou MIMD pour des CPU double ou quadri-cœurs.

2.3.2 Processeur spécialisé : GPU

Les processeurs spécialisés sont conçus et adaptés pour certains types de calculs. Lorsque le type d'application est ciblé, il s'agit de concevoir des unités fonctionnelles spécifiques adaptées aux besoins de l'algorithme dans le processeur. La conception d'une telle architecture dédiée permet d'accélérer les temps de calcul.

Un GPU (Graphics Processing Unit) est un microprocesseur présent sur les cartes graphiques au sein d'un ordinateur ou d'une console de jeux vidéo, et est dédié à l'accélération graphique 3D. Dans les années passées, les puissances de calcul des GPU programmables ont progressé de manière significative, comme représenté dans la Figure 2-7.



	Peak GFLOPS	Year		Peak GFLOPS	Year
Intel Pentium 4	7	2003	Nvidia Geforce 6800	54	2004
Intel Pentium D	13	2005	Nvidia Geforce 7800	165	2005
Intel Core 2 Duo	23	2006	Nvidia Geforce 8800	576	2006
Intel Core 2 Quad	51	2007	Nvidia Geforce GTX 280	933	2008
Intel Core i7 Quad	70	2008	Nvidia Geforce GTX 480	1,350	2010
Intel Core i7 Hex	109	2010	Nvidia Geforce GTX 580	1,580	2011
Intel Ivy Bridge	130	2012	Nvidia Kepler	3,200	2012
Intel Haswell	400	2014	Nvidia Maxwell	5,000	2014

Figure 2-7 : Evolution de processeurs généralistes / GPU[20]

Le GPU est particulièrement bien adapté au problème d'adressage qui peut être exprimé pour des calculs de données parallèles. Un même programme est exécuté pour chaque donnée, le besoin en flot de contrôle est faible. Dans un GPU, la latence d'un accès à la mémoire peut être cachée par le calcul au lieu d'utiliser les grandes mémoires caches de données.

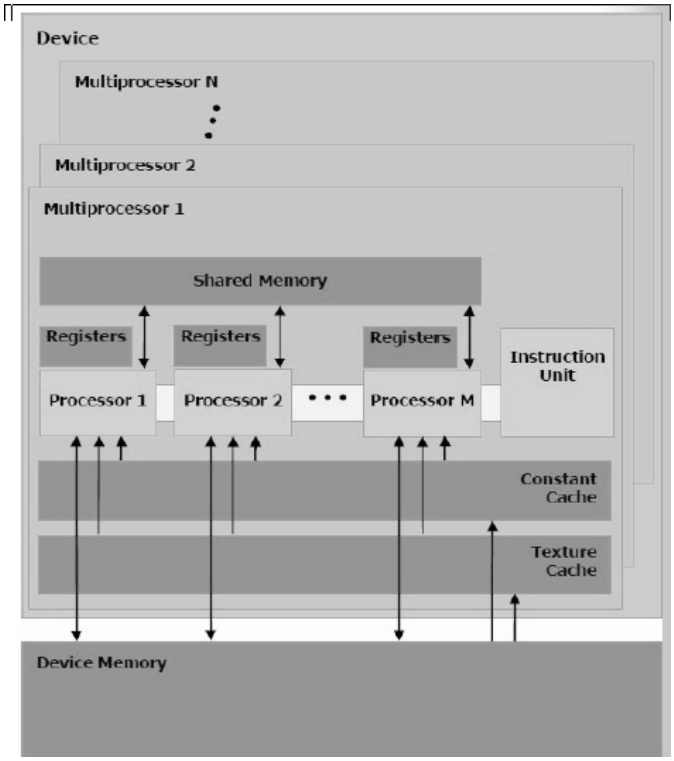


Figure 2-8 : Architecture GPU NVIDIA Geforce [22]

Les GPUs les plus courants présentent des solutions efficaces pour l'implantation d'algorithmes graphiques, des applications 3D telles que la reconstruction en tomographie en raison de leur haut niveau de parallélisme. De plus, les GPUs NVIDIA sont efficacement et facilement programmables dans l'environnement CUDA. La famille de NVIDIA Geforce GT200 (Figure 2-8) a au maximum 30 multiprocesseurs (MP) fonctionnant en streaming. Chaque multiprocesseur dispose de 8 processeurs pour notre architecture. Les MP fonctionnent de manière asynchrone et sans possibilité de communication entre eux. Chaque MP travaille en mode SIMD, ce qui signifie que chaque processeur d'un même MP exécute la même instruction en utilisant le même type de donnée. Un code non-incrémental est parallélisé afin d'exécuter efficacement sur les 1 × 8 multi-threaded stream processeurs. Les threads sont regroupés en blocs (32x1 threads dans notre cas) qui sont définis au moment de l'exécution d'un bloc pour chaque multiprocesseur. Chaque couple de multiprocesseurs est associé à une mémoire cache de texture2D et une mémoire de données. Les processeurs disposent de quelques registres. De plus, le GPU offre une grande bande passante mémoire (BWmem = 112 Go / s dans notre cas) et utilise la virgule flottante pour les calculs. Toutceci permet de paralléliser efficacement les boucles.

Le GPU est programmable en utilisant le langage C standard avec quelques extensions sans aucune connaissance du pipeline graphique.

La stratégie des architectures GPU est l'exécution d'un nombre important de tâches en parallèle (Figure 2-9). Le parallélisme de donnée peut couvrir jusqu'à 1000 tâches et le pipeline est exploité. Les GPU supportent le multitâche massif et cachent les latences avec du calcul et non pas des mémoires caches. L'architecture GPU est de type SIMT (Single Instruction Multiple Threads) avec une structure de parallélisme hiérarchique.

Figure 2-9 : Stratégie des architectures CPU et GPU.

2.4 Performances sur CPU et GPU

Les spécifications suivantes sont choisies pour l'algorithme présenté dans le chapitre précédent dans le cas de l'authentification d'une peinture :

- Nombre de régions = 2 000.
- Nombre de longueurs d'onde pour chaque région = 128, 480, 960, 992.
- Tailles de la fenêtre (pour la moyenne) =1x1, 2x2, 4x4, 8x8, 16x16, 32x32, 64x64 pixels.

L'algorithme paramétrable est d'abord exécuté sur une nouvelle génération de PC dual-core, ensuite sur une génération de GPU GTX280 afin d'obtenir différentes performances.

2.4.1 Exécutions sur CPU

L'algorithme d'authentification est exécuté sans optimisations, à partir de fonctions C décrivant chaque nœud de l'application. Les spécifications du CPU sont les suivantes :

- AMD Athlon64x2 Dual Core Processeur 4800+ 2,5GHz,
- 2Go de RAM
- L'accès aux régions et aux spectres se font de manière identique pour les systèmes embarqués visés. Ainsi le chargement de chaque région ou de chaque spectre est fait de manière successive à partir d'une mémoire globale.

Tableau 2-1 : Temps d'exécution sur CPU (pour une région)

	Total	Moyenne (64x64)	Accès de fichier	Distance multispectrale
Temps d'exécution (sec)	5.654	0.234	5.42	0.05

- Le temps d'exécution sur un CPU est environ de 188 minutes (soit 2h30) pour 2000 régions avec des accès de fichiers sur disque dur standard (lectures de coefficients, des régions pour CR et OR).
- Les accès au fichier correspondent à la majeure partie du temps d'exécution avec un temps de lecture et d'écriture de 5.42 s. Ce temps est relativement long car il est lié aux performances de la mémoire utilisée dans le PC d'expérimentation. L'accès aux fichiers pour les 2000 régions pourrait se faire beaucoup plus vite avec l'utilisation de mémoires externes rapides. Par exemple, on peut espérer atteindre environ 10 minutes au lieu de 2h30 pour un disque dur de 250 Mo / s d'accès en pipeline.
- Le calcul de la moyenne constitue la fonction la plus gourmande en temps de calcul. Des optimisations de ce temps de calcul sont difficilement envisageables, les régions étant distinctes les unes des autres.

2.4.2 Exécutions sur GPU (Graphics Processing Unit)

La plateforme de GPU utilisée dans notre cas est GTX280 Nvidia Geforce. Les spécifications sont les suivantes :

- 240 stream processeurs
- Core clock 576MHz
- 1GB de mémoireDDR3

Les techniques d'accélération utilisées pour l'algorithme sont basées sur le parallélisme de données. Cette technique décompose les longueurs d'onde sur chaque processeur pour réaliser du multitâche. Le nombre de longueurs d'onde est choisi en fonction du nombre de processeurs de chaque MP. Les processeurs du MP exécutent une fonction et renvoient le résultat vers un processeur pour le calcul final de la fonction à exécuter. L'implantation de l'algorithme se résume donc à l'exécution d'une fonction en parallèle puis de la même fonction en séquentiel pour les calculs de moyenne, les projections dans l'espace X, l'espace Y et Z ainsi que les calculs de distance couleur et de distance multispectrale.

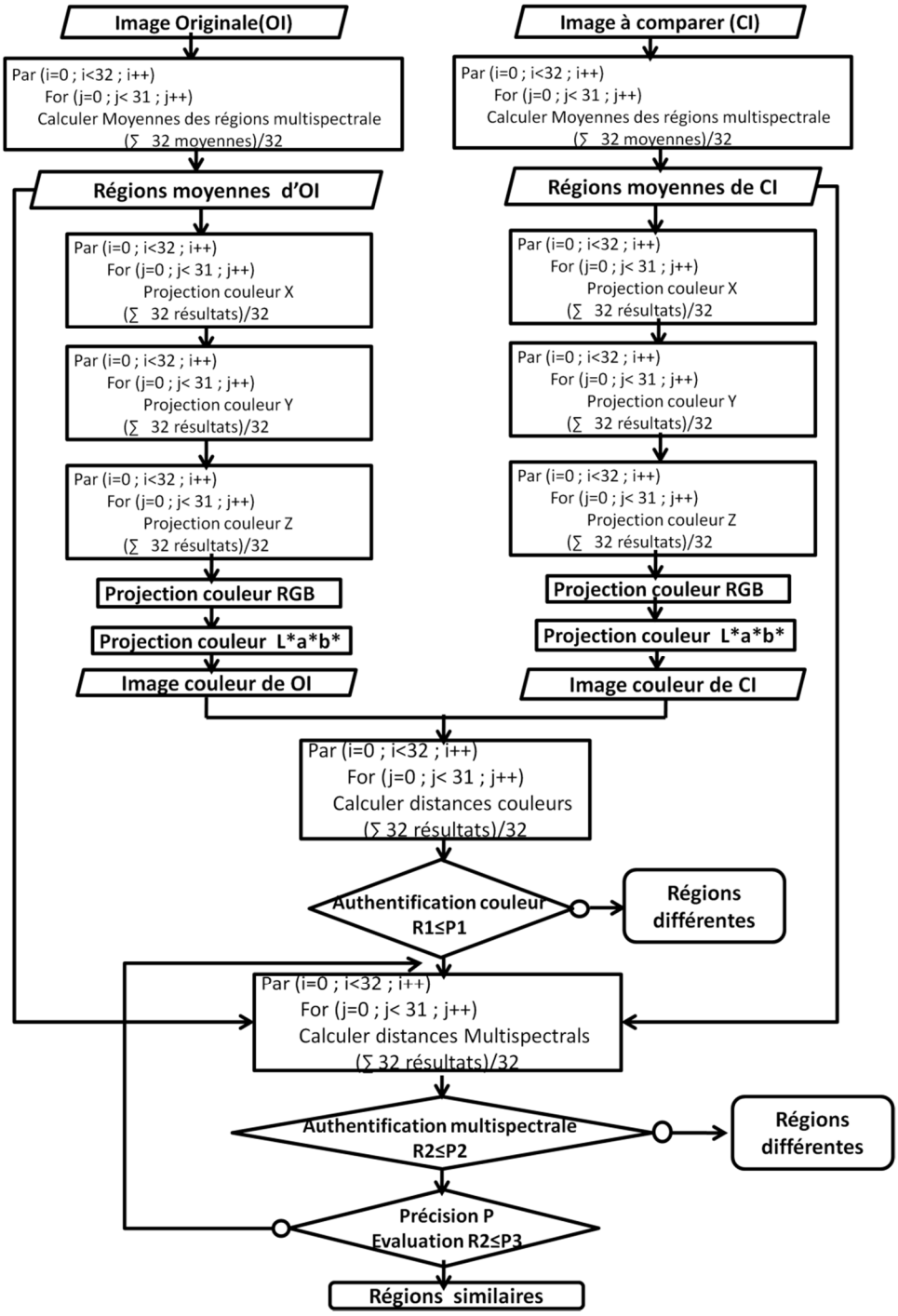


Figure 2-10 : Algorithme d'authentification parallélisé sur une architecture GPU à 16 processeurs par MP.

L'algorithme proposé pour une implantation sur GPU est donné en Figure 2-10. L'exécution en parallèle se décrit par la fonction par{}. L'architecture contient 16 processeurs par MP, chaque processeur pouvant effectuer deux tâches. Il est donc possible d'exécuter 32 calculs en parallèle. Ainsi pour le calcul des moyennes des 992 longueurs d'onde, on effectue 32 moyennes en parallèles

31 fois. Ces résultats sont envoyés ensuite sur un seul et même processeur afin de calculer le résultat final. Ceci correspond aux boucles imbriquées des fonctions de moyenne et de la fonction qui la suit de la Figure 2-10.

La localité 2D de l'architecture GPU est préservée en utilisant la mémoire cache de texture 2D. Les résultats des fonctions exécutées en parallèle étant réutilisés par la fonction séquentielle associée, il est nécessaire de considérer l'organisation mémoire des résultats et la synchronisation des tâches.

Les performances du GPU pour une seule région sont présentées dans la Figure 2-11. Sur cette figure les courbes décrivent les temps de calcul de la moyenne pour différentes tailles de région, le temps des accès à la mémoire (défini comme le temps de transfert) ainsi que le temps total d'exécution de l'algorithme.

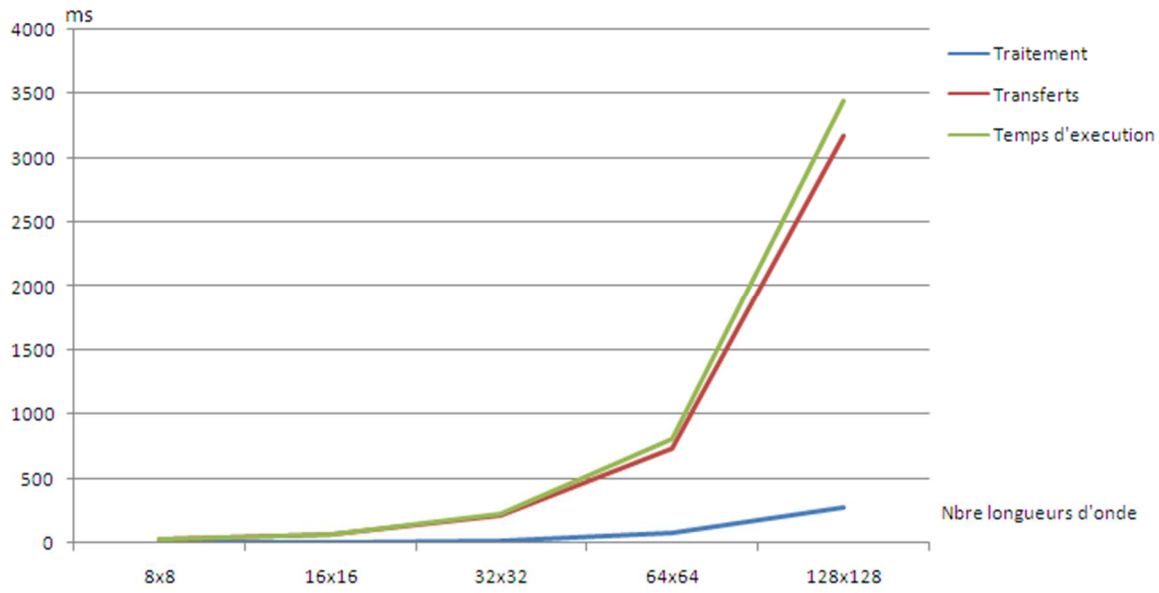


Figure 2-11 : Temps de traitement, d'accès mémoire et temps d'exécution total

Comme nous pouvons le voir, le temps de transfert des données est important par rapport au temps de traitement lui-même. De ce fait, le taux d'occupation est faible. Le calcul représente entre 5% et 10% du temps total pour l'ensemble des paramétrages testés pour l'application spectrale. La Figure 2-12 représente le pourcentage d'occupation du GPU entre les transferts des données et les calculs à effectuer. Le traitement représente moins de 10% du temps total, la majorité du temps étant réservé aux transferts des données dans le GPU.

Le temps d'exécution obtenu avec 2000 régions de taille 64x64 est d'environ 4 secondes, avec la récupération des copies mémoire et les traitements de données. Le goulot d'étranglement se fait sur le bus PCI-Express 2.0 avec 10 secondes de copies mémoire à la vitesse de 6 Go / s.

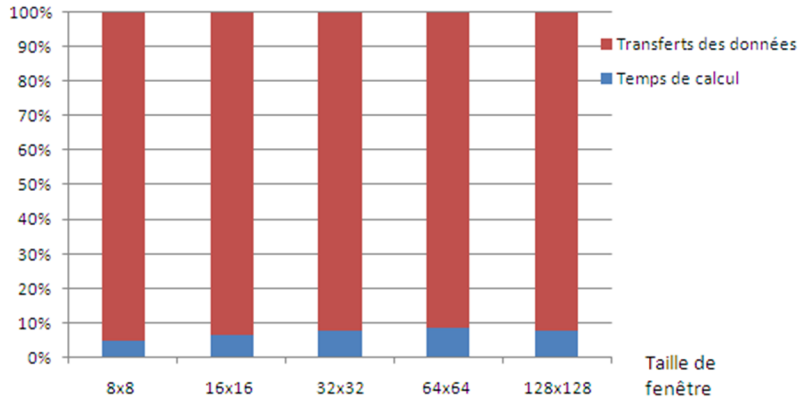


Figure 2-12 : Pourcentage d'occupation du GPU entre le calcul et le transfert.

Les fonctions de calcul sont exécutées sur 4 MPs (32 processeurs) pour une région, et chaque processeur exécute séquentiellement 31 fois le même calcul. Le GPU ne peut exécuter que 4 régions au maximum en parallèle. Au-delà, la taille de la mémoire du GPU ne permet pas de stocker toutes les régions et il est nécessaire d'effectuer le calcul de manière séquentielle. Ainsi, le temps d'exécution augmente de 4 en 4 régions comme le montre la Figure 2-13.

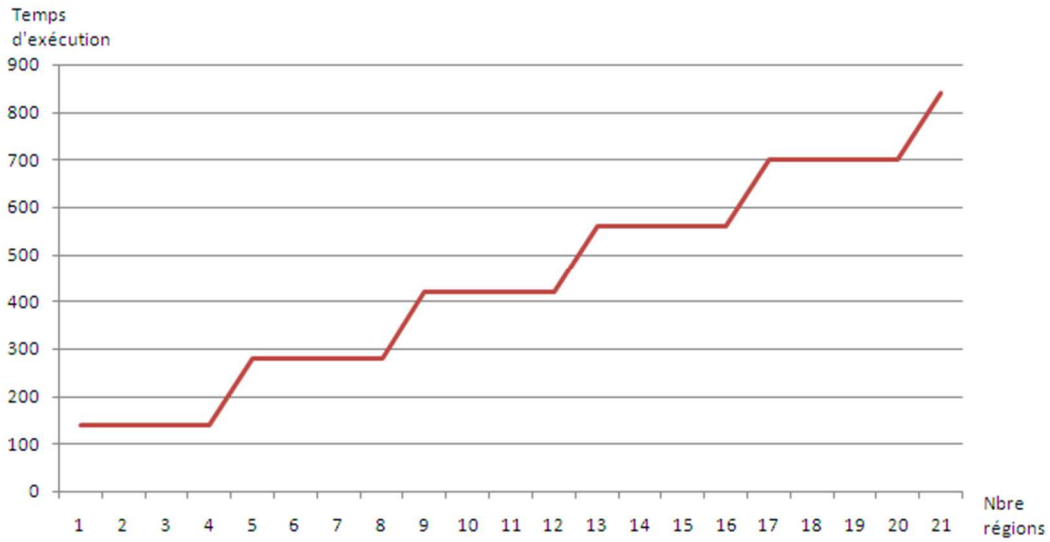


Figure 2-13 : Calcul du temps d'exécution en fonction du nombre de régions (taille de fenêtre de 64*64 pour 992 longueurs d'onde).

D'un point de vue programmation, le code de l'algorithme adapté à l'architecture GPU est environ 300 lignes. L'adaptation du code C en code CUDA a nécessité environs 2 heures en considérant un programmeur expérimenté. Cette adaptation requiert une connaissance des besoins de l'algorithme et de l'architecture pour adapter de manière la plus efficace.

2.4.3 Analyse des temps d'exécution sur architecture CPU et GPU.

Les temps d'exécution de l'algorithme pour des architectures CPU et GPU sont donnés en Figure 2-14 et Figure 2-15 pour des nombres de longueurs d'onde respectifs de 128 et 480.

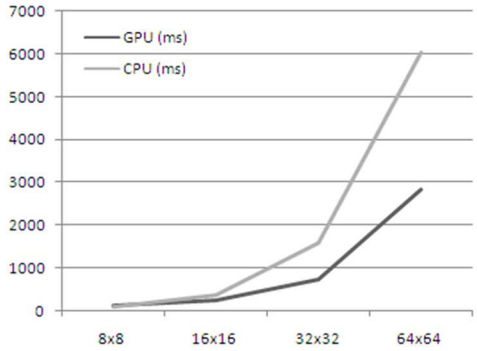


Figure 2-14 : Temps d'exécution de l'application sur CPU et GPU pour W=128

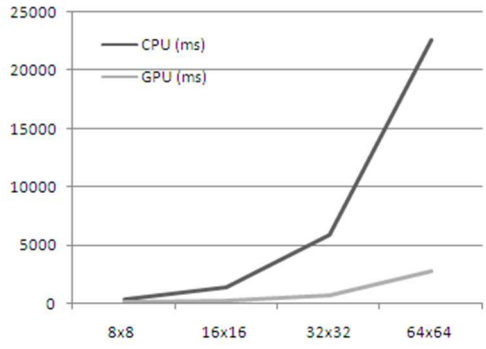


Figure 2-15 : Temps d'exécution de l'application sur CPU et GPU pour W=480

L'architecture GPU donne de meilleures performances que l'architecture CPU, notamment lorsque la taille des fenêtres pour le calcul de la moyenne est élevée ainsi que le nombre de longueurs d'onde. Le temps d'exécution de l'application est fonction du nombre de longueurs d'onde pour les architectures CPU mais ce n'est pas le cas pour les GPUs. En effet, le temps d'exécution est multiplié par un facteur d'environ 4 entre une application utilisant 128 et 480 longueurs d'onde. Le temps d'exécution reste le même pour le cas du GPU. D'un point de vue général, le parallélisme de données et le pipeline disponibles dans les architectures GPU sont adaptés à notre application, cette dernière présentant les mêmes caractéristiques que les applications graphiques. Les stratégies des architectures GPU sont aussi appropriées pour des algorithmes traitant des tailles de régions ou de nombre de données (ici des longueurs d'onde élevées). Néanmoins, quelques limitations apparaissent dans l'utilisation de ces architectures GPU :

- La taille mémoire du GPU limite le parallélisme potentiel de l'algorithme. Dans notre cas, le nombre maximal de processeurs est limité à 32 du fait du remplissage complet de la mémoire. La puissance de calcul du GPU n'est pas pleinement utilisée.
- La communication avec le PC étant faite par un bus auquel accèdent les différents MP du GPU, la bande passante est limitée, ce qui limite le temps d'exécution de l'algorithme.
- Le portage de l'algorithme d'authentification dépend fortement de la plateforme utilisée. Le parallélisme de l'application a été déployé pour une configuration GPU bien précise. En cas de changements de paramètres de l'application ou d'architecture GPU, le concepteur doit adapter le code, augmentant de manière significative le temps de conception. Ce portage

nécessite des connaissances approfondies sur les techniques de parallélisation et les architectures GPU.

2.5 Les réseaux de communication sur puces

Les bus partagés constituaient les solutions d'interconnexion des SoC il y a quelques années. Néanmoins, avec l'augmentation de la complexité des SoC, cette solution classique reste limitée aux architectures à faible nombre de cœurs avec une flexibilité et des performances limitées. Aujourd'hui, le système d'interconnexion de type réseau sur puce est la solution incontournable pour les communications dans des architectures SoC multicores et manycores.

2.5.1 Le réseau sur puce (NoC)

Au début des années 2000, le réseau sur puce appelé Network-on-Chip (NoC) est présenté comme une approche appropriée pour la communication haute performance entre les nombreux IPs disponibles dans un SoC[24][25][26]. Le NoC se base sur les concepts des réseaux informatiques, ces concepts ayant été adaptés aux SoC. Le NoC améliore l'évolutivité et l'extensiblité des SoCs en maintenant une faible consommation.

D'une façon générale, un NoC est constitué de plusieurs éléments de base qui sont :

- **Les Network Interface (NI, Interface réseau) ou Network Adapter (NA, adaptateur réseau)** qui réalisent l'interface entre les blocs IP et les communications gérées par le NoC. L'interface adapte les données provenant du bloc IP au format de communication du NoC en rajoutant des informations nécessaires au transfert des paquets dans le réseau. De la même manière, le NI extrait les données provenant des paquets du NoC pour les envoyer vers le bloc IP.
- **Les routeurs** permettent l'aiguillage des paquets de données dans le réseau, l'aiguillage se faisant selon l'algorithme de routage du NoC qui constitue l'arbitrage du routeur.
- **Les liens** qui relient les routeurs entre eux ou les routeurs aux NI. Ils sont mono ou bidirectionnels selon les réseaux (mais en général, ils sont bidirectionnels) et déterminent la bande passante des communications entre source et destination.
- **Les unités de traitement** qui correspondent aux différents modules du SoC comme les blocs IP, les mémoires, les processeurs, … chargés de traiter les données.

La Figure 2-16 représente une architecture NoC. Les routeurs (notés switch) sont interconnectés entre eux via des liens selon une topologie donnée. Des unités de traitement sont connectées sur le port local des routeurs et l'adaptation entre l'unité de traitement de le switch se fait au moyen

d'adaptateur réseau (notés NI). Plusieurs unités de différents types peuvent se greffer sur un routeur, le bloc NI doit posséder plusieurs sorties vers l'unité de traitement comme représenté à droite.

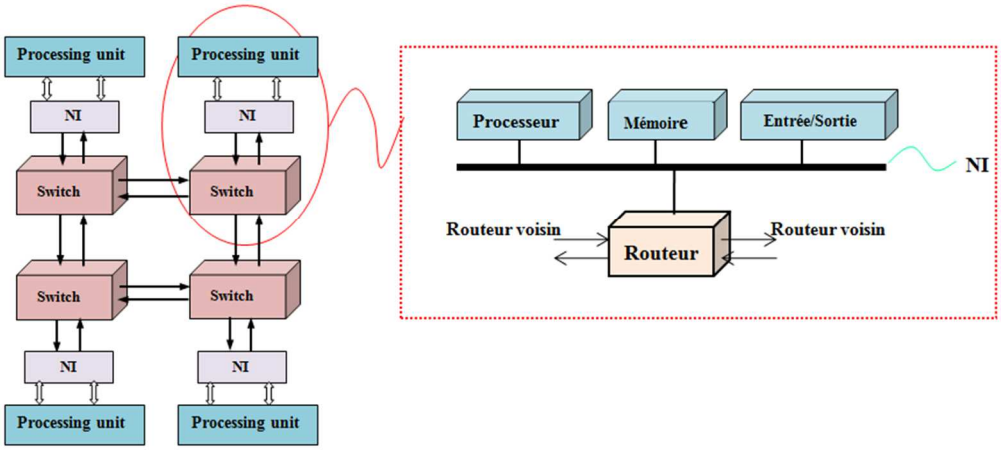


Figure 2-16 : Architecture de NoC

La conception du NoC doit prendre en considération de nombreuses contraintes pour répondre aux besoins des applications, entre autres performances, consommation d'énergie, surface de silicium, synchronisation, et performance du système. Pour faire face, les caractéristiques du NoC telle que la topologie, la gestion de flux, les technique de commutation doivent être considérées [43].

2.5.1.1 Les topologies de réseaux sur puce

La topologie d'un réseau décrit l'organisation physique du réseau autour de laquelle sont positionnés les différents nœuds du réseau. La topologie du réseau correspond à l'organisation de l'interconnexion des unités par des canaux de communication. Elle est définie comme un ensemble N de nœuds connectés par un ensemble C de canaux. Il existe plusieurs topologies de réseau, chacune avec leurs avantages et leurs inconvénients. Il faut donc trouver la topologie permettant le meilleur compromis entre performances requises par l'application et le coût matériel engendré par le réseau. Nous allons présenter ci-dessous quelques topologies du réseau, les plus couramment utilisées.

- Les topologies maillées 2D: la topologie 2D maillée (2D Mesh,Figure 2-17)est certainement la plus utilisée pour les NoCs de part sa simplicité de mise en œuvre et sa structure régulière [27][28]. En effet, cette topologie est facilement implantable sur une technologie silicium ainsi que sur FPGA. De plus, grâce à sa structure régulière, les algorithmes de routage sont simples à instaurer et il est possible d'accroître facilement cette structure pour des applications exigeant plus de performances. La topologie 2D maillée

torus en est une extension, offrant une meilleure bande passante au détriment de la simplicité d'implantation sur des circuits.

- La topologie **3D maillée** (Figure 2-18) offre une bande passante supérieure aux deux topologies présentées précédemment en conservant des algorithmes de routage simples. Cependant, le routage sur silicium est beaucoup plus complexe ce qui rend difficile son implantation. La possibilité d'accroître la structure est très limitée ce qui en fait une topologie beaucoup moins utilisée que les topologies 2D. Néanmoins, avec l'apparition des circuits 3D ou multi-chip, il est à penser que cette topologie pourra être de plus en plus utilisée.

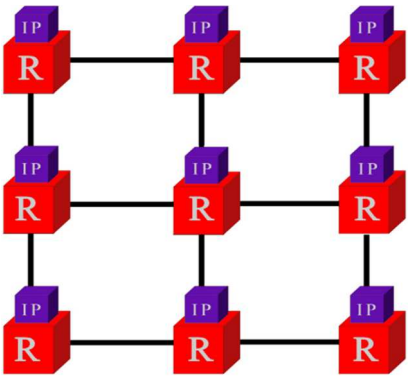


Figure 2-17 : Topologie 2D maillée

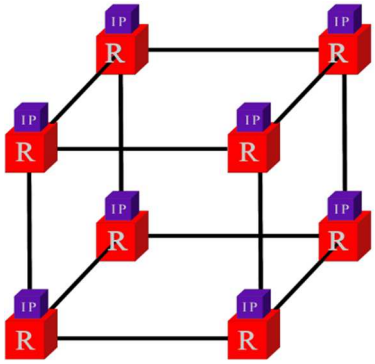


Figure 2-18 : Topologie 3D maillée

- Les **topologies en arbre** : un exemple de topologie en arbre élargi (Figure 2-19), aussi appelée Fat-Tree, est composée de « branches » et de « feuilles » sur lesquelles on retrouve les différents routeurs et blocs IP. Cette topologie permet une extensibilité importante, pouvant supporter d'autres applications exigeant plus de performances. De plus, la latence dans une telle architecture peut être plus faible que les topologies maillées, à condition que les nœuds dépendant les uns des autres soient placés à proximité. Mais une telle topologie est limitée lorsque plusieurs nœuds (représentant les blocs IP) d'une même branche veulent communiquer avec des nœuds d'autres branches. Il se crée un goulot d'étranglement des communications au niveau des routeurs intermédiaires. Afin de réduire cet étranglement des communications, il est indispensable de bien placer les différents blocs IP.
- Les **topologies en anneau et octogone** : les routeurs sont reliés à leurs voisins par des liens unidirectionnels (Figure 2-20). Elles sont en général facilement implantables sur une technologie silicium et les algorithmes de routage sont assez simples. Le principal inconvénient survient lorsque le réseau possède une grande taille, car dans ce cas, les communications doivent traverser un grand nombre de routeurs ce qui limite la bande passante.

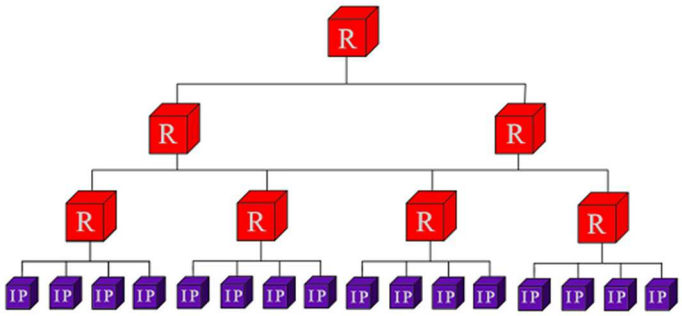


Figure 2-19 : Topologie en arbre élargie (fat-tree)

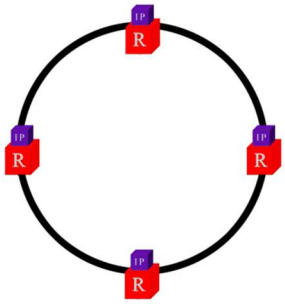


Figure 2-20 : Topologie en anneau

- Les **réseaux hybrides** : ces réseaux regroupent toutes les topologies particulières qui ne peuvent être décrites par les topologies précédentes. Ces réseaux sont souvent une association de topologies régulières, modifiées par la suite ou non. Bien que les réseaux hybrides possèdent généralement des performances plus intéressantes que les réseaux composés de topologies régulières [29], ils sont le plus souvent dédiés à une application donnée. Un réseau hybride avec une topologie en anneaux hiérarchiques est présenté dans la Figure 2-21.

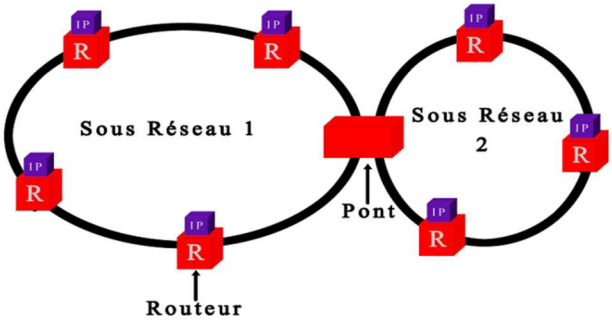


Figure 2-21 : Réseau hybride avec topologie en anneaux hiérarchiques

2.5.1.2 Mode de commutation et acheminement des données dans les NoCs

Après avoir défini une topologie de NoC, il faut pouvoir assurer la communication des différentes données transférées dans le réseau. Le NoC doit garantir la communication entre chaque routeur du réseau pour permettre le bon fonctionnement de l'application. Les formats des données circulant dans le NoC sont présentés, ainsi que les modes de commutation.

a. Les données de communication

Pour mieux comprendre les différents points abordés dans cette partie, il est indispensable de définir quelques éléments de base caractérisant le format des données circulant à travers un NoC :

- L'information que l'on veut transmettre est appelée message et représente la totalité de l'information à transmettre. Le message peut contenir autant de données qu'il le souhaite.
- Un message est découpé en paquets (packet) pour permettre les communications en parallèle ou lorsque le message est trop grand.
- Un paquet peut être décomposé en FLIT, FLow control unIT : il correspond au plus petit élément de base d'un message circulant dans le NoC.

Figure 2-22 : Format des données circulant dans un NoC

b. Les mécanismes et mode de commutation

Les mécanismes de commutation permettent de gérer les données de communication dans un NoC. Les deux principaux mécanismes de commutation utilisés dans les Network-on-Chip sont le circuit switching (*commutation de circuit*) et le packet switching (*commutation de paquet*)[30].

- Le circuit switching ou commutation de circuit

Le réseau transmet les données du nœud source au nœud destination par l'intermédiaire d'un circuit dédié. Un circuit dédié correspond à un chemin physique réservé dans le réseau reliant source et destination afin de transmettre toutes les données via un seul et même chemin [26]. L'établissement de la liaison physique se fait via un paquet émis par la source, qui va émettre un message à travers le réseau et bloquer un chemin physique entre la source et la destination. Les liens nécessaires pour envoyer les paquets à destination sont réservés grâce à des signaux de contrôle ou par un paquet

contenant l'adresse de destination du message. Ces liens sont alors bloqués pour transmettre l'information envoyée par la source et sont ensuite libérés une fois que le message est arrivé à destination.

De part son principe de fonctionnement, la commutation de circuit est idéale pour la transmission de données en temps réel. Ce mécanisme de commutation est approprié pour les communications peu fréquentes avec une grande quantité de données et permet l'utilisation totale de la bande passante du canal physique réservé pour établir la connexion. Par contre, une fois que ce lien est réservé, il est réservé pour une seule communication : il est donc impossible que d'autres communications empruntent le même chemin ou même une partie du chemin. Les messages restent bloqués et stockés dans des buffers (stockage difficile à mettre en œuvre étant donnée la taille importante des messages) jusqu'à la libération du lien réservé.

Le réseau utilisant la commutation de circuit le plus connu est le circuit téléphonique pour lequel une ligne est réservée puis bloquée pendant une communication.

- Le packet switching ou commutation de paquet

Ce mécanisme de commutation est celui utilisé par internet. Lorsqu'un message à transmettre a une taille importante, il est découpé en plusieurs **paquets** de taille fixe qui sont envoyés individuellement vers la destination choisie. Les paquets peuvent emprunter des chemins différents. Il est alors nécessaire d'identifier les paquets envoyés et ceux reçus. Pour ce faire, chaque paquet issu du message est composé :

- d'une partie contrôle, constituée d'un **entête** contenant l'adresse de destination du paquet et d'autres informations utiles [31] ;
- d'une partie donnée, formée par ce qu'on appelle la **charge** du paquet, contenant les données de l'information à transmettre.

De cette façon, lorsqu'un paquet arrive à un routeur, celui-ci le stocke dans un buffer puis le routeur vérifie l'adresse de destination contenue dans l'entête du paquet. Ensuite, il sélectionne un chemin vers un autre routeur selon cette adresse et envoie le paquet à ce routeur si toutefois le lien qui relie les deux routeurs est disponible. Le lien entre deux routeurs est donc utilisé uniquement lorsqu'il est nécessaire de transmettre des données. Le chemin qu'empruntent les paquets entre une source et une destination est choisi par l'algorithme de routage choisi et intégré dans chaque routeur. Plusieurs types d'algorithmes de routage existent : les algorithmes déterministes, les semi-adaptatifs et les adaptatifs.

Le mécanisme de commutation de paquets possède plusieurs modes de commutation. Les modes les plus utilisés sont présentés :

- **Store-and-forward** : ce mode de commutation consiste à faire transiter les paquets dans son ensemble d'un nœud à l'autre. Chaque nœud doit être en mesure de stocker la totalité d'un paquet avant de le renvoyer. Cette capacité de stockage limite la taille maximale des paquets. De plus la latence des échanges est importante puisqu'il faut multiplier le temps de transfert d'un paquet entre deux nœuds par le nombre de nœuds traversés du réseau.
- **Virtual cut-through** : un nœud peut commencer à envoyer un paquet si le nœud suivant lui garantit qu'il peut stocker le paquet dans sa totalité. Dans le cas contraire le nœud doit pouvoir garder le paquet. La capacité de mémorisation du nœud est donc la même que pour le mode store-and-forward mais la latence est diminuée puisqu'il n'est plus nécessaire d'attendre la réception complète du paquet pour qu'il passe d'un nœud à l'autre.
- **Wormhole** : ce mode a pour but de réduire la taille des mémoires dans les nœuds sans limiter la taille des paquets. Les flits passent de nœud en nœud dès qu'il y a de la place pour un flit et pas nécessairement pour un paquet complet. Le 1er flit est un entête qui contient des informations de contrôle, en particulier sur la destination du paquet. Tous les flits qui suivent doivent emprunter le même chemin que le flit d'entête. Un même paquet peut donc être réparti sur plusieurs nœuds du réseau. Cette commutation permet alors de réduire la latence.

Le choix du mécanisme et du mode de commutation dépend des contraintes de l'application (quantité de données à mémoriser et transférer, nombre de nœuds source et destination…) et des performances souhaitées (bande passante, latence, ressources). D'un côté, il semblerait que le mécanisme« *packet switching* » de type « *Wormhole* » soit le mieux adapté pour des messages de petites tailles ou un grand nombre de paquets. Cette méthode de gestion de flux de communication permet d'optimiser les ressources matérielles et une réduction de la latence des communications. D'un autre côté, le *circuit « switching »* est idéal pour les messages de grande taille et garanti le temps d'acheminement des données. Cependant, les ressources matérielles ne sont pas optimisées.

2.5.2 Les NoCs sur FPGA

Il existe à l'heure actuelle un très grand nombre de NoCs[44]. Ces architectures sont développées pour des composants ou plateformes dédiées ou uniquement pour des validations fonctionnelles ou de nouvelles propositions de concepts de NoC. De nombreux NoC ciblant des ASIC sont disponibles en littérature et il est difficile de les présenter tous [35][35][36][37][39][40][41]. La

plateforme de prototypage utilisée par la suite étant constituée de FPGAs, les NoCs présentés dans le Tableau 2-2 ciblent des FPGAs.

Tableau 2-2 : NoC sur architecture mono-FPGA

Nom	Topologie	Mécanisme de commutation	Algorithme de routage	FPGA
HERMES[45]	2D maillée	Wormhole	XY + algorithmes semi-adaptatifs	Xilinx Virtex II
NoC de l'IMEC [51]	2D tore	Wormhole	XY	Xilinx Virtex
SOCIN[52]	2D maillée ou tore	Wormhole	XY	Altera EPF10K200
PNoC[46][53]	Arbre élargi	Circuit	Source	Xilinx Virtex II/1305
HIBI[54]	variable	Circuit	Distribué et adaptatif	Altera Statix 1S40
DyNoC[49]	2D maillée	paquet	S-XY	Xilinx VirtexII sur RC200 Celoxica
CuNoC[47][48]	2D maillée	Paquet	XY	Xilinx Virtex II et IV
ESIEE[42]	Anneau	Store and forward	Dynamique entre 2 maîtres	Altera Stratix III

La plupart des NoCs implantés sur composant FPGA possède une topologie maillée 2D. Cette topologie correspondant bien aux architectures de programmation de ces circuits, la structure du circuit de programmation étant une structure 2D de type maillé (les cellules logiques pouvant être reliées via des liens courts sur les 4 directions identiques des NoCs). Le mécanisme de commutation est en majorité le « paquet switching » avec l'utilisation d'algorithmes de routage XY.

Il nous semble inutile de concevoir notre propre NoC pour l'application d'authentification, des NoCs étant déjà disponibles en VHDL synthétisable. Les NoC étant des architectures de communication flexibles, génériques et évolutives, il apparait donc évident que la réutilisation d'un NoC non dédié doit pouvoir s'intégrer dans la conception de la plateforme d'émulation et d'exploration souhaitée, ce qui constitue un des objectifs de la thèse.

Par conséquent, les caractéristiques choisies pour le NoC sont :

- Topologie maillée : les topologies en anneau ou en arbre sont moins extensibles que la topologie maillée, cette dernière semble bien adaptée à un portage sur des circuits FPGA.

- Mécanisme de commutation « packet switching » en « wormhole ».
- Algorithmes de routage : XY

Le NoC Hermes a donc été choisi comme architecture de test dans la suite de ce manuscrit. Il présente les caractéristiques recherchées, est disponible en VHDL synthétisable et peut se paramétrer aisément.

2.6 Conclusion

Dans ce chapitre, nous avons présenté l'application d'imagerie spectrale pour l'authentification d'œuvres d'art et montré que les architectures programmables telles que des CPU et GPU ne sont pas adaptées pour de telles applications pour des raisons différentes. Le CPU ne permet pas d'accélérer suffisamment les temps de calcul, le nombre de cœurs étant limité et la stratégie des architectures CPU n'étant pas adaptée aux applications spectrales. Les GPUs sont plus prometteurs car ils présentent un parallélisme de données important. Cependant l'adaptation de l'application sur l'architecture est très spécialisée et propre à chaque paramétrage de l'application et chaque architecture GPU. De plus, des goulots d'étranglement apparaissent au niveau mémoire et communication. Il est donc difficile d'envisager le faire des explorations d'espace de conception sur de telles architectures.

Le choix des plateformes se tourne donc vers des architectures MPSoC à base de NoC sur FPGA. Le NoC choisi est le Noc Hermes car il offre toutes les caractéristiques les plus adaptées à notre problème. De plus, le code source est disponible, ce qui nous permettra de le faire évoluer.

Chapitre 3. Proposition d'une méthodologie pour la conception et l'exploration de NoC

Dans ce chapitre, nous proposons un flot de conception générique pour l'émulation des données circulant dans un MPSoCs via un NoC sur architectures FPGA. Les scénarii proposés permettent de couvrir l'émulation des différentes données pour des applications de traitement d'images. Ce flot de conception construit automatiquement la plateforme d'émulation basée sur un NoC existant. Le flot proposé intègre des bibliothèques nécessaires pour l'insertion de l'algorithme de routage, des blocs d'émulation et les interfaces nécessaires pour déployer le NoC sur une plateforme multi-FPGA si un partitionnement multi-FPGA a été spécifié. L'architecture d'émulation générée par le flot est basée sur une structure hiérarchique VHDL synthétisable afin de permettre une portabilité maximale.

3.1 Architectures NoC et plateformes d'évaluation

Les deux contributions majeures présentées dans ce chapitre sont le déploiement de NoC de taille importante sur des architectures multi-FPGA ainsi que l'émulation des transferts de données pour le traitement d'images sur FPGA. Il est par conséquent important de positionner notre travail par rapport aux travaux existants pour montrer nos contributions dans ces domaines.

3.1.1 Les NoC sur FPGA

Aujourd'hui, de nombreuses architectures NoC ont été développées pour des architectures FPGA. Les NoCs les plus courants sont présentés dans le chapitre 2.5.2. Ces architectures ont été développées pour des cibles mono-FPGA. Ceux-ci souffrent d'un manque d'extensibilité vers des architectures multi-composants possédant plusieurs FPGAs. Quelques NoCs présents dans la littérature ciblent des cibles multi-FPGA. Deux approches bien différentes en ressortent. La première approche consiste à couper le NoC, généralement de manière équilibrée et de placer chaque partie sur un FPGA [64][65]. Les liens initialement internes sont remplacés par des communications inter-FPGA utilisant des bus série ou parallèles pour connecter deux routeurs situés sur 2 FPGA différents. Afin de préserver les performances de communication du NoC, un

lien est remplacé par un bus externe. Les avantages de cette approche sont le maintien de la structure du NoC permettant d'évaluer les performances des communications et une portabilité directe de la structure. Toute l'architecture du NoC, y compris les algorithmes de routage 2D, est directement réutilisable (tous les routeurs restant identifiés par des coordonnées en X et Y), la modification consistant aux remplacements de liens externes. Il est par conséquent facile de définir la latence d'une communication entre deux routeurs situés sur 2 FPGAs en connaissant les temps de transfert via les bus externes utilisés et leurs équivalences via un lien interne. La limitation de cette approche vient des cartes multi-FPGA utilisées. Le nombre de bus externes d'une carte reste généralement limité, et il s'avère difficile d'envisager le déploiement d'un NoC de grande taille en utilisant un nombre minimal de cartes standards. Par exemple déployer un NoC 8*8 sur 2 FPGAs nécessite déjà l'utilisation de 8 bus externes. Il devient vite nécessaire d'augmenter le nombre de FPGAs pour augmenter le degré de partitionnement et disposer du nombre d'interfaces demandé. Un exemple est le NoC maillé 4x4 déployé sur 4 FPGAs par Hammami en Figure 3-1[65].

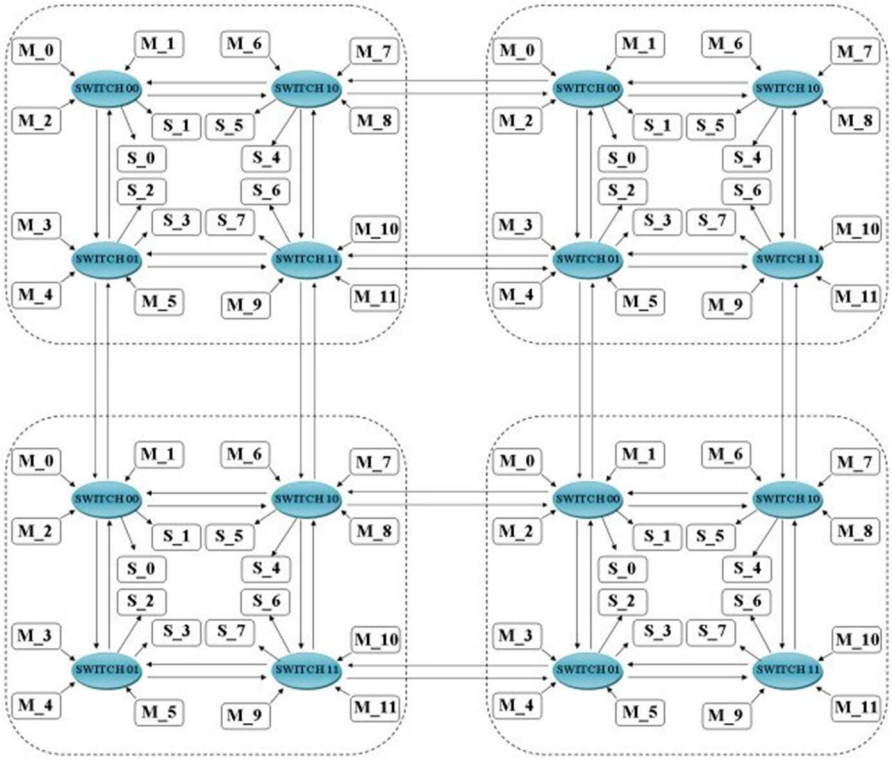


Figure 3-1 : Déploiement du NoC sur 4 FPGAs.

La deuxième approche consiste à développer une structure dédiée aux communications inter-FPGA en rajoutant une structure hiérarchique supplémentaire prenant en compte la couche de communication entre les composants. Cette approche se base sur le concept des architectures 3D

pour lesquelles une dimension Z est ajoutée (Figure 3-2). A partir d'une structure de NoC 2D, une couche de communication dédiée aux communications entre FPGA est ajoutée. Le NoC contient en plus des blocs dédiés intégrés à la structure pour permettre ces communications inter-FPGA adaptés aux performances des liens externes [66][67][68][69]. Des routeurs dédiés appelés « gateways » sont proposés par Stepniewska et al. en [70]. Ces routeurs sont dédiés et optimisés pour des communications entre FPGAs. L'avantage d'une telle structure est l'extensibilité importante du NoC qui n'est plus limitée par le nombre de bus externes disponibles sur la carte. L'inconvénient majeur est la modification importante de la structure du NoC : l'algorithme de routage est dédié (algorithme de routage 3D), les communications inter-FPGA sont modifiées et des blocs supplémentaires sont insérés modifiant le nombre de ressources. Ce type de structure n'est pas conçu pour évaluer les performances d'un NoC en le prototypant sur une architecture multi-FPGAs. Il n'est pas envisageable d'évaluer les performances en communication d'un NoC en utilisant une telle architecture, la structure du NoC étant grandement modifiée ainsi que les chemins qu'empruntent les données.

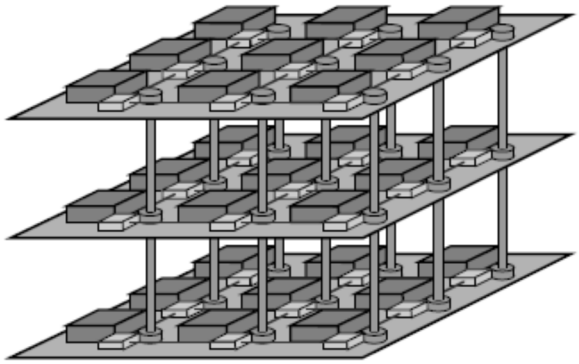


Figure 3-2 :Structure de NoC dédiée aux architectures multi-composant. Des blocs dédiés aux communications inter-composant sont conçus et intégrés dans la structure du NoC [74].

Dans ces deux approches, les limitations ne permettent pas d'évaluer les performances temporelles et matérielles d'un NoC en réalisant le déploiement de cette structure sur multi-FPGA. Nous proposons de présenter un NoC évaluable sur des architectures multi-FPGA pour laquelle la structure n'en est que légèrement modifiée et pour laquelle les évaluations de performances restent encore pertinentes.

3.1.2 Les plateformes d'évaluation et d'émulation de NoC

De nombreux outils et environnement ont été développés pour évaluer les performances de communications d'un NoC. Certains ont été développés dans des contextes applicatifs de traitement du signal et de l'image, d'autres pour des applications orientées télécommunication. Les évaluations

des communications sont basées sur la simulation en VHDL, en SystemC ou une combinaison de spécification, de simulation, d'analyse et la génération des NoCs à différents niveaux d'abstraction.

Dans [61], Mahadevan propose une architecture de modélisation, de simulation et d'évaluation SystemC du MPSoC intégrant un système d'exploitation temps réel basée autour d'un NoC. Dans [62] Chan propose un flot de conception mixte basé sur la simulation SystemC et l'implémentation en VHDL de la structure du NoC. Cette plateforme utilise un modèle de routeur pour simuler plusieurs réseaux d'interconnexion en utilisant SystemC. Un environnement de modélisation SystemC permet de définir une topologie ad hoc [63]. Ces approches sont limitées à leurs niveaux de précision définis en SystemC pour les estimations et le niveau de synthèse sur le FPGA.L'augmentation du niveau de précision de la simulation augmente le temps de simulation de manière considérable. Ces temps de simulations sont plus élevés que dans le cas d'émulation du NoC sur FPGA.L'évaluation des performances du NoC se fait en simulation et avec SystemC pour la plupart, l'implantation sur FPGA ne concerne que la structure du NoC à laquelle sont connectés les cœurs de calculs et de mémorisation pour constituer le système complet final. Ces outils ne permettent pas l'implantation des blocs de trafic sur FPGA pour émuler les trafics. Nous nous proposons d'émuler le NoC sur FPGA afin d'estimer les performances des communications directement sur le circuit. Ceci permet d'obtenir une plus grande précision des résultats en diminuant les temps de vérification. Cette approche a été utilisée par Bennini dans [57].Les auteurs présentent une architecture mixte logicielle matérielle implémentée sur FPGA Virtex-II. Cette architecture contient un réseau de communication, les générateurs de trafic, les récepteurs de trafic et un module de contrôle. Un processeur Hardcore (PowerPC) constitue le contrôleur global connecté à la plateforme d'émulation. Ce contrôleur définit les paramètres de l'émulation, envoie les données à tester et récupère les résultats de l'émulation. Cependant les paramètres proposés ne sont pas adaptés pour tous les transferts réalisés par les applications de traitement de signal et d'image. Une plateforme d'émulation rapide utilisant des cœurs matériels pour la reconfiguration partielle sur FPGA Virtex-II est également présentée dans [58].

Les plateformes d'émulation permettant l'évaluation des transferts de données dans un NoC sur FPGA ne sont pas adaptées aux traitements du signal et de l'image. Les trafics ne couvrent pas tous les scenarii des applications ciblées. De plus ces plateformes sont des plateformes d'évaluation et non d'exploration. Elles ne permettent pas de réaliser des explorations d'espace de conception en faisant varier certains paramètres de manière automatique aidant ainsi l'utilisateur à définir les paramètres les plus adaptés à son application. Dans tous les cas, la charge du réseau est un paramètre prédéfini par exemple.

Afin de résoudre ces problèmes, une plateforme d'émulation de NoC est proposée. Cette plateforme se compose de blocs d'émulation (générateur de trafic et récepteur de trafic) et d'une structure de NoC synthétisable. Cette plateforme permet d'explorer l'espace de conception du NoC et d'émuler tous les trafics requis par l'application de traitement d'image et du signal.

3.2 Méthodologie pour la conception et l'exploration de NoC : proposition d'un flot de conception

Un flot de conception générique est proposé pour générer une architecture d'émulation sur FPGA. La méthodologie construit automatiquement la plateforme complète et synthétisable en prenant toutes les entrées définies correctement au préalable. Les entrées sont soit des couples entité/architecture génériques décrits en VHDL synthétisable (blocs) soit des packages (spécifications). Le flot de conception contient :

- Une structure de NoC existante paramétrable,
- Des blocs d'algorithmes de routage,
- Des blocs de communication inter-FPGA,
- Des blocs d'adaptation paramétrables,
- Des blocs d'émulation : générateurs de trafic et récepteurs de trafic,
- Une spécification de routage,
- Une spécification de transfert de données,
- Une spécification de partitionnement du NoC.

Les différents éléments du NoC sont tout d'abord détaillés puis le flot de conception est présenté.

3.2.1 Structure de NoC

La structure utilisée dans le flot est un NoC maillé 2D pour laquelle chaque routeur possède 5 connexions, un exemple est donné en Figure 3-3. Le mode de commutation par paquet est choisi.

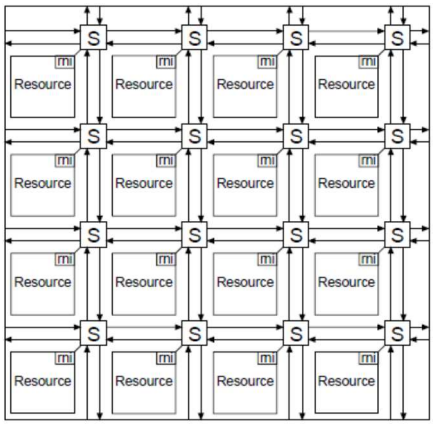


Figure 3-3 : Structure de NoC intégrée au flot de conception.

Dans ce mode de commutation, les paquets sont envoyés grâce à la charge du réseau. La charge du réseau est définie comme le rapport de la quantité de données qu'il reçoit sur sa capacité à acheminer des données [71] et est exprimée en pourcentage. La charge est définie par la formule suivante :

$$\boldsymbol{charge = \frac{ipr}{chr}} \qquad \textbf{Équation 3-1}$$

Avec la charge du réseau (en %), ipr le taux d'injection des données d'un bloc IP (en bps ou Mbps) et chr la bande passante totale du canal (en bps ou Mbps).

La charge du réseau varie entre 0% à 100 %. Des exemples de charge de 100% et de 50% sont donnés enFigure 3-4. Une charge de réseau de 100% indique que des paquets sont envoyés en continu dans le réseau. Généralement, une telle charge provoque des goulots d'étranglement, pénalisant la latence des paquets qui sont bloqués ou ralentis au niveau de certains routeurs. Une charge de réseau de 50% indique que le temps de transferts des paquets est suivi d'un temps d'attente de durée identique (un envoi de paquets de 10 ms suivi d'un temps d'attente de 10 ms sur l'exemple de la figure). La limite entre des transferts non bloquants et des goulots d'étranglement est donnée par le point de saturation.

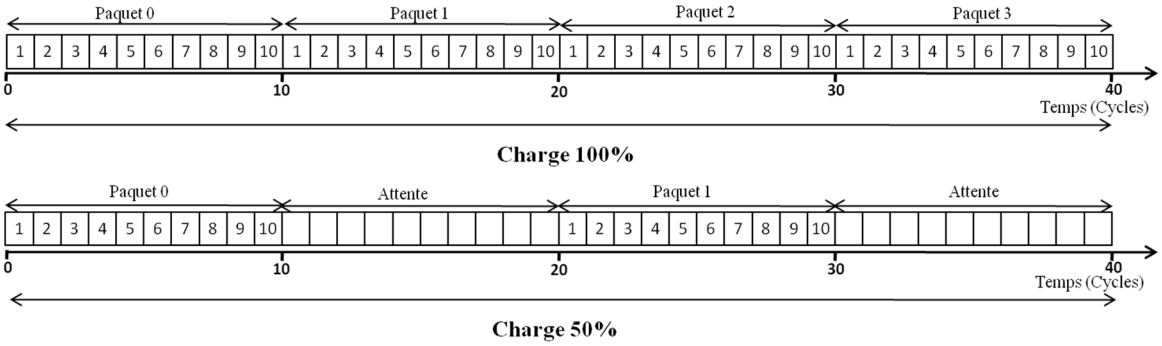


Figure 3-4 : Exemples de charge de réseau : 100% pour une charge continue du réseau et 50% pour une charge intermédiaire.

3.2.2 Algorithme de routage

Plusieurs algorithmes de routage de type Turn-Model sont implémentés sur FPGA et ASIC [72][73]. Les algorithmes de routage déterminent le chemin emprunté par un paquet entre la source et le routeur cible. Il existe trois types d'algorithmes de routage: déterministe, partiellement adaptatif et entièrement adaptatif. La plupart des algorithmes couramment utilisés sont les algorithmes de routage déterministes en raison de leur simplicité et d'un nombre minimal des ressources. L'algorithme XY est l'algorithme de routage déterministe le plus répandu. D'autres algorithmes de routage sont semi-déterministes(ou semi-adaptatifs) et il est affirmé en littérature que ceux-ci sont plus performants au détriment d'un nombre de ressources plus important. Un exemple d'algorithme semi-adaptatif est l'algorithme West first (Figure 3-5).Pour ces algorithmes, certains routages sont déterministes, d'autres sont adaptatifs, cela dépendant de la position des routeurs source et destination en général. Les paquets sont routés de manière déterministe si l'abscisse du nœud de destination est inférieure à l'abscisse du nœud source (chemins 1 et 2). Les paquets sont routés de manière adaptative si l'abscisse du nœud de destination est supérieure à l'abscisse du nœud source (chemins 3 et 4). Certains chemins sont interdits, le trajet sud ouest (représenté à gauche) et le trajet nord ouest (représenté à droite).

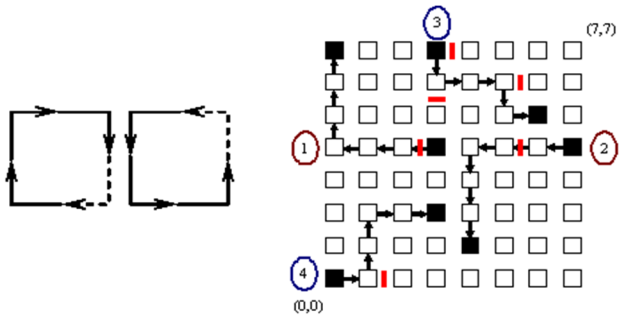


Figure 3-5 : Algorithme de routage West First. Les routages du chemin 1 et 2 sont déterministes alors que les chemins 3 et 4 utilisent un routage adaptatif. Les tours sud-ouest et nord-est ne sont pas autorisés.

Des algorithmes de routage entièrement adaptatifs sont proposés mais ils ne sont pas utilisés pour l'implémentation sur FPGA. La raison annoncée est le trop grand nombre de ressources et un algorithme plus complexe par rapport aux deux autres types. Tous les algorithmes de routage peuvent être insérés dans le flot de conception s'ils sont décrits en un langage HDL synthétisable. Certains blocs IP d'algorithme de routage ont déjà été développés en VHDL et sont insérés dans le flot de conception dans la bibliothèque **blocs IP de routage**.

3.2.3 Blocs d'adaptation pour la communication inter-FPGA

L'utilisation de plusieurs FPGAs peut s'avérer nécessaire dans les cas de grands systèmes ou de large NoC. Il convient alors de partitionner le NoC et de le déployer sur plusieurs FPGAs avec les communications inter-FPGA adaptées. Nous considérons deux types de communications : la communication routeur-routeur et la communication multi-routeurs. Dans le premier cas, deux routeurs communiquent via un média de communication externe. Le second type permet le partage d'un média par plusieurs routeurs (cas où le nombre de routeurs est supérieur au nombre de liens inter-FPGA existants).

L'idée est la réutilisation des liens de communication existants sur les cartes FPGA disponibles. Les communications parallèles et séries peuvent être utilisées entre deux FPGAs. Le lien série haut débit est utilisé comme média de communication et inséré dans le flot. Tout autre protocole peut être intégré si les blocs d'adaptation dédiés sont proposés. Les FPGAs les plus courants, Xilinx et Altera, possèdent des blocs série haut débit sérialiseur/déserialiseur (serialiser/deserialiser pour SerDes).Des liens « SerDes » possèdent deux blocs: PISO (Parallel in Serial out) et SIPO (Serial In Parallel Out).

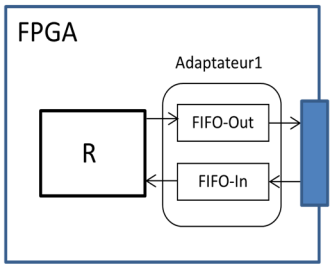


Figure 3-6 : Blocs d'adaptation pour une communication routeur-routeur.

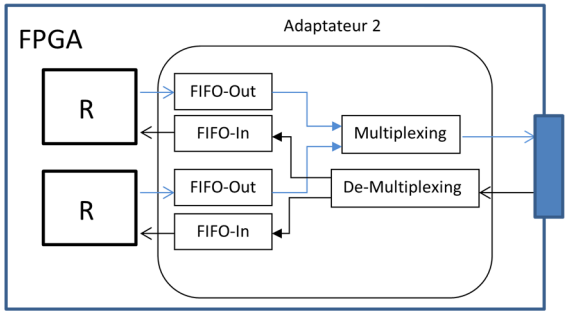


Figure 3-7 : Blocs d'adaptation pour une communication multi-routeurs.

Les blocs d'adaptation sont conçus pour répondre aux deux types de communication inter-FPGA. Ce choix dépend du nombre de routeurs par FPGA (N_R) et du nombre de liens inter-FPGA (N_{LI}) :

- $N_{LI}= N_R$: un routeur d'un FPGA communique avec un routeur situé sur un second FPGA au moyen d'une communication externe. Cela correspond aux communications routeur-routeur. L'adaptation consiste à insérer une FIFO double horloge dont la taille correspond au nombre de flits d'un paquet (Figure 3-6).
- $N_{LI}< N_R$: plusieurs routeurs d'un FPGA communiquent avec plusieurs routeurs d'un second FPGA. Cela correspond aux communications multi-routeur. Nous considérons que le nombre de routeurs est le même sur chaque FPGA. Dans cette configuration des blocs de multiplexage, demultiplexage, ainsi que des FIFOs sont insérés (Figure 3-7).

Ces blocs d'adaptation sont des entités paramétrables décrites en VHDL puis insérées dans la bibliothèque **Blocs d'adaptation** du flot.

3.2.4 Blocs d'émulation

Afin d'évaluer les performances du NoC indépendamment du système complet, les blocs IPs sont remplacés par les blocs d'émulation paramétrables. Les blocs d'émulations sont des générateurs de trafic et des récepteurs de trafic. Les blocs d'émulations sont connectés au NoC pour simuler divers trafics et analyser les performances, ces dernières permettant de choisir le meilleur paramétrage du NoC ou de choisir le NoC le plus adéquat parmi un panel de NoC disponibles.

3.2.4.1 Générateur de trafic

Le générateur de trafic (TG pour Traffic Generator) simule les flots de données sortant d'un cœur envoyé vers l'architecture de communication. Il existe deux types de générateurs de trafic :

- Le générateur de trafic déterministe : un TG déterministe permet en quelque sorte de modéliser les communications qu'émettent les blocs IP connectés aux NoC à partir de la trace

laissée par ceux-ci. Ce type de TG peut générer des transactions précises dans le temps, la taille et le temps d'inactivité qui correspondent au comportement d'un IP connecté au NoC. Ce type de trafic est utilisé pour un système complet (type et nombre de nœuds) et pour une application donnée. Les avantages de ces générateurs de trafic sont une précision élevée et le facteur accélération pour l'émulation par rapport à la simulation de tous les trafics.

- Le générateur de trafic stochastique : ces générateurs modélisent le trafic à l'aide de processus stochastiques et synthétisent des réalisations de ces processus pour générer du trafic. Ces TGs sont utilisés soit lorsque l'IP n'est pas totalement disponible ou lorsque le comportement est susceptible de changer d'une exécution à l'autre. Ce type de trafic fait abstraction des blocs IP et des scénarii de l'application et est utilisé pour dimensionner les limites de fonctionnement en réalisant l'ensemble des scénarii réalisables. Cependant, certains trafics sont très difficiles à modéliser et l'environnement de génération de trafic doit contenir des outils ou des architectures avancées d'analyse statistique parfois difficiles à mettre en place.

Dans ce travail, les générateurs de trafic déterministes sont utilisés, l'objectif étant de définir les performances du NoC pour les applications spectrales. Les blocs d'émulation sont développés à partir de la représentation de l'application sous forme de graphe flot de données (donné en 1.2.4). Les besoins en transferts de données sont :

- Un initiateur envoyant des données vers un ou plusieurs récepteurs,
- Plusieurs initiateurs envoyant des données vers un ou plusieurs récepteurs,
- Des variations de charge de réseau,
- Des variations de taille et de nombre de paquets pour chaque TG,
- Des informations sur l'origine des données arrivant au récepteur.

Deux formats de paquets sont proposés. Le premier format, Figure 3-8, permet le transfert de données dans un FPGA (cas d'implantation du NoC sur une architecture mono-FPGA).Le second, Figure 3-9, est dédié aux transferts de données sur plusieurs FPGAs (architecture multi-FPGA).

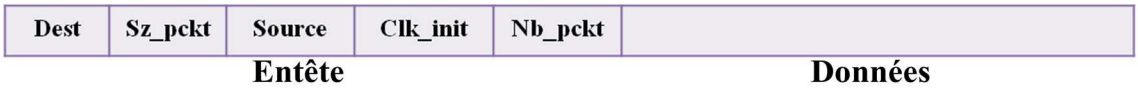


Figure 3-8 : Format des paquets générés par le TG pour une émulation mono-FPGA.

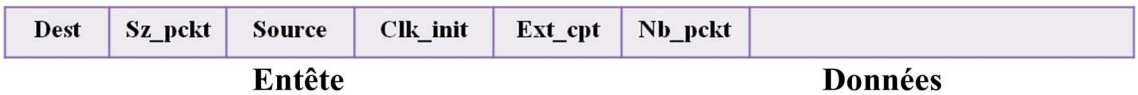


Figure 3-9 : Format des paquets générés par le TG pour une émulation multi-FPGA.

Dans notre plateforme d'émulation, les paquets contiennent une partie entête et une partie données avec les informations suivantes:

- Adresse du nœud de destination (Dest).
- Adresse de l'initiateur (Source).
- Horloge Init (Clk_init). Ce flit est réservé à l'évaluation de la latence. Lorsque le paquet est envoyé, la donnée de ce flit correspond au coup d'horloge sur lequel le paquet est envoyé.
- Taille du paquet transmis(Sz_pckt).
- Ext_cpt (uniquement pour le multi-FPGA): nombre de cycles pour le transfert des paquets en dehors des FPGAs.
- Nombre de paquets transmis (Nb_pckt).

Le TG générique développé et intégré au flot est représenté dans la Figure 3-10. Le TG génère des signaux de contrôle (*router_rx* et *router_ack_rx*) et le paquet dans la sortie *data_in* dont la taille est égale à la taille du bus. Les formats des paquets présentés précédemment sont envoyés sur ce dernier signal.

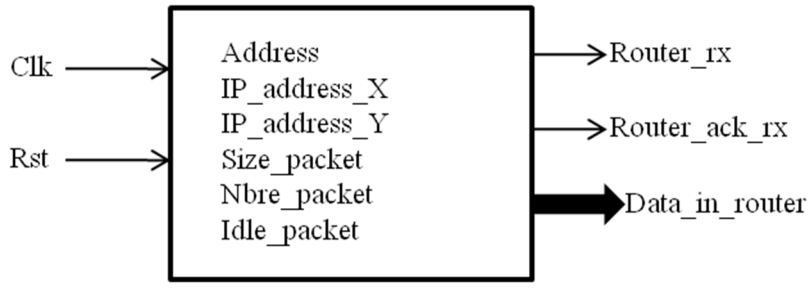


Figure 3-10 : Signaux et paramètres pour les générateurs de trafic génériques.

Le nombre et le format des paquets dépendent du scénario de trafic spécifié dans le package **Data_transfer** décrit plus loin dans ce chapitre.

Ce TG est écrit en VHDL générique et est inséré dans la bibliothèque **TG** et **TR** du flot.

3.2.4.2 Récepteur de trafic

Les flux générés par les générateurs de trafic sont envoyés parle NoC et sont ensuite reçus par le récepteur de trafic. Le récepteur de trafic permet d'extraire soit des informations de timing (latence des flux pour estimer les performances temporelles du NoC) soit des données brutes transférées dans les flux (pour vérifier que les transferts des données se passent bien). De même que pour les TGs, les récepteurs de trafic (TR pour Traffic Receptor) peuvent être classés en deux catégories :

- Le récepteur de trafic avec analyse d'activité : il consiste à analyser les performances temporelles des flux circulant dans le NoC. Ce type de récepteur est plutôt utilisé pour l'extraction de latences ou de débits. L'utilisateur définit la granularité de l'analyse, qui correspond au nombre de cycles considéré pour l'évaluation des performances. Par cette méthode, le TR peut générer un histogramme qui représente l'activité (en nombre de cycles ou en périodes) au niveau du récepteur. Dans la plupart des cas, un processeur embarqué constitue ce type de récepteur pour permettre l'analyse d'activité.

- Le récepteur de trafic avec analyse de trace : il génère un rapport continu de traces reçu avec les valeurs détaillées pour l'émulation. Le récepteur récupère les données et les affiche pour vérifier l'envoi, le transfert et la réception correcte des données dans le réseau. Ce type de récepteur ne permet pas d'évaluer les performances, juste de vérifier les valeurs des données arrivant en sortie du NoC.

Les deux types de récepteurs de trafic décrits sous forme d'entités VHDL paramétrables sont proposés et insérés dans la bibliothèque **TG et TR** dans notre flot de conception.

3.2.5 Spécification des transferts de données

Les besoins en transferts dans les applications de traitement d'images développées sous forme de graphe flot de données ont été détaillés au chapitre 3.2.4.1. Ces besoins constituent le point d'entrée pour la proposition du package dans lequel les transferts de données sont spécifiés.

Les blocs TG et TR sont paramétrés et insérés (ou non) selon la spécification de transfert de données. Ces spécifications de transferts sont données au dernier niveau de la description de l'architecture d'émulation correspondant à la dernière étape du flot. Dans ce package, le concepteur spécifie la taille (*size_packet*) et le nombre des paquets (*nb_packet*) envoyés pour chaque TG avec la charge (*idle_packet*).Les données ont le même format pour un TG mais peuvent différer pour chaque TG. Le concepteur indique tous les nœuds de destination avec une valeur de destination. La valeur 1 indique que le nœud est un TR et 0 que le nœud n'émet aucune donnée. *Last_destination* indique le nombre de TRs pour chaque TG. Le nombre de paquets reçus par chaque TR est donné par *total_packet*. Puis les liens entre les paquets envoyés par les TGs et reçus par les TRs sont donnés avec *destination_links*.

```
Size_packet: IntegNORT:= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15);
Nb_packet: IntegNORT:= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,10,10);
Idle_packet: IntegNORT:= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,20);
Destination : IntegNORT:= (1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0);
Last_destination : IntegNORT:= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,3);
Total_packet : IntegNORT:= (20,10,10,0,10,0,0,0,0,0,0,0,0,0,0,0);
Destination_link: addrs
:= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);  -- Node (0,0)
:= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);  -- Node (1,0)
.....
:= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);  -- Node (1,3)
:= (1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0);  -- Node (2,3)
:= (1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0);  -- Node (3,3)
```

Figure 3-11 : Spécification de transferts de données de plusieurs initiateurs vers plusieurs destinataires avec des charges de réseau fixe.

L'exemple représenté dans la Figure 3-11 permet de définir les transferts de données sur un NoC maillé 4x4. Les spécifications données indiquent :

- Les deux derniers nœuds du NoC, les routeurs (2,3) et (3,3) sont les générateurs de trafic,
- Ces TGs envoient 10 paquets de 15 flits,
- Le temps d'attente entre les paquets est de 20 cycles d'horloge,
- Les nœuds de destination sont les routeurs (0,0) (1,0) (2,0) (1,0),
- Le routeur (0,0) reçoit 20 paquets, les autres en reçoivent 10,
- Le routeur (2,3) envoie à deux TRs, le routeur (3,3) à 3 TRs,
- Le routeur (2,3) envoie les paquets aux nœuds (0,0) et (2,0),
- Le routeur (3,3) envoie les paquets aux nœuds (0,0) (1,0) et (1,0).

Dans ce TG, l'utilisateur doit définir la charge du réseau pour chaque bloc. Dans un objectif d'exploration, nous proposons un TG qui génère des paquets en faisant varier la charge. Ce TG est spécifié dans un package différent donné en Figure 3-12. Ce TG permet d'envoyer plusieurs paquets de tailles différentes d'un initiateur vers plusieurs destinataires en faisant varier les charges du réseau. Ces charges évaluées sont données par *idle_percent* allant de 10% à 100%.

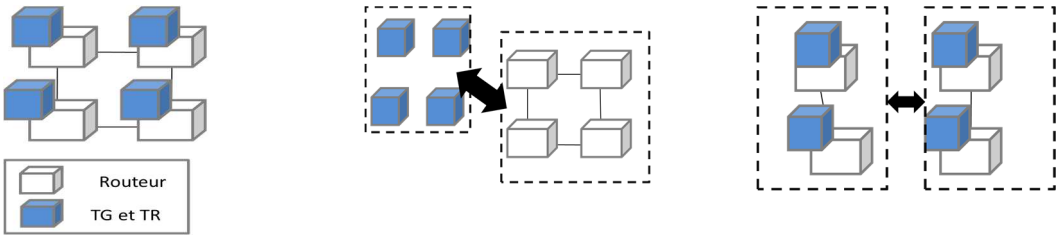
```
address_destination: regmetadeflit:= "00100001";
-- address of the destination Y=0010, X=0001
size_of_packet : integNORT := (6,0,0,0,0,0,0,0,0);
-- size of packet (number of the flit)
nbre_packet_send : integNORT := (4,0,0,0,0,0,0,0,0);
-- number of packets sent
idle_percent : integidle := (1,1,1,1,1,1,1,1,1,1);
-- idle percent (10%,20%,30%,40%,50%,60%,70%,80%,90%,100%)
Function idle_clk (nb: integer) return integidle;
Nbre_packet_received : integNORT := (0,0,0,0,0,4,0,0,0);
-number of packets received
constant nb_data_injection_rate : integer :=10;
```

Figure 3-12 : Spécification de transferts de données d'un initiateur vers plusieurs destinataires avec une variation automatique de la charge du réseau.

Grâce au package spécifiant le transfert de données, les TGs et TRs sont instanciés aux routeurs si ceux-ci envoient ou récupèrent des données. Si le nœud n'émet ou ne reçoit pas de données, aucun bloc d'émulation n'est instancié.

3.2.6 Spécification de partitionnement

Plusieurs solutions de partitionnement ont été étudiées, notamment les solutions les plus simples découlant de la structure du NoC. Deux solutions ressortent et sont représentées en Figure 3-13**.** La première consiste à séparer la structure du NoC des nœuds, les composants contenant soit des nœuds, soit la structure du NoC (b). Cette solution semble peu adaptée pour des tailles de NoC grande car cette solution nécessite un nombre de liens important directement lié aux paramètres du NoC (nombre de routeur et taille des flits). Une expérimentation portant sur l'implantation d'un NoC 6x6 de taille de flits de 32 bits sans blocs d'émulation sur Virtex 5 montre que le nombre d'entrée sortie nécessaire est de 2450 (alors qu'un FPGA Virtex ne possède que 480 blocs d'entrée sortie). La seconde consiste à conserver le nœud à son routeur et de découper la structure du NoC (c). La seconde solution est choisie car plus appropriée à des tailles de NoC élevées.



a) b) c)

Figure 3-13 : Solutions de partitionnement d'un NoC maillé. A partir d'un NoC original (a) : la 1ère solution consiste à séparer le NoC des nœuds (b) alors que la 2ème solution consiste à couper le NoC en gardant chaque nœud associé à son routeur (c).

Dans le cas d'un partitionnement multi-FPGA, l'utilisateur spécifie dans le package *Partitioning* le partitionnement du NoC. Dans ce package, il spécifie sur quel FPGA sont implantés les nœuds du NoC (champ *Partitionning*), les connections entre les FPGAs (champ *Connections*) et les liens externes qu'il utilise pour réaliser les connexions inter-FPGA. Un exemple de partitionnement est donné en Figure 3-14. Dans cet exemple, un NoC 4x1 est partitionné en 2 NoC 2x1 sur un seul FPGA. Le champ Partitionning indique les routeurs (0,0) et (0,1) sont porté sur le FPGA_0 et les routeurs (0,2) et (0,3) sont quant à eux portés sur le FPGA_1. La communication inter-FPGA se fait entre les routeurs (0,1) et (0,2) via un protocole de communication Aurora pour une communication bidirectionnelle. Le résultat du NoC partitionné et implanté sur 2 FPGAs est représenté à droite de la Figure 3-14.

```
** Partitioning
Switch (0,0) =FPGA_0
Switch (0,1) =FPGA_0
Switch (0,2) =FPGA_1
Switch (0,3) =FPGA_1
** Connections
Switch(0,1)=Switch(0,2)
** Aurora Links
Aurora_Link_0_in= Switch(0,1)
Aurora_Link_0_out= Switch(0,2)
```

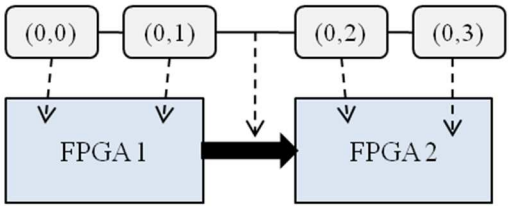


Figure 3-14 : Partitionnement d'un NoC 4x1 sur deux FPGAs. Le package contient le placement des routeurs sur les FPGAs, indique les routeurs connectés via un ou plusieurs liens externes ainsi que le sens des communications externes.

3.2.7 Flot de conception

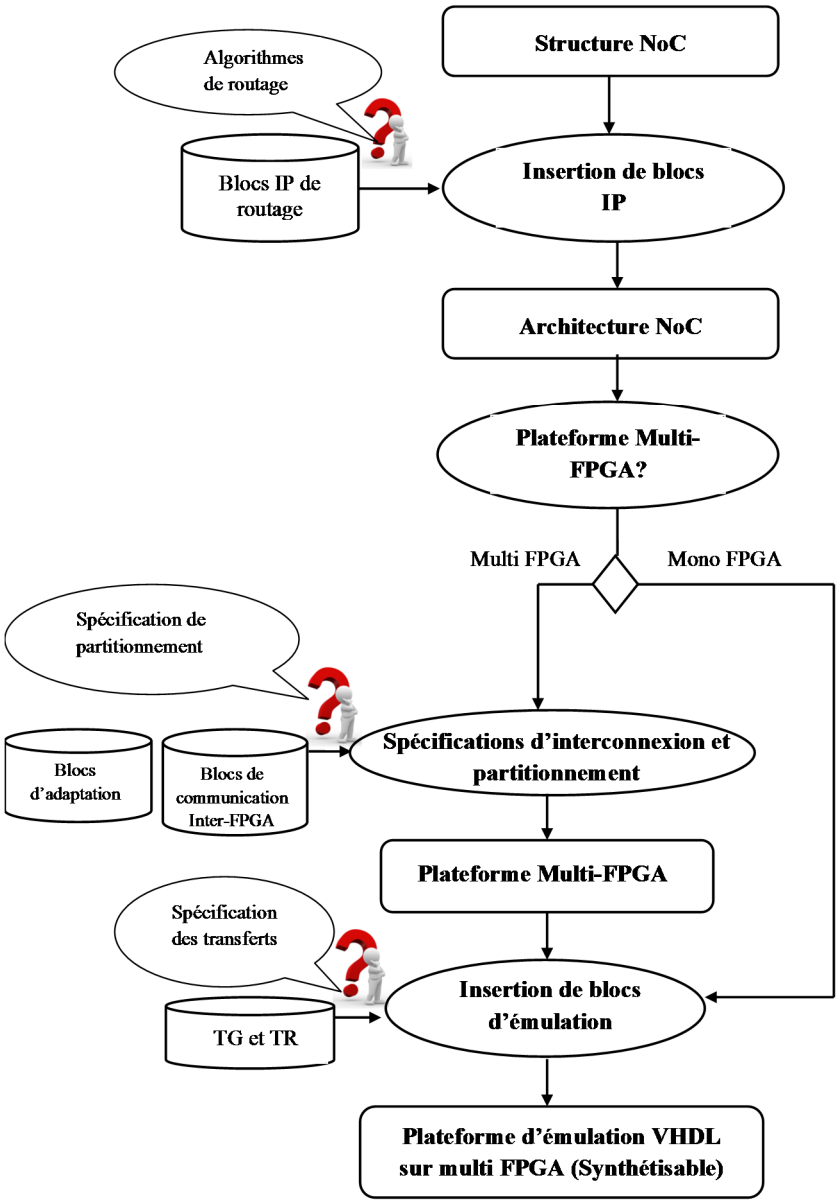


Figure 3-15 : Flot de conception pour la génération de la plateforme d'émulation sur FPGA.

Le flot de conception présenté Figure 3-15 permet la génération automatique de la plateforme d'émulation sur FPGA (mono-FPGA ou multi-FPGA). Le flot de conception est conçu dans un esprit de structure VHDL hiérarchique pour assurer l'instanciation des composants sur les différents niveaux du flot de conception. Plusieurs packages (routing, data_transfers, partitioning) sont conçus pour la spécification de l'architecture finale. Les blocs IP génériques sont disponibles sous forme

d'entité/architecture paramétrables et disponibles dans différentes bibliothèques à différentes étapes du flot.

Le flot a été également conçu dans un objectif de portabilité vers différentes structure de NoC. Celui-ci peut prendre différente structure de NoC en entrée si une structure HDL synthétisable contenant les routeurs, buffers, et les liens est disponible. Il peut être nécessaire de développer un bloc d'adaptation entre routeurs et blocs d'émulation si les protocoles de communication diffèrent de ceux utilisés dans la structure de NoC de référence. Dans ce cas, ces blocs d'adaptations supplémentaires peuvent être insérés dans le flot de conception lors de l'instanciation des blocs d'émulation.

Le point d'entrée du flot est la structure de NoC que l'utilisateur souhaite évaluer ou explorer. Il spécifie les différents paramètres du NoC comme le nombre de routeurs, la topologie et la taille de bus. La structure du NoC ainsi que son paramétrage n'est pas pris en charge par le flot de conception. Il appartient aux concepteurs du NoC de proposer les outils et structure adéquats. La première étape du flot concerne le choix de l'algorithme de routage qu'il souhaite utiliser. L'algorithme de routage est validé en affectant 1, les autres algorithmes de routages étant mis à 0 dans le package *routing*(Figure 3-16).

```
Package routing is
Constant routage_XY: std_logic :='1';
Constant routage_NFN: std_logic :='0';
.....
End routing;
```

Figure 3-16 : Exemple de sélection d'algorithme de routage dans le package routage.

A partir des spécifications du package routage, l'algorithme de routage est instancié dans le composant routeur. Nous supposons qu'une bibliothèque de blocs IP de routage contient les blocs des algorithmes de routage décrit en langage HDL synthétisable. Les blocs de routage instanciés permettent d'obtenir une architecture de NoC complète. L'utilisateur indique ensuite si cette dernière est implantée sur un ou plusieurs FPGAs. Dans le second cas, un package *partitionnement* multi-FPGA est spécifié.

L'étape suivant le partitionnement dans leflot est l'insertion des blocs d'émulation. A partir de l'architecture de NoC mono ou multi-FPGA, les blocs d'émulation sont insérés aux routeurs. L'utilisateur choisi le package *data-transfer* en fonction du type d'émulation comme cela a été détaillé en section 3.2.5. Un package permet d'émuler un scénario avec une variation de charge du

réseau automatique alors que le second package permet d'émuler tous les transferts de données d'une application de traitement d'image avec une charge de réseau prédéfinie. Une fois le package correctement rempli, les générateurs de trafic et récepteurs de trafic sont connectés au NoC pour obtenir la plateforme finale d'émulation.

Cette plateforme est ensuite synthétisée, placée et routée avec les outils traditionnels du marché. Si tous les packages sont renseignés correctement, la génération de la plateforme complète à partir de la structure du NoC d'entrée ne prend que quelques minutes. Les instanciations de blocs et les paramétrages sont pris en charge par le flot, évitant à l'utilisateur de posséder des connaissances en langage de description HDL et la conception d'architectures matérielle.

3.3 Evaluation des performances

Pour évaluer les performances d'un NoC, on considère le NoC comme une boîte noire où deux paramètres sont mesurés : la latence moyenne d'un paquet et le débit global dans le réseau. On ne considère pas dans notre cas les performances internes.

3.3.1 Latence

La mesure de la latence se fait au moyen d'une variable compteur insérée dans le paquet. La latence calculée correspond au temps de transfert des paquets entre l'émetteur et le récepteur. La donnée latence s'incrémente à chaque front d'horloge depuis l'émission du premier flit dans le réseau jusqu'à la fin de l'émulation qui se termine une fois tous les paquets arrivés à destination. Ensuite lorsque le dernier flit de ce même paquet est arrivé à destination et est sorti du réseau, le récepteur de trafic note la valeur du compteur à ce moment et fait la différence entre cette valeur et la valeur prise par le paquet, obtenant ainsi la latence du paquet. Cette opération est réalisée sur chaque paquet envoyé. Nous avons alors, d'une façon générale :

$$L_{moy} = \frac{\sum_{i=1}^{p} L_i}{P} \qquad \textbf{Équation 3-2}$$

Où L_{moy} est la latence moyenne (en nombre de cycles), L_i est la latence d'un paquet (en nombre de cycles) et Pest le nombre total de paquets.

Lorsque tous les paquets sont arrivés à destination, chaque récepteur de trafic fait la somme des latences pour chaque paquet reçu, la latence moyenne est alors obtenue en ajoutant les valeurs données par chacun des récepteurs de trafic divisé par le nombre total de paquets.

3.3.2 Mesure du débit

La mesure du débit est plus simple à mettre en œuvre que la mesure de la latence puisque il suffit de savoir à quel moment tous les paquets sont arrivés à destination, c'est-à-dire le temps de simulation. Grâce à cette donnée, nous pouvons calculer le débit global grâce à la formule suivante :

$$deb_{Glo} = \frac{ncpk \times pksize \times flitsize}{nsimc^{*}T} \qquad \textbf{Équation 3-3}$$

Avec deb_{glo} le débit global du réseau (en bps ou Mbps), *npck* le nombre de paquets total, *pcksize* la taille d'un paquet (en nombre de flits), *flitsize* la taille d'un flit, *nsimc* le temps total de simulation pour envoyer tous les flits de tous les paquets (en nombre de cycles) et *T* la période de l'horloge (en s).

3.4 Conclusion

Dans ce chapitre, nous proposons un flot de conception permettant la génération de plateformes d'émulation de NoC sur FPGA pour l'évaluation ou l'exploration. Le flot décrit dans une structure hiérarchique permet l'instanciation des blocs décrits en langage de description matériel générique. La plateforme est conçue autour des applications de traitement de signal et d'images. Il est possible d'évaluer ou d'explorer tous les scénarii de ces applications.

Ce flot est générique et ouvert. L'utilisateur peut prendre différents NoCs en point d'entrée et peut également enrichir les différentes bibliothèques sans modifier le principe général du flot. Toutes les cartes FPGA peuvent être ciblées et le NoC peut être déployé sur plusieurs FPGAs sans pénaliser les résultats temporels des évaluations ou des explorations qui ont été faites.

Chapitre 4. Implémentations et évaluation des performances des NoCs

L'objectif de ce chapitre est d'illustrer le principe du flot de conception présenté précédemment en implantant différentes plateformes d'émulations sur des architectures mono et multi-FPGA. L'implantation et l'évaluation des performances s'appuient sur le NoC Hermes. Nous allons tout d'abord présenter le NoC Hermes et son environnement logiciel pour la génération de la structure du NoC, les plateformes d'émulation générées puis réaliser quelques expérimentations. Ces expérimentations permettront de proposer une structure de NoC pour l'algorithme d'authentification d'œuvres d'art en fin de chapitre.

4.1 Le NoC Hermes

Le NoC Hermes est développé par l'équipe de F. Moraes au PUCRS (Pontificale Université Catholique de Rio Grande do Sul, Brésil) depuis 2003. Hermes est un NoC flexible, simple, de faible coût et qui satisfait pleinement à l'implémentation dans un circuit FPGA en assurant la communication entre les différents modules pour la conception de SoCs.

4.1.1 Structure du NoC Hermes

Le NoC Hermes possède une topologie maillée à deux dimensions, chaque nœud du réseau est constitué d'un routeur relié à un ou plusieurs blocs IP par l'intermédiaire du NI. Chaque routeur est identifié grâce à ses coordonnées XY, où X représente sa position horizontale et Y sa position verticale (Figure 4-1). Les origines des coordonnées se trouvent en bas à gauche. Le NoC Hermes intègre des routeurs, chaque routeur contient au maximum cinq ports bidirectionnels qui sont :

- Un port bidirectionnel qui relie chaque routeur au cœur, appelé le port « local ». Ce port sert à connecter les routeurs aux différents blocs IPs utilisés.

- Quatre ports bidirectionnels au maximum connectés vers les routeurs voisins appelés Nord, Sud, Est, Ouest pour assurer les échanges de données entre les routeurs voisins. L'architecture du routeur dépend de sa position dans le réseau.

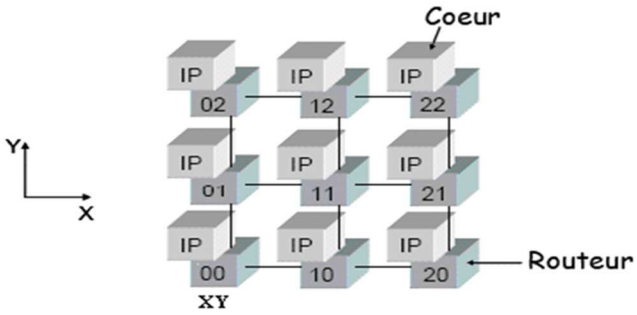


Figure 4-1 : Topologie maillée 2D du NoC Hermes.

Chaque routeur, représenté en Figure 4-2, possède des blocs buffer (fond en gris) pour chaque lien d'entrée et un bloc de contrôle (situé au milieu). Ce buffer est associé à chaque port d'entrée du routeur pour stocker temporairement les données en transit vers le port local. Le bloc de contrôle est constitué d'un arbitre et d'un aiguilleur (« routing logic »). L'arbitre sélectionne les ports d'entrées à traiter grâce à un algorithme de routage prédéfini. L'aiguilleur permet l'acheminement des données vers le port de sortie correspondant.

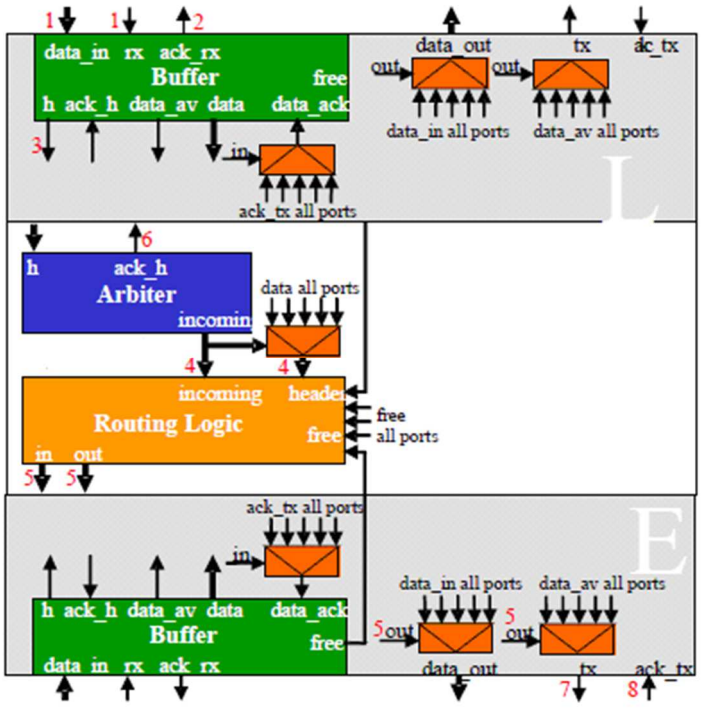


Figure 4-2 : Structure du routeur Hermes. Chaque entrée contient un buffer pour la mémorisation des paquets. Un bloc de routage et d'arbitrage transfère des paquets vers le lien correspondant.

Le contrôle de flux est indispensable pour la transmission des données entre les routeurs ou entre le routeur et le bloc IP. Deux types de contrôle de flux peuvent être choisis dans le NoC Hermes : le mode « Handshake » et le mode « Credit-based ».

- Le contrôle de flux « handshake » :il fonctionne en mode poignée de main 2 phases. Le routeur A envoie une requête au routeur B à chaque émission de paquet. L'émetteur attend la réception d'un acquittement qui vient de la part du récepteur B avant d'émettre une nouvelle donnée (Figure 4-3). Dans ce mode, chaque paquet est envoyé en deux coups d'horloge dans le cas d'un NoC synchrone.
- Le contrôle de flux « credit-based » : un signal de synchronisation est rajouté dans ce mode par rapport au mode précédent (Figure 4-4). Le routeur émetteur effectue une requête vers le routeur destinataire puis envoie une succession de paquets une fois la communication établie. Ces paquets sont stockés dans un buffer, le signal *credit* est renvoyé au routeur émetteur pour indiquer la taille encore disponible dans le buffer. Lorsque ce signal est à zéro, indiquant que le buffet est plein, le routeur émetteur arrête l'envoi des paquets. Dans ce mode, le paquet est envoyé en un coup d'horloge. Ce mode est plus difficile à mettre en œuvre et nécessite des buffers supplémentaires pour chaque routeur.

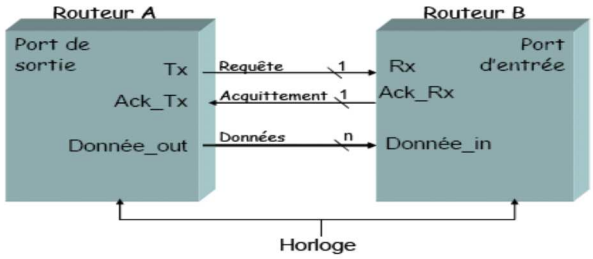


Figure 4-3 : Structure de communication entre deux routeurs en mode « handshake ».

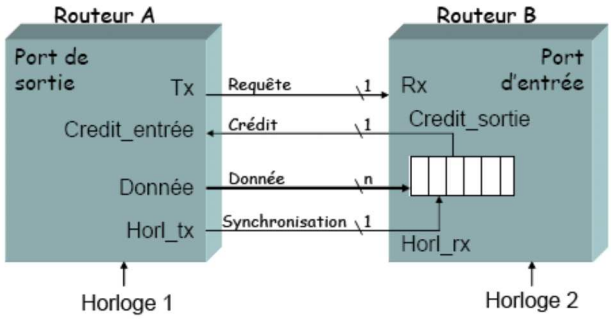


Figure 4-4 : Structure de communication entre deux routeurs en mode « credit-based ».

4.1.2 L'interface graphique pour la génération du NoC : ATLAS

L'environnement logiciel ATLAS est entièrement dédié à la génération et à l'évaluation de performances du NoC Hermes. ATLAS permet la génération de l'architecture NoC en VHDL synthétisable ainsi que la simulation des performances en SystemC au moyen de blocs synthétiques de trafic.

- **Génération de NoCs** : à partir des paramètres du NoC, ATLAS génère le code VHDL synthétisable de l'architecture de communication en vue d'une implantation directe sur FPGA. Les paramètres d'entrée sont le choix du contrôle de flux (« handshake » et « credit-based »), le nombre de « Virtual Channel » en mode « credit-based », la taille du NoC, la taille des flits (8, 16, 32 ou 64 bits) et la taille des buffers pour chaque routeur (4, 8, 16 et 32 flits).

- **Simulations et tests :** ATLAS peut également générer, à la demande de l'utilisateur, le « testbench » en SystemC relatif au NoC pour des simulations fonctionnelles et temporelles hors ligne au moyen des outils de simulation traditionnels. Les paramètres de la simulation consistent à définir les générateurs de trafic, la fréquence d'horloge du NoC ou de la fréquence pour chaque routeur, la sélection des émetteurs, récepteurs et liens entre eux (quel émetteur vers quel récepteur), le nombre de paquets à envoyer au(x) routeur(s) cible(s), la taille des paquets (plus précisément le nombre de flits par paquet, ainsi que la distribution des données dans les routeurs).

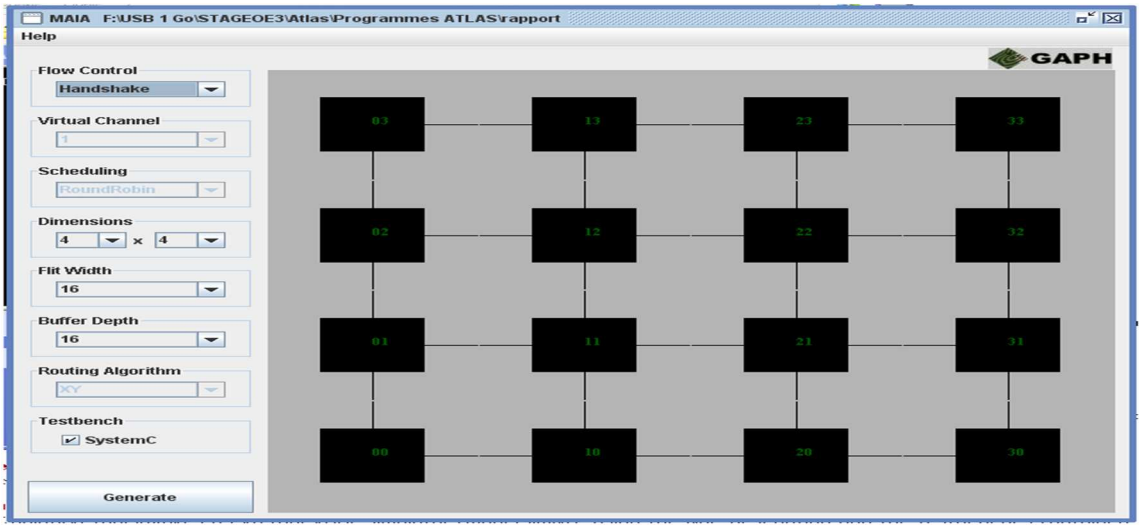


Figure 4-5 : Le logiciel ATLAS. A gauche, l'utilisateur définit les paramètres du NoC et choisit ou non de générer le testbench en SystemC. A droite, la vue du NoC généré.

Toutes ces caractéristiques sont alors prises en compte pour la génération du code VHDL synthétisable et de ses modèles de simulation et test en SystemC. Les fichiers VHDL synthétisables obtenus sont les suivants :

- Un package contenant tous les types et sous-types permettant de dimensionner le NoC ;
- Une entité décrivant le fonctionnement des ports d'entrée et de leurs buffers ;
- Une entité décrivant le fonctionnement du bloc logique (arbitre et aiguilleur) etdonc l'algorithme de routage ;
- Des entités pour chaque type de routeurs : routeurs centraux, routeurs en bas à droite, à gauche, en haut à droite,… ;
- L'entité globale du NoC.

En ce qui concerne l'évaluation des performances du NoC, l'outil génère les résultats obtenus par la simulation sous forme de tableaux et de graphiques. Grâce à cet outil, il est possible de voir comment évoluent plusieurs paramètres tels que la latence moyenne, le débit global du NoC, le débit dans chaque routeur, le nombre de cycles nécessaire pour transmettre un flit dans chaque routeur et le nombre de flits transmis par unité de temps.

Le logiciel ATLAS est donc un environnement de développement bien approprié pour l'étude des performances du NoC HERMES selon ses paramètres. Seul l'outil ATLAS a été réellement utilisé afin d'obtenir la structure VHDL du NoC Hermes. Une plateforme d'exploration de paramètres et d'analyse des performances a été développée pour réaliser ces tâches dans le ou les composants FPGA, remplaçant ainsi l'environnement de simulation SystemC.

4.2 Objectifs et outils pour l'évaluation et l'exploration

Le flot de données proposé permet de réaliser des évaluations de performances ainsi que de dimensionner le NoC pour des applications ciblées. Plusieurs explorations sont menées dans ce chapitre. Nous distinguons les explorations sur cible mono-FPGA puis sur cibles multi-FPGA. Dans un contexte général, l'ensemble de ces expérimentations ont pour objectifs :

- De vérifier que certaines règles énoncées pour le NoC s'appliquent sur circuit FPGA (impact des algorithmes de routage, du dimensionnement du NoC sur les performances…),
- De connaitre les limites en termes de ressources du NoC sur un ou plusieurs FPGAs,
- De trouver des mécanismes permettant d'évaluer les performances temporelles du NoC et ceci indépendamment du ou des FPGAs,

- De définir des modèles de scénario réaliste associé à un NoC sur FPGA,
- Enfin de proposer une architecture de NoC pour l'algorithme d'authentification étudié.

Les expérimentations sont réalisées au moyen d'un outil dédié à la génération de la plateforme d'émulation, outil développé au laboratoire avec une interface graphique ainsi que des plateformes de prototypage FPGA.

4.2.1 L'interface graphique NoCGen

Une interface graphique, Figure 4-6, a été développée sous Python pour le flot de conception proposé en chapitre 3. Il s'agit d'un utilitaire graphique, qui à partir des entités/architectures et des packages VHDL, génère la plateforme d'émulation complète. L'utilisateur entre les paramètres demandés dans les packages du flot pour générer le code VHDL synthétisable de la plateforme complète d'émulation. Cette interface ainsi que toutes les bibliothèques associées, appelée NoCGen est disponible en opensource (http://tima-sls.imag.fr/www/research/nocgen).

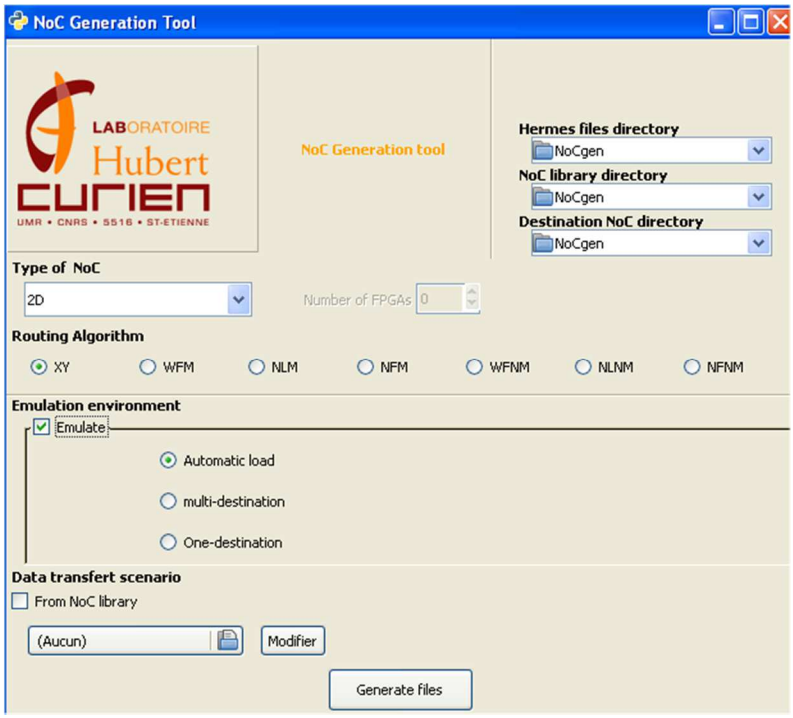


Figure 4-6 : Interface graphique NoCGen développée autour du flot de conception.

4.2.2 Les architectures Xilinx et les outils associés

Nous utiliserons les plateformes d'évaluation ML506 de Xilinx [75]. Chaque carte possède un FPGA Xilinx Virtex 5 XCV5VSX50 contenant chacun 32 640 registres et 32 640 LUTs. La ML506 permet d'exploiter des liens séries haut débit en utilisant les communications Rockets IO GTP. Les blocs Aurora des RocketsIO sont utilisés lors des déploiements du NoC sur plusieurs FPGAs.

L'outil logiciel pour la génération et l'implantation sur FPGA est Xilinx ISE 10.1 intégrant l'outil de synthèse XST. La génération des blocs IPs pour la communication Aurora est réalisée par l'outil intégré de Xilinx (Logicore). Les simulations fonctionnelles et temporelles sont réalisées avec modelsim 6.5.

4.3 Etude des performances sur mono-FPGA

L'évaluation du NoC mono-FPGA permet les explorations et analyses suivantes :

- Le dimensionnement du NoC : l'impact des paramètres du NoC en fonction du nombre de nœuds, de la taille des flits, de la taille et du nombre de paquets,
- L'impact de l'algorithme de routage de type Turn-Model utilisé,
- L'impact des scénarii choisis : le nombre de TG et de TR, la position des blocs, la taille des paquets…

Pour chaque expérimentation, les ressources et les performances temporelles sont étudiées.

4.3.1 Exploration du nombre de nœuds.

Dans cette expérimentation, seule l'architecture du NoC est implantée, les blocs d'émulation ne sont pas intégrés. La taille du NoC, plus précisément le nombre de routeurs varie et ceci pour différentes tailles de flits. La taille des buffers est fixée à 32 flits quelque soit la taille du NoC. Les ressources (LUTs et registres) utilisées pour les différentes tailles de NoC sont représentées Figure 4-7 et Figure 4-8.

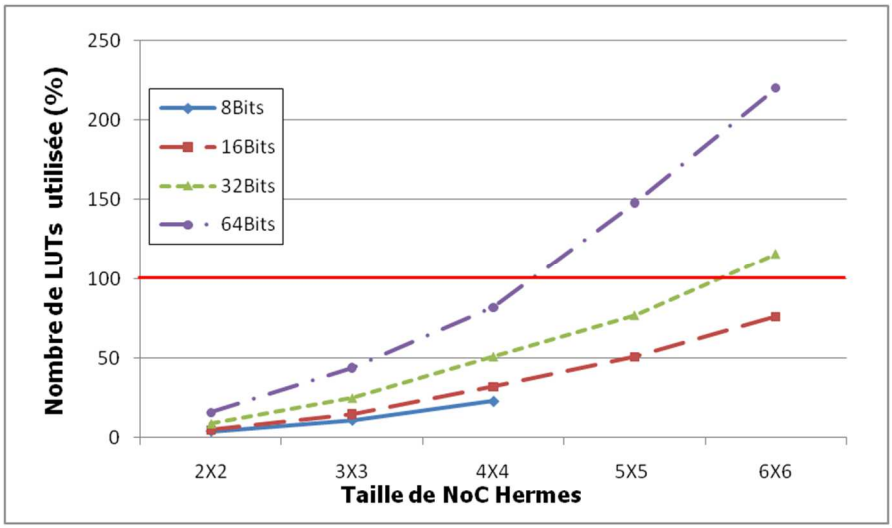


Figure 4-7 : Nombre de LUTs utilisées sur Xilinx Virtex 5 pour différentes tailles du NoC Hermes.

On observe qu'il n'est pas possible d'utiliser un NoC 4x4 possédant une taille de flit de 8 bits. La taille du flit dépend de la taille du NoC, ceci étant dû à la structure des flits utilisée pour le NoC Hermes qui met l'adresse du nœud de destination sur une moitié de flit. Ainsi des paquets de taille de flits de 8 bits pourront adresser un NoC de taille maximale 3x3 (la taille des coordonnées de la destination se faisant sur 2 bits).

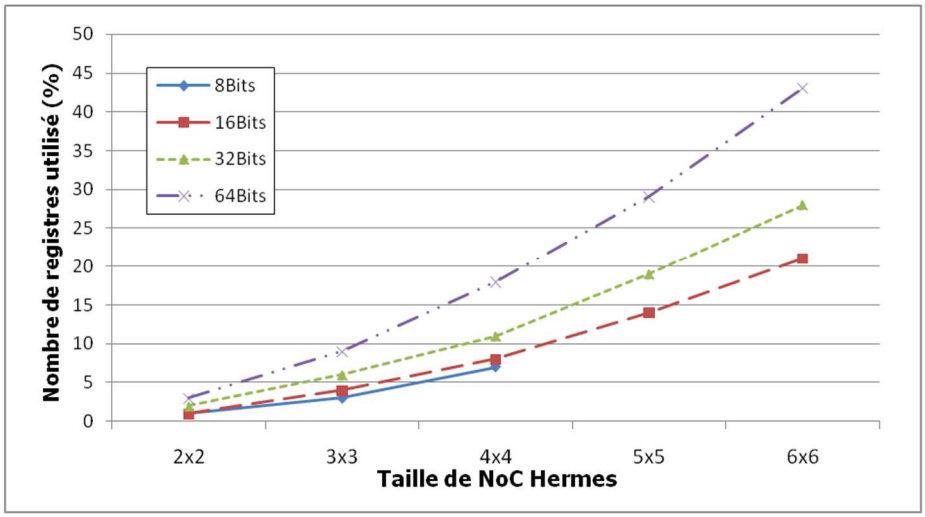


Figure 4-8 : Nombre de registres utilisés par Hermes sur Xilinx Virtex 5 pour différentes tailles de NoC.

On constate que le nombre des ressources dépend de la taille du NoC ainsi que de la taille des flits. Selon la Figure 4-8, un NoC 4x4 de taille de flit de 64 bits utilise presque la totalité des ressources en LUTs du FPGA, environ 90% des LUTs disponibles alors que 30% des LUTs est utilisé pour un NoC 4x4 de taille de flit de 8 bits. Le nombre de routeurs peut augmenter si la taille du flit diminue. Ainsi il est possible d'implanter sur un Virtex 5 un NoC 5x5 pour une taille de flit de 32 bits et un NoC 6x6 pour une taille de flit de 16 bits. Il est également observé que les LUTs sont les ressources limitatrices dans la taille du NoC.

4.3.2 Impact des blocs d'émulation

L'idée est d'analyser l'impact des blocs d'émulation sur l'analyse des ressources de la plateforme d'émulation. Pour ce faire, nous choisissons un NoC 4x2.Les ressources, registres et LUTS, de ce NoC 4x2 avec et sans blocs d'émulation sur chaque routeur sont données respectivement sur la Figure 4-9.Les deux types de blocs d'émulation sont connectés sur chaque routeur. Ainsi 8 TGs et 8 TRs sont placés sur les 8 routeurs du NoC.

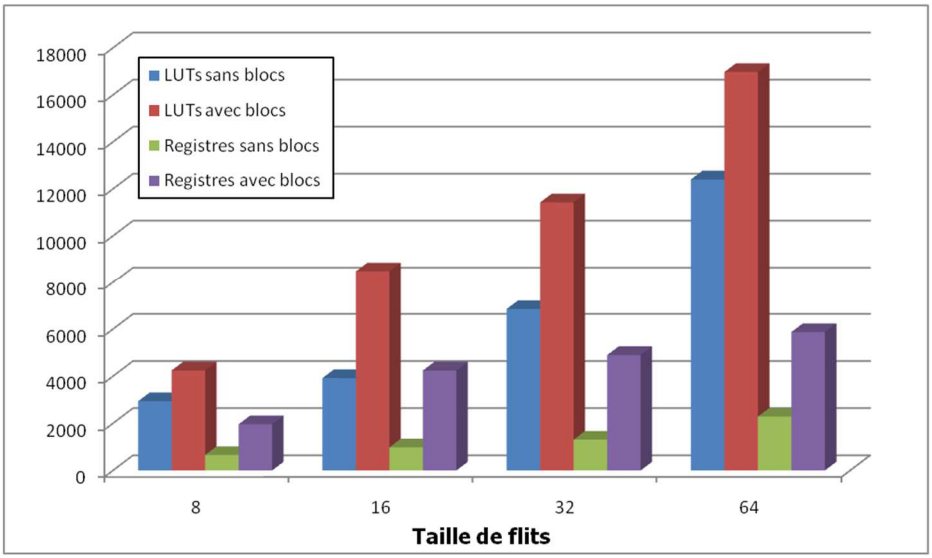


Figure 4-9 : Nombres de ressources pour un NoC 4x2 sans blocs d'émulation et avec blocs d'émulation en fonction de la taille des flits.

On observe que pour une petite taille de NoC (type réseau maillé 4x2), l'ajout de blocs d'émulation accroit de manière non négligeable le nombre de LUTs et de registres, et ceci même si le nombre de blocs d'émulation est faible. Dans notre cas, le nombre de registres est approximativement multiplié par 3 (entre 2,57 et 4,33) et le nombre de LUTs est approximativement multiplié par 2 (entre 1,36 et 2,16) pour un ajout de 8 TG et 8 TR. C'est un résultat sans surprise car les TGs et TRs, même de petite taille, représente utilisent néanmoins entre 200 et 250 registres et 360 et 390 LUTs. Même si ceci représente moins de 1% des ressources du FPGA par blocs d'émulation, l'intégration de plusieurs blocs augmente le nombre de ressources (Tableau 4-1).

Tableau 4-1 : Ressources utilisées des blocs TG et TR sur FPGA.

	Registres	LUTs
TG	214/32640 (0%)	362/32640 (1%)
TR	240/32640 (0%)	382/32640 (1%)

4.3.3 Exploration de la taille des paquets, des flits et du nombre de blocs d'émulation

Dans cette partie, nous allons étudier l'impact du choix de la taille des paquets et des flits ainsi que le nombre de générateurs de trafic (Initiateur) et de récepteurs de trafic (Récepteur). Deux scénarii sont choisis pour un NoC de taille 3x3 (Figure 4-10). Le premier présente un déséquilibre important entre le nombre d'initiateurs et de récepteurs, 8 initiateurs envoyant des données à un seul récepteur, le second scénario ayant des flux plus équilibrés avec 2 initiateurs envoyant des données sur un

récepteur situé sur la même ordonnée. Le premier scénario est basé sur des fonctions d'images bas niveau, type filtre pour lequel des flux de données important peuvent se présenter. Le second scénario est basé sur des opérations de type moyen niveau, telle que les projections couleurs ou calcul de distance couleur ou spectral.

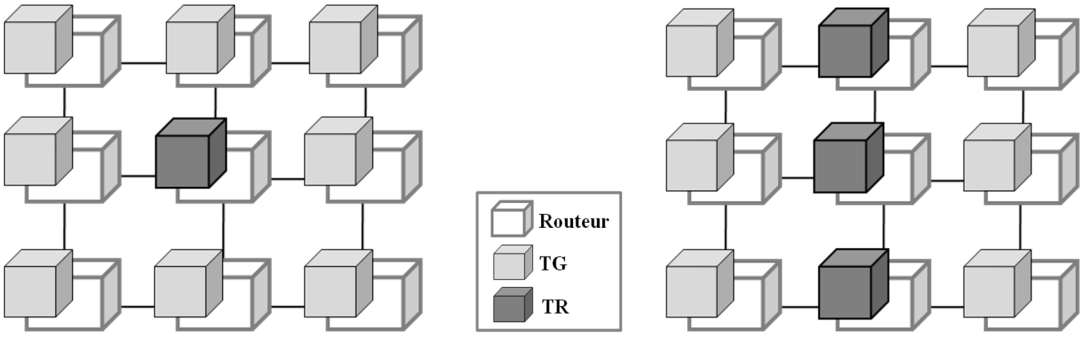


Figure 4-10 : Deux scénarii utilisés pour l'exploration de paramètres du NoC (scénario 1 à gauche et scénario 2 à droite).

La première expérimentation concerne la variation de la taille des flits sur les performances temporelles (la latence et la bande passante) avec des envois de 100 paquets de 72 flits pour chaque initiateur. La fréquence du FPGA est fixée à 27MHz. Les résultats des performances (latence moyenne et débit global) sont donnés Figure 4-11 et Figure 4-12 pour le scénario 1 et Figure 4-13et Figure 4-14pour le scénario 2.

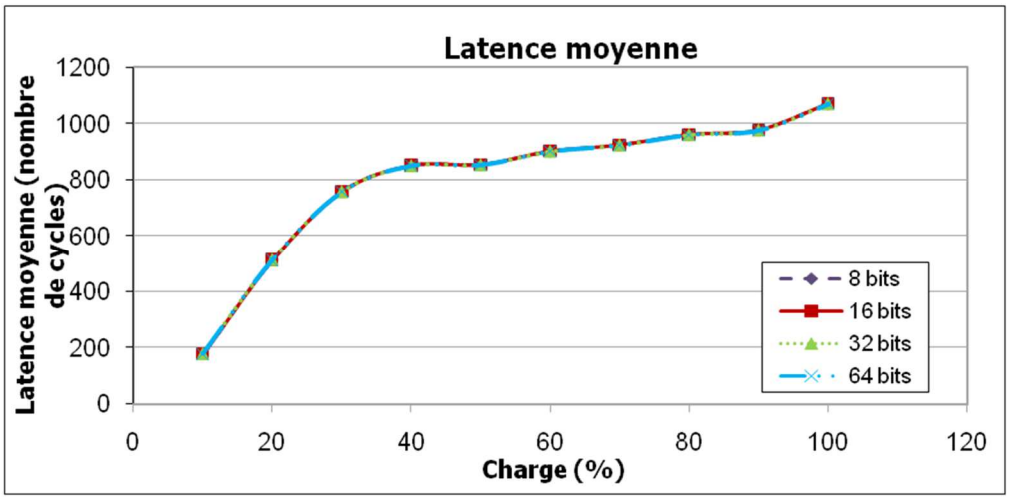


Figure 4-11 : Latence moyenne pour le scénario 1 sur le NoC 3x3 pour différentes charges.

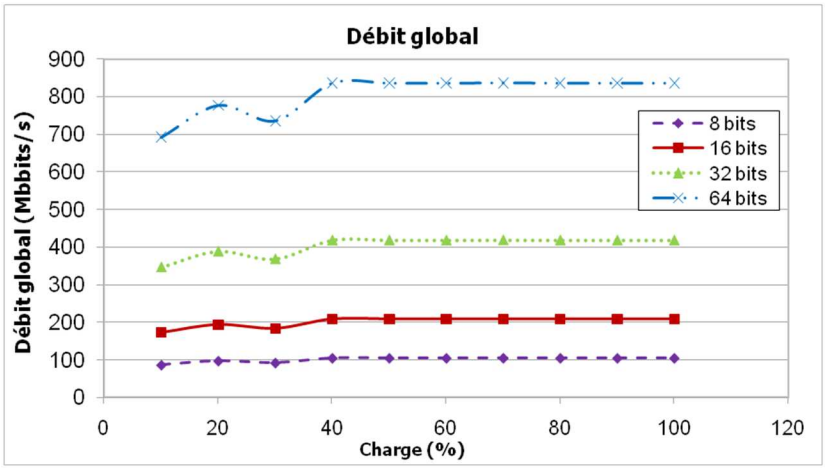


Figure 4-12 : Débit global pour le scénario 1 sur le NoC 3x3 pour différentes charges.

On observe que le scénario 1 n'est pas adapté à l'architecture proposée. En effet, on constate que le point de saturation est en dessous d'une charge de 10%. (Figure 4-12) Le point de saturation définit à partir de quelle valeur de charge de réseau les communications sont saturées. Au dessus de ce point, les données sont bloquées (ou ralenties) au niveau des routeurs, augmentant la latence de la communication. Dans ce scénario 1, le nombre d'initiateurs est élevé par rapport au nombre de récepteurs et le réseau ne peut supporter cette densité de transfert. Dans la zone de saturation, le débit est faible et la latence moyenne élevée. Le scénario 2 est quant à lui plus adapté à l'architecture, la zone de saturation se trouvant aux environs de 50% de la charge du réseau (Figure 4-13). L'utilisateur peut donc envoyer des paquets avec une charge inférieure à 50% pour utiliser au maximum les performances du NoC.

On observe que la latence totale n'est pas influencée par la taille du flit alors que le débit global est directement fonction de ce paramètre (Figure 4-15et Figure 4-16), en zone de saturation ou non.

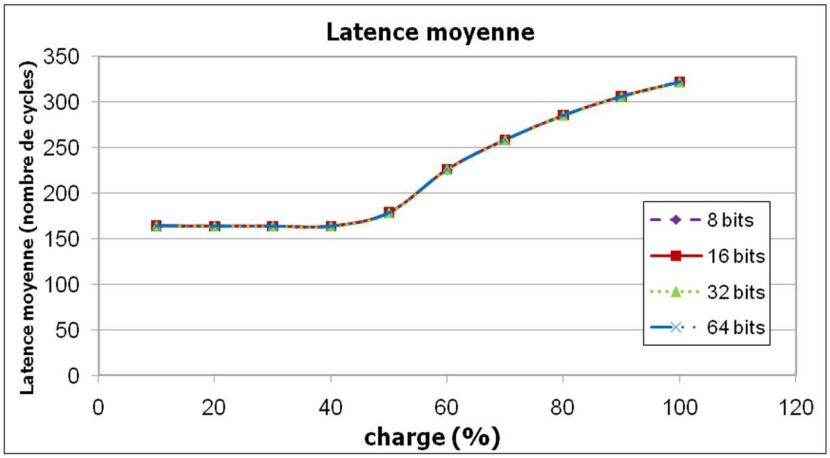


Figure 4-13 : Latence moyenne pour le scénario 2 sur le NoC 3x3 pour différentes charges.

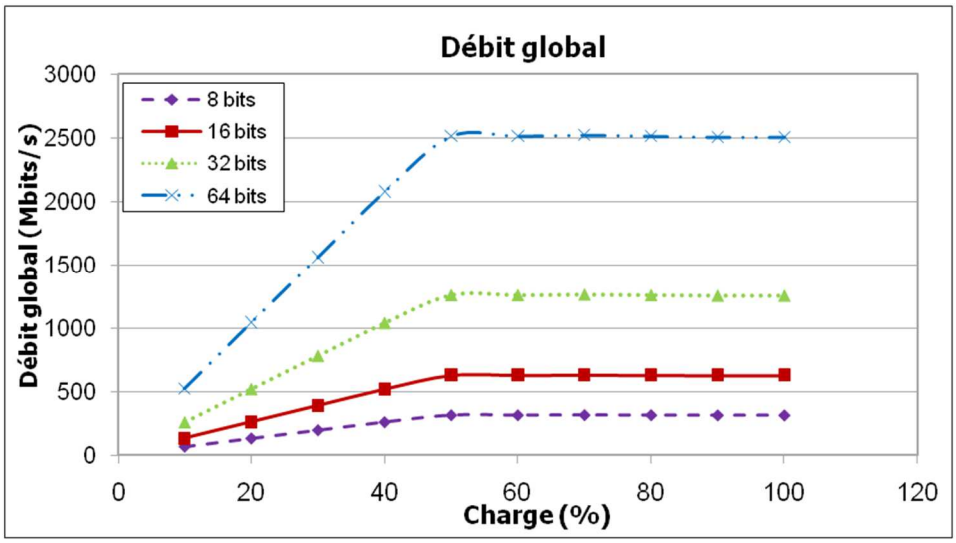


Figure 4-14 : Débit global pour le scénario 2 sur le NoC 3x3 pour différentes charges.

Dans notre expérimentation, on peut augmenter le débit d'un facteur 7 en augmentant la taille du flit par 8 (en passant de 8 bits à 64 bits). L'utilisateur peut augmenter le débit de la communication en augmentant la taille des flits au détriment d'une augmentation du nombre de ressources comme cela a été démontré en 4.2.2.

Une analyse des performances temporelles est également faite en fonction de la taille du paquet, plus précisément le nombre de flits par paquets variant de 12 flits par paquet jusqu'à 202 flits par paquet. Pour cette analyse, seul le scénario 2 sera utilisé, le scénario 1 étant inapproprié. La latence totale et le débit global sont donnés en Figure 4-15 et Figure 4-16.

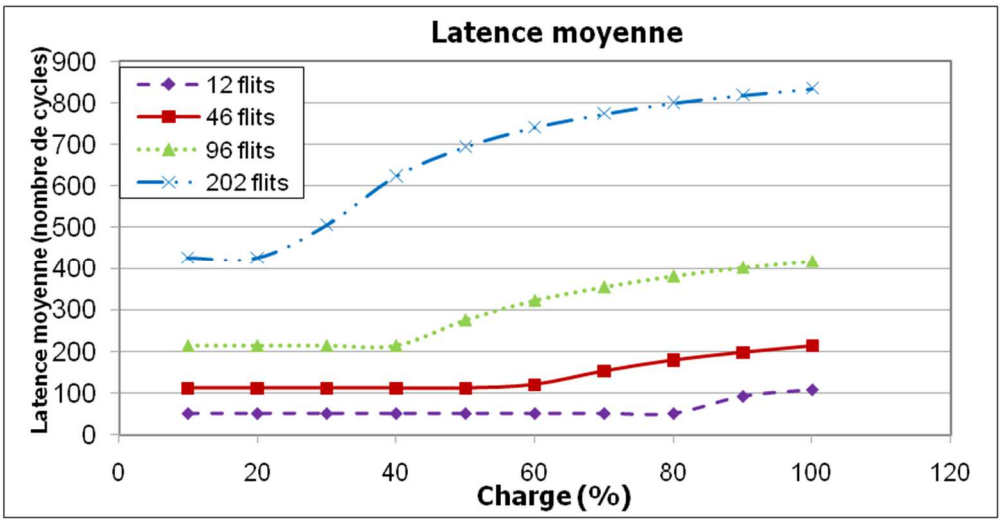


Figure 4-15 : Latence moyenne pour le scénario 2 sur le NoC 3x3 en fonction de la taille de paquet.

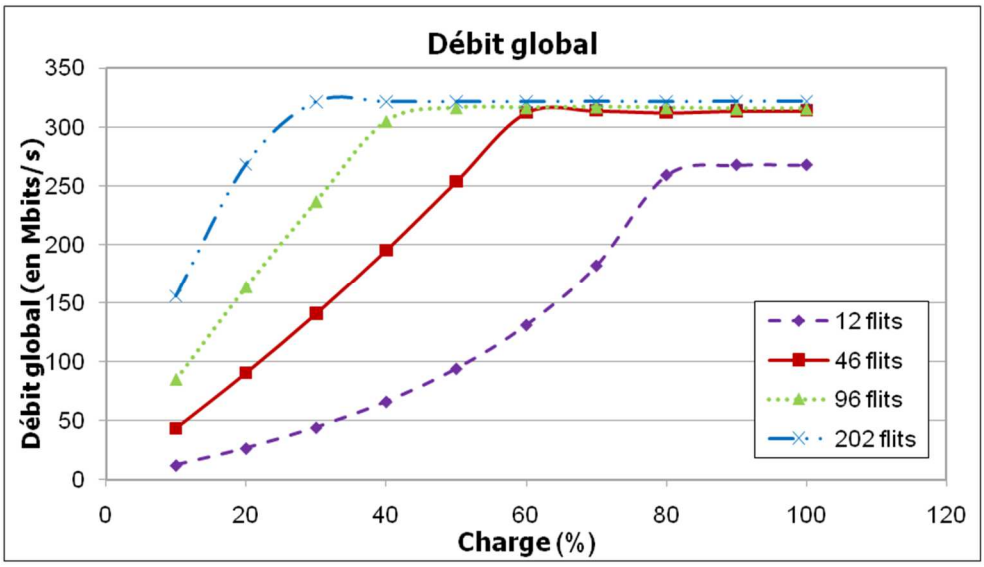


Figure 4-16 : Débit global pour le scénario 2 sur le NoC 3x3 en fonction de la taille de paquet.

On observe que la latence moyenne dépend de la taille du paquet mais reste constante pour une taille constante de paquet quelque soit la charge du réseau en dessous du point de saturation (Figure 4-15).La latence d'un flit est de 2 cycles d'horloge en dessous du point de saturation. Plus on augmente la taille du paquet, plus la latence moyenne est élevée. De plus, l'augmentation de la taille du paquet modifie le point de saturation qui diminue vers des charges de réseau plus faible (Figure 4-16). Le point de saturation se trouve à 80% de la charge pour des tailles de paquets de 12 flits alors que celui-ci se trouve à 20 % pour une taille de paquets de 202 flits. L'augmentation de la taille du paquet permet d'augmenter le débit avec un point de saturation qui varie.

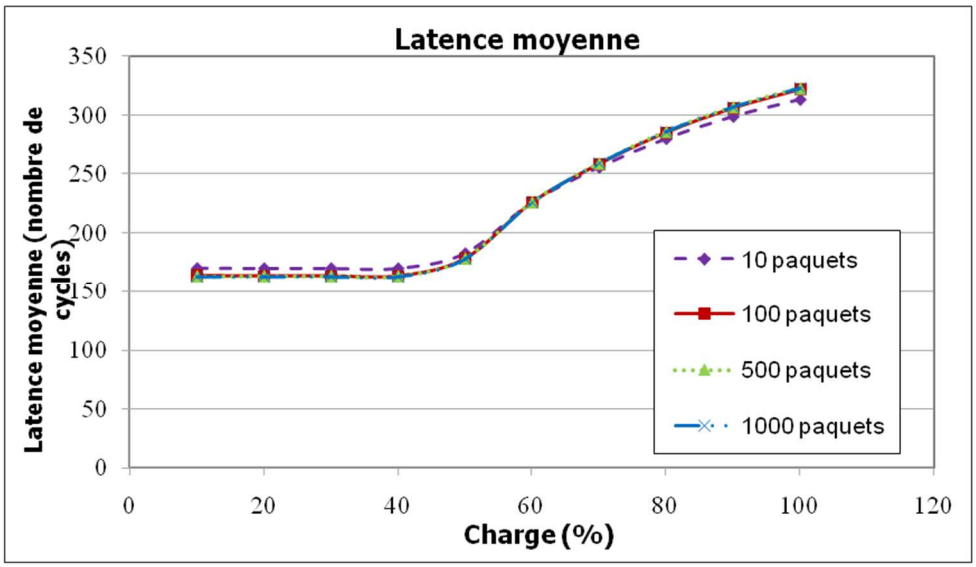


Figure 4-17 : Latence moyenne pour le scénario 2 sur le NoC 3x3 en fonction du nombre de paquets.

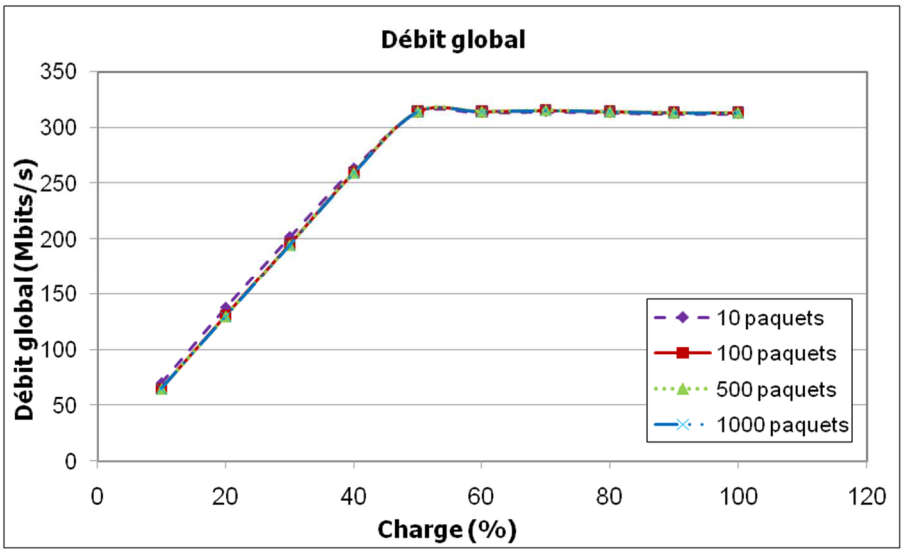


Figure 4-18 : Débit global pour le scénario 2 sur le NoC 3x3 en fonction du nombre de paquets.

Une dernière expérimentation porte sur l'impact de la taille des paquets sur les performances temporelles représenté Figure 4-17 et Figure 4-18. Le nombre des paquets n'a aucune influence sur la latence totale ou le débit global. Ce résultat est sans surprise car les estimations temporelles se font sur un paquet, et non pas sur la totalité des paquets envoyés.

4.3.4 Exploration des différents algorithmes de routage et du placement des blocs d'émulation

Nous étudions ici l'impact du choix des différents algorithmes de routage de type Turn-Model proposés dans l'architecture NoC Hermes ainsi que la position des blocs d'émulation. Les algorithmes de routage utilisés sont XY, West First (WFM), North-Last (NLNM) et Negatif first(NFNM). Tous ces algorithmes de routage sont conçus sous forme d'IP VHDL pour une insertion immédiate dans les routeurs. Le NoC Hermes est un NoC 4x4 avec 4 TGs et 3 TRs (Figure 4-19).

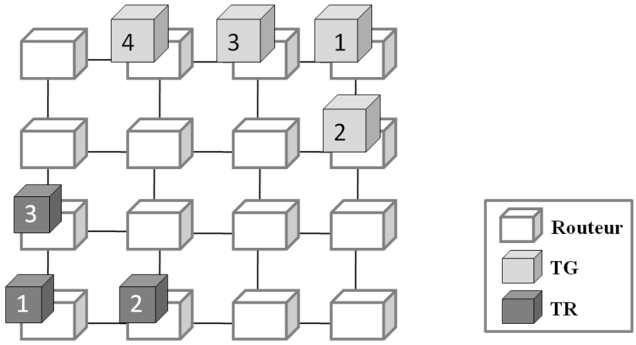


Figure 4-19 : Les scénarii utilisés pour l'exploration des algorithmes de routage et de la position des blocs d'émulation pour le NoC Hermes.

La première expérience représentée Figure 4-20 et Figure 4-21 montre le nombre de registres et de LUTs en fonction de la taille de l'architecture NoC pour plusieurs algorithmes de routage.

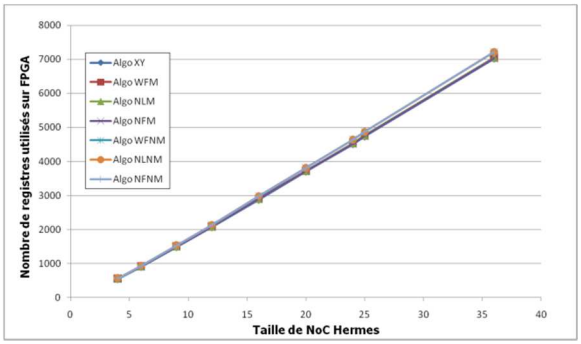


Figure 4-20 :Nombrede registres utilisés pour différents algorithmes de routage sur Hermes.

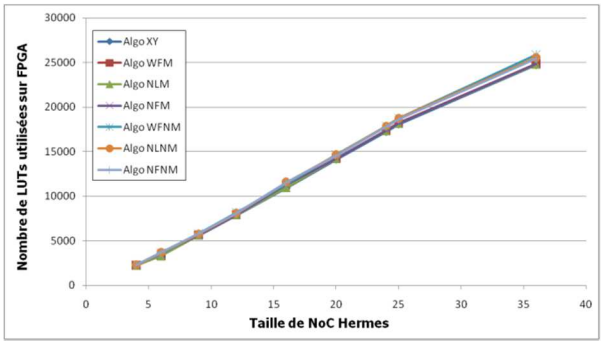


Figure 4-21 : Nombre de LUTs utilisées pour différents algorithmes de routage dans Hermes.

Les ressources concernent seulement l'architecture du NoC, les blocs d'émulation ne sont pas considérés. Le nombre de LUTs et de registres est approximativement le même pour tous les algorithmes de routage. Afin d'analyser plus finement les différences entre les algorithmes de routage, une différence entre deux algorithmes, l'algorithme XY et l'algorithme NLNM est faite (Figure 4-22).

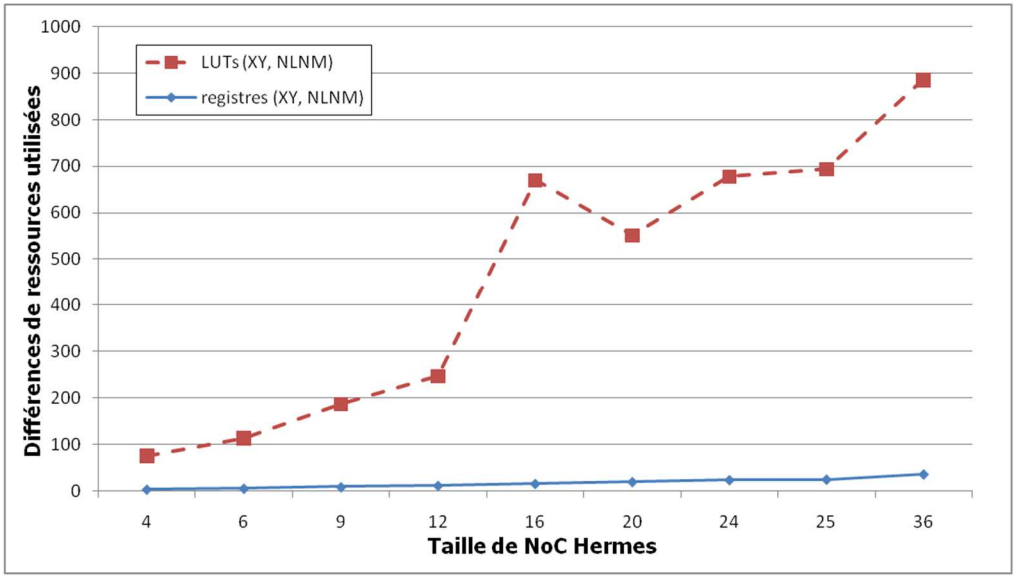


Figure 4-22 : Différence du nombre de ressources entre les algorithmes de routage XY et NLNM.

Ces deux algorithmes de routage sont choisis car ils utilisent respectivement le plus petit nombre et le plus grand nombre de ressources. La différence la plus importante dans le nombre de LUTs est de 71pour un NoC à 4 nœuds et 850pour un NoC à 36 nœuds. Ce nombre semble élevé mais reste cependant insignifiant par rapport au nombre de LUTs requis pour le NoC lui-même (cela

représente respectivement 3,2% et 3,4% de LUTs en plus).La différence quant au nombre de registres est inférieure par rapport au nombre de LUT (18 pour un NoC à 4 nœuds et 196pour un NoC à 36 nœuds), représentantentre2,79%et 3,65% de registres en plus. Par conséquent, il est possible de confirmer que le choix de l'algorithme de routage n'a pas vraiment d'impact sur le nombre de ressources du NoC Hermes sur FPGA comme cela peut être affirmé dans la littérature. Les avantages et inconvénients sont plus significatifs dans les performances temporelles. Cette analyse est menée ci-dessous via les analyses temporelles utilisant les scénarii donnés en Figure 4-19.

Pour les expériences suivantes, 4 TG (TG1 àTG4) et 3TR (TR1 à TR3) sont utilisés comme illustré à la Figure 4-19. Les transferts de données sont basés sur40 paquets et 500flitspar paquet avec une taille de 16bits par flit. Les algorithmes de routage utilisés sont XY, NFM et NFNM. La position des nœuds est choisie pour assurer le transfert de données de droite à gauche pour comparer différents algorithme de routage (comme NFNM utilise l'algorithme de routage XY pour les transferts de données gauche à droite, il serait inutile de faire de telles analyses). Une première analyse se fait sur l'envoi d'un TG vers plusieurs TRs et en faisant varier la position du TG sur les nœuds 1, 2 et 3. Les résultats de performances de l'envoi des paquets pour le TG1 avec l'algorithme de routage XY et NFNM sont présentés Figure 4-23. Les résultats similaires sont donnés Figure 4-24 pour le TG2 ainsi que Figure 4-25 pour le TG3.

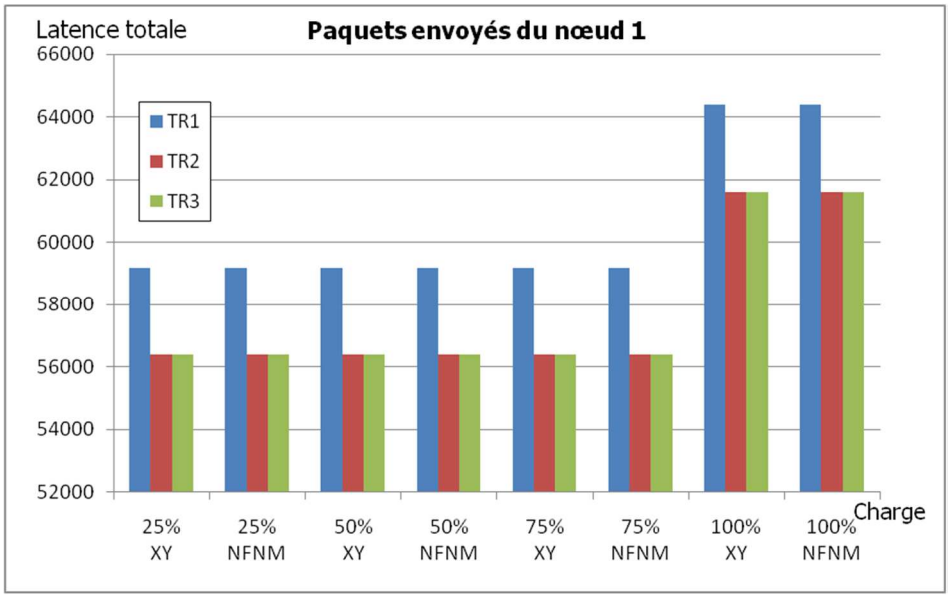


Figure 4-23 : Latence totale pour chaque destinataire pour des paquets envoyés du nœud 1avec une charge réseau variable avec l'algorithme de routage XY et NFNM.

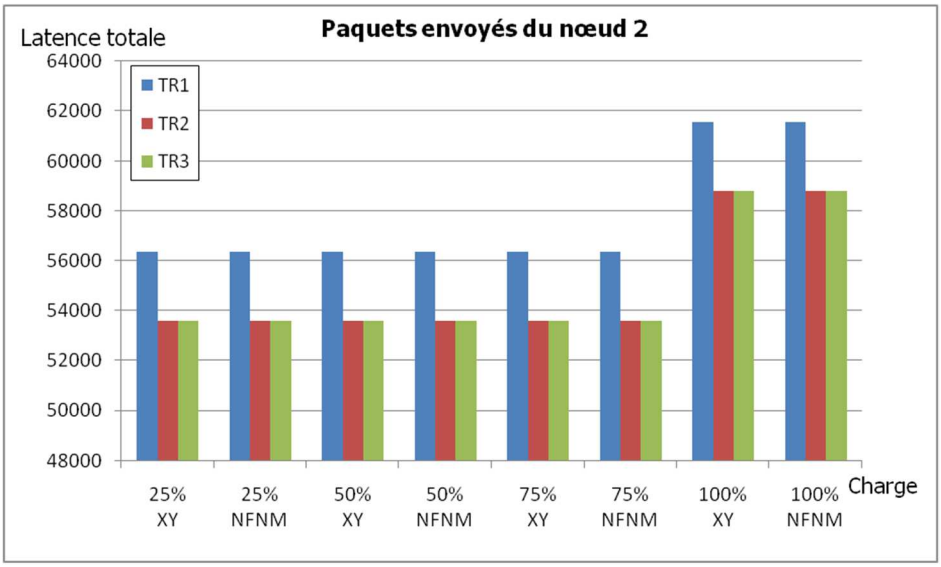


Figure 4-24 : Latence totale pour chaque destinataire pour des paquets envoyés du nœud 2 avec une charge réseau variable avec l'algorithme XY et NFNM.

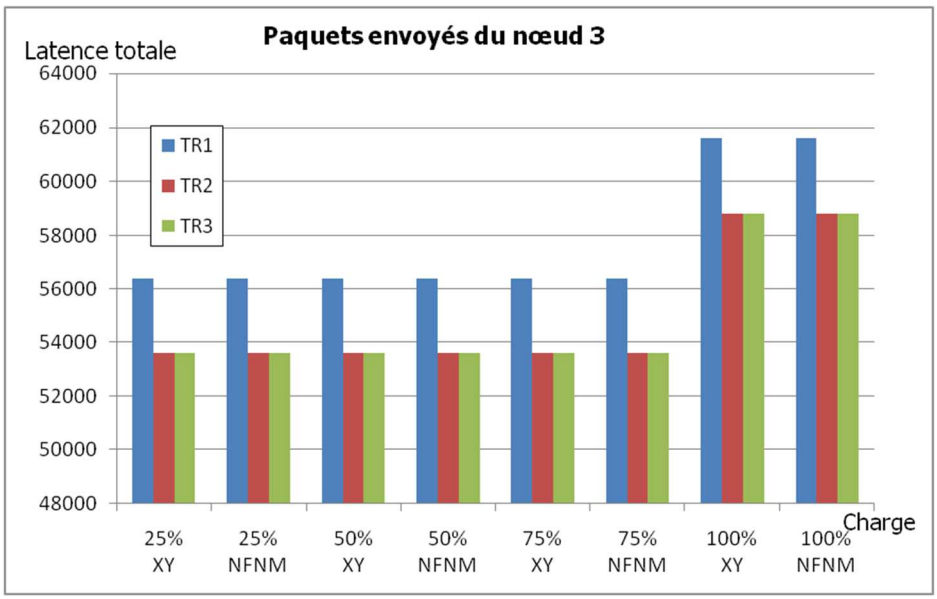


Figure 4-25 : Latence totale pour chaque destinataire pour des paquets envoyés du nœud 3 avec une charge réseau variable avec l'algorithme XY et NFNM.

De ces figures en ressortent quelques analyses pertinentes. La latence dépend de la position des nœuds TG et TR car elle définit le nombre de sauts (« hops »= nombre de routeurs traversés) pour chaque TR. Ainsi la latence de la transmission est plus longue pour TR1 que TR2 et TR3, le nombre de nœuds routeurs à traverser étant plus élevé. La position des TR n'as pas d'influence sur la latence, on constate que les temps pour des envois de TR2 et TR3 sont identiques. Les latences sont identiques en dessous du point de saturation. Ainsi on peut estimer le point de saturation entre

75% et 100%, une augmentation de la latence apparaissant pour une charge de réseau de 100%. Enfin, les transferts étant simples et dans un cas d'un initiateur vers plusieurs récepteurs, l'influence de l'algorithme de routage est nulle, l'algorithme n'ayant pas de goulot d'étranglement à gérer.

L'analyse suivante concerne l'envoi simultané des paquets des 4 TGs vers les 3 TRs en modifiant l'algorithme de routage (XY, NFM, NFNM) pour différentes charge de réseau. On constate que l'algorithme XY donne la meilleure latence pour la récupération des données vers TR0 avec des charges de 25% et 50% alors que l'algorithme NFM donne de meilleurs résultats pour une charge de 75%. L'algorithme NFNM donne les moins bonnes latences pour TR0 quelle que soit la charge. Les différences en nombre de cycles sont au maximum 152 599 cycles entre le meilleur algorithme de routage et celui donnant la latence la plus grande. Ceci correspond à 33% de cycles en plus sur la latence la plus faible. Concernant le bloc TR1, les algorithmes donnent les meilleures et les pires latences pour les différentes charges étudiées. La meilleure latence est obtenue avec l'algorithme de routage NFNM et la plus grande latence est obtenue avec l'algorithme de routage NFM pour une charge de 25%. Leurs performances sont opposées pour une charge de 50%, l'algorithme NFM donnant les meilleures performances alors que les latences sont les plus importantes pour l'algorithme NFNM. Ainsi, pour une destination donnée, les algorithmes sont plus ou moins performants selon la charge. Enfin, les 3 algorithmes donnent les meilleures et les pires latences pour les différentes charges en ce qui concerne TR2. La différence de latence maximale obtenue est de 144 096 cycles entre les algorithmes NFNM et NFNM pour une charge de 25% (ce qui correspond à 113% de cycles en plus).

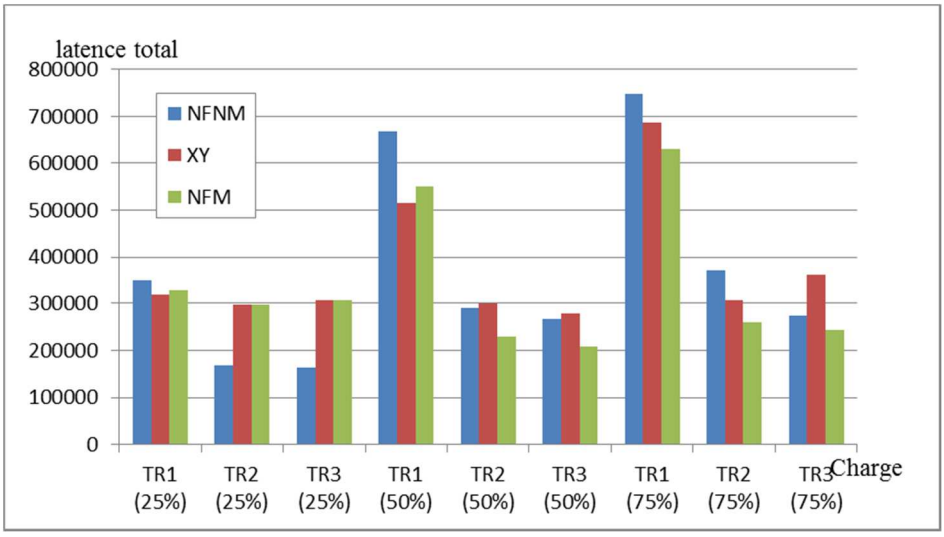


Figure 4-26 : Latence totale pour chaque destinataire pour des charges de 25%, 50% et 75% en fonction de 3 algorithmes de routage (1 déterministe et 2 semi adaptatifs)

Ainsi il n'y a pas de meilleur algorithme de routage selon une charge de réseau, une position de TG ou une position de TR. Il est par conséquent nécessaire de faire une analyse des performances de chaque scénario de l'application pour évaluer l'algorithme de routage pouvant présenter de meilleurs résultats dans l'ensemble des scénarii testés.

4.4 Etude des performances du NoC sur multi-FPGA

Les évaluations du NoC Multi-FPGA ont pour objectif:

- D'étudier la faisabilité d'un déploiement automatique de NoC sur plusieurs FPGAs en cas de ressources insuffisantes sur un FPGA,
- De trouver des mécanismes pour évaluer les ressources du NoC indépendamment des cibles multicomposants (notamment les blocs pour la communication externe),
- De trouver des mécanismes pour évaluer les performances temporelles des communications dans le NoC indépendamment des cibles multicomposants (notamment le temps de communication entre deux FPGAs),

Cette étude aborde le déploiement du NoC sur une architecture multi-FPGA. Cela indique que des informations de partitionnement sont données sur le flot de conception pour découper le NoC puis le déployer sur plusieurs FPGAs. L'architecture utilisée pour cette partie expérimentale en vue de l'exploration est une architecture 4x3 avec plusieurs TGs et un TR, comme représenté sur la Figure 4-27.Le format des données est de 50flitspar paquet et de 16 bits par flit.

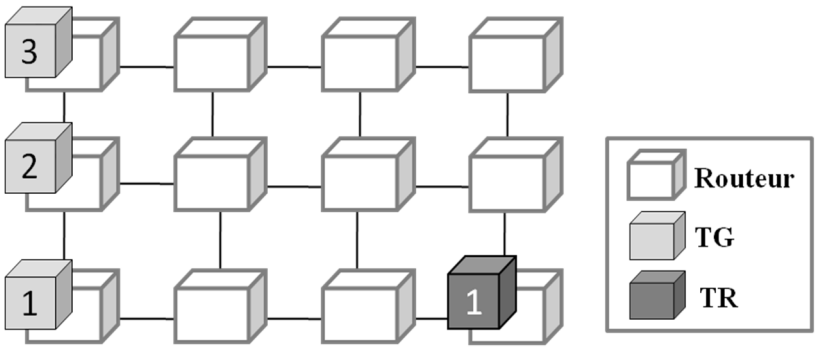


Figure 4-27 : Scénario utilisé sur un NoC 4x3 pour l'analyse du déploiement sur multi-FPGA.

Pour les cibles multi-FPGA, le NoC Hermes est partitionné en deux versions, chaque version correspondant à une version présentée à lasection3.2.3. La version 1 remplace un lien par une connexion inter-FPGA(N_{LI}= N_R= 3en utilisant le bloc d'adaptation 1).La version 2remplace plusieurs liens par une connexion inter-FPGA (avec N_{LI}=1< N_R= 3en utilisant le bloc d'adaptation2).

Les architectures générées par le flot de conception décrit au chapitre 3 sont représentées en Figure 4-28 et Figure 4-29. Quand les spécifications sont correctement remplies, la génération dure de quelques minutes.

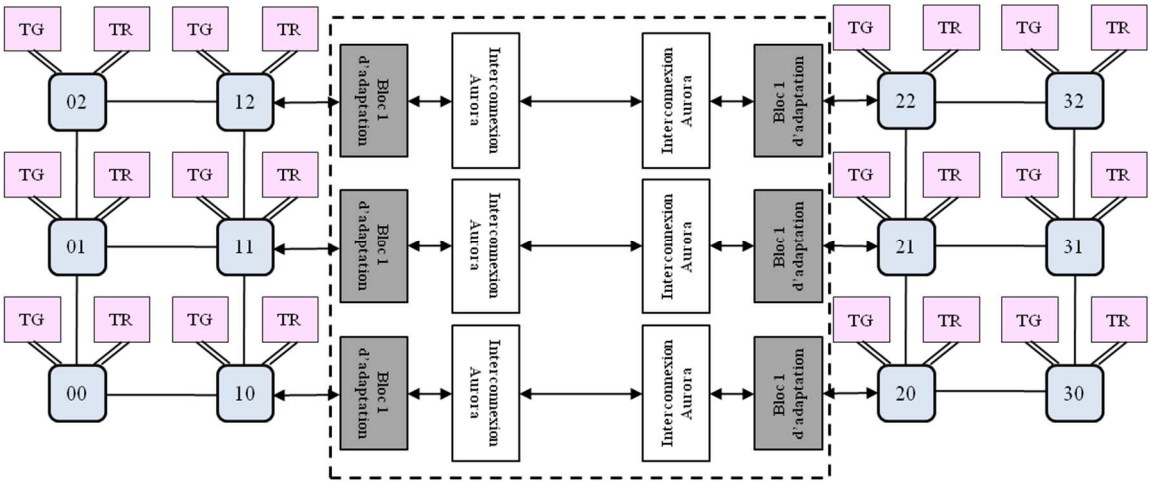


Figure 4-28 : Plateforme d'émulation générée par le flot de conception proposée pour l'implémentation sur multi-FPGA (scénario 1: NLI= NR). L'adaptateur 1 est utilisé pour la communication inter-FPGA.

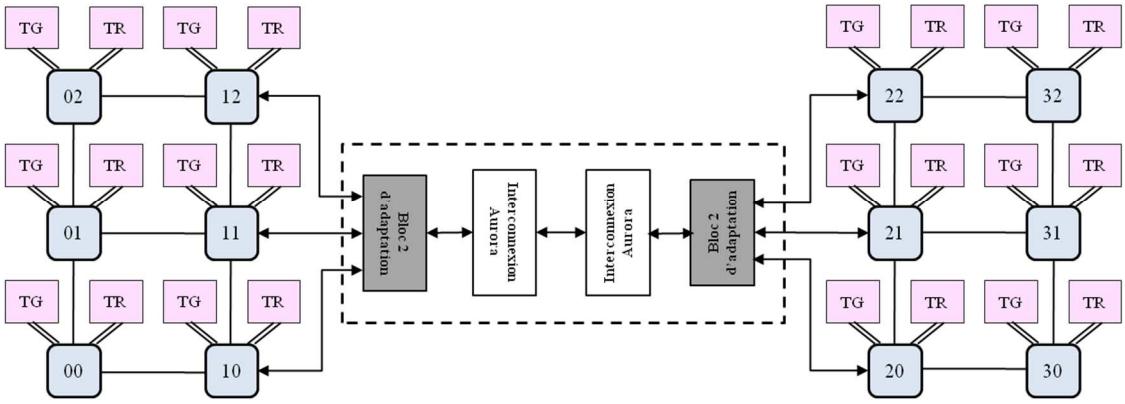


Figure 4-29 : Plateforme d'émulation générée par le flot de conception proposée pour l'implémentation sur multi-FPGA (cas 2: NLI<NR). L'adaptateur 2 est utilisé pour la communication inter-FPGA.

A partir de ces deux architectures, plusieurs explorations sont menées pour l'analyse des ressources et des performances temporelles.

Les Tableau 4-2 et Tableau 4-3 représentent respectivement le nombre de registres et de LUTs utilisés sur un FPGA. Pour le mono-FPGA, le nombre des ressources correspond aux ressources totales du NoC. Pour la version 1 et version 2, les ressources correspondent aux ressources d'un NoC 4 * 3 (la moitié du NoC), des blocs d'adaptation et des blocs RocketIO sur un FPGA.

Tableau 4-2 : Nombre de registres utilisés par FPGA

	Nombre de registres utilisés	% de registres utilisé	Gain
Mono-FPGA	5 260	16	
Version 1	3 441	10	1.53
Version 2	2 898	8	1.81

Tableau 4-3 : Nombre de LUTs utilisées par FPGA

	Nombre de LUTs utilisés	% de LUTs utilisés	Gain
Mono-FPGA	8 831	27	
Version 1	5 356	16	1.65
Version 2	5 094	15	1.73

Le nombre de ressources par FPGA est plus faible pour les implémentations sur multi-FPGA. On constate pour le NoC 4x3 réalisé sur 2 FPGAs que le nombre de ressources requis par FPGA. Le nombre total de ressources comprend le NoC mais aussi le ou les blocs d'adaptation et le bloc de transmission RocketIO.

Le nombre de ressources requis pour le bloc d'adaptation et les blocs RocketIO est plus faible que les ressources du NoC déportées sur l'autre FPGA. La version2 n'utilise qu'un seul bloc d'adaptation paramétrable et un bloc RocketIO alors que la version 1 en utilise 3. Le nombre de registres et de LUTs est divisé par1, 8 et 1,7 pour la version 2 et par 1.5 et 1.6 pour la version 1.Il est donc possible de déployer un NoC sur plusieurs FPGAs si un seul FPGA ne permet pas de contenir toutes les ressources. L'ajout de blocs d'adaptation ne limite pas son déploiement, les ressources associées étant faibles par rapport aux ressources du NoC, comme le montre le Tableau 4-4 et le Tableau 4-5 qui représentent respectivement le nombre total de registres et de LUT sur un FPGA pour la version mono-FPGA et sur deux FPGAs pour la version 1 et la version 2. Les registres supplémentaires sur le FPGA sont de 811 pour la version 1 et 268 pour la version 2 (soit 2,5% et 0,8% des ressources du FPGA). Les LUTS supplémentaires sont de 941 et 679 pour les versions 1 et 2 (soit 2,88 et 2,08% de LUTs en plus).

Tableau 4-4 : Nombre total de registres utilisés pour le déploiement d'un NoC sur plusieurs FPGAs.

	Nombre de registres utilisés	Nombre de registres supplémentaires	% de registres supplémentaires
Mono-FPGA	5 260		
Version 1	6 882	1622	30
Version 2	5 796	536	10.2

Tableau 4-5 : Nombre total de LUT utilisées pour le déploiement d'un NoC sur plusieurs FPGAs.

	Nombre de LUT utilisés	Nombre de LUTs supplémentaires	% de LUTs supplémentaires
Mono-FPGA	8 831		
Version 1	10 712	1881	21.9
Version 2	10 098	1267	14.37

La plateforme d'émulation multi-FPGA de la version 1 nécessite 30% de registres en plus et21% de LUTs en plus par rapport à la plateforme d'émulation monoFPGA. Ceci représente une augmentation de 10% pour les registres et d'un facteur 14 pour les LUTs. Ces facteurs sont sensiblement les mêmes pour une architecture contenant plus de 2 FPGAs. Considérant la taille maximale du NoC qu'un FPGA peut porter et l'augmentation des ressources nécessaires au déploiement, le concepteur peut estimer le nombre de FPGAs nécessaire pour intégrer la plateforme d'émulation complète. Cette analyse a été faite via l'utilisation d'une communication externe Aurora. Il serait intéressant de déployer le NoC sur plusieurs FPGAs en utilisant d'autres protocoles de communication externe.

4.4.1 Evaluation temporelle

La Figure 4-30présente la latence moyenne d'envoi des paquets sur les 3 architectures (1 mono-FPGA et 2 multi-FPGAs) avec une variation de la charge du réseau. L'évaluation temporelle est basée sur l'envoi de 50 paquets.

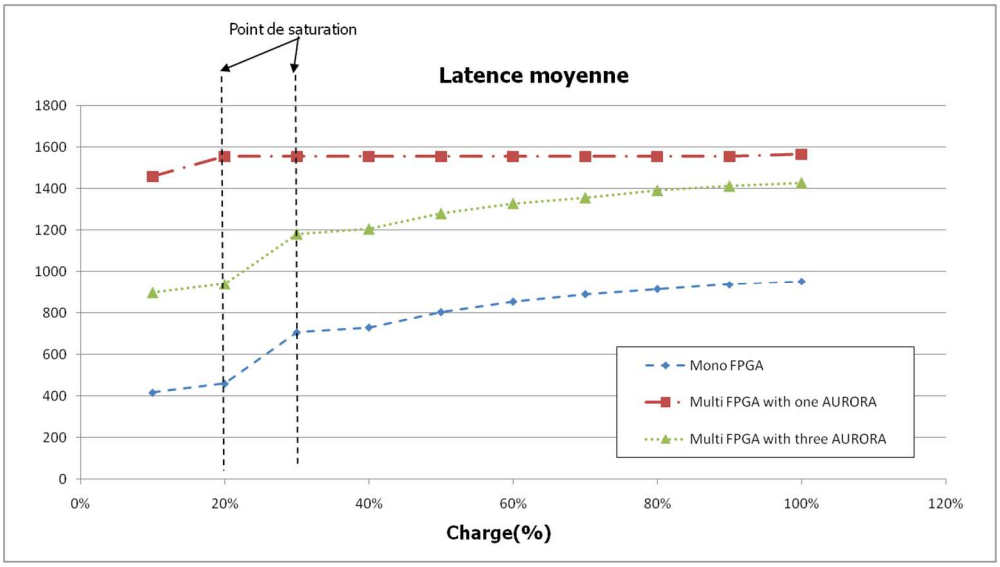


Figure 4-30 : Latence moyenne pour les trois versions d'implémentations sur mono et multi-FPGA.

On constate que le point de saturation de la solution mono-FPGA et de la version 1 est le même (sur une charge de 25%) alors qu'il est inférieur pour la version 2 (approximativement de 10%). Le multiplexage des données inséré sur les blocs d'adaptation pour la version 2 rajoute un goulot d'étranglement, diminuant ainsi le point de saturation et la zone de fonctionnement du NoC. La communication déployée de la version 1 consiste juste à remplacer un lien par une communication externe, ce qui ne modifie pas le comportement des communications des paquets dans le NoC (les deux courbes mono-FPGA et version 1 ont une évolution identique). L'utilisation multi-FPGA peut définir le point de saturation du NoC indépendamment des FPGAs via l'analyse d'une architecture de type version 1.

En plus du point de saturation, les latences des communications peuvent être évaluées. La latence du paquet pour le NoC sur mono-FPGA(T_{mono}) peut être déduite des latences multi-FPGA de la version 1 (T_{multi}) en considérant la communication inter-FPGA ($T_{inter\text{-}FPGA}$).Le temps de transfert d'un paquet est obtenu via l'équation suivante :

$$\boldsymbol{T_{mono} = T_{multi} - (T_{inter-FPGA} * ext\_cpt)} \qquad \textbf{Équation 4-1}$$

Avec $T_{inter\text{-}FPGA}$ représentant le temps de communication entre deux FPGA et ext_cpt représentant le nombre de liens d'inter-FPGA traversés comme expliqué précédent. $T_{inter\text{-}FPGA}$ est défini selon :

$$\boldsymbol{T_{inter-FPGA} = Size\_packet * (T_{FIFO\_in} + T_{FIFO\_out} + T_{aurora})} \qquad \textbf{Équation 4-2}$$

Le temps nécessaire pour la communication entre deux FPGAs correspond à la somme (Size_packet) des chargements du paquet dans la FIFO du bloc d'adaptation du FPGA1 (T_{FIFO_in}), au temps de

transfert via le protocole Aurora (T_{aurora}) puis au chargement du paquet dans la FIFO du FPGA de sortie (T_{FIFO_out}).

Le temps pour le transfert de paquets à travers le protocole Aurora est de 24cyclespour l'initialisation et1 cycle /flit. Pour laFIFO, T_{FIFO_in}=T_{FIFO_out}= 1 cycle/flit. Dans notre expérimentation,T_{inter_FPGA}= 474cycles.

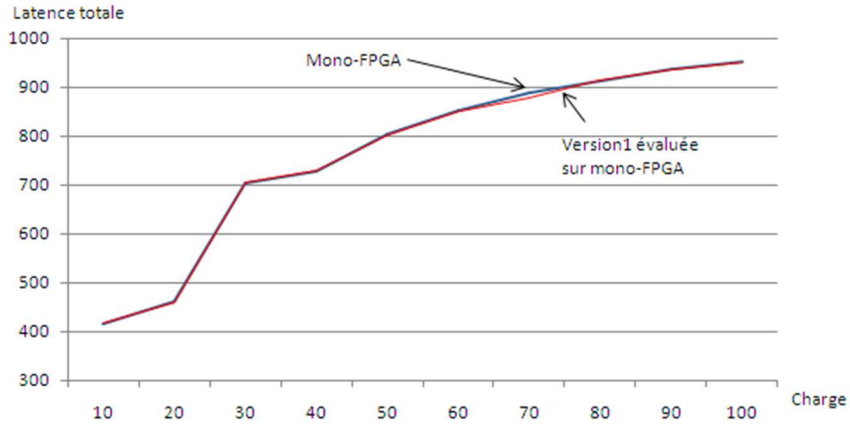


Figure 4-31 : Latence estimée du NoC à partir de la latence multi-FPGA version 1et latence obtenue pour la version mono-FPGA.

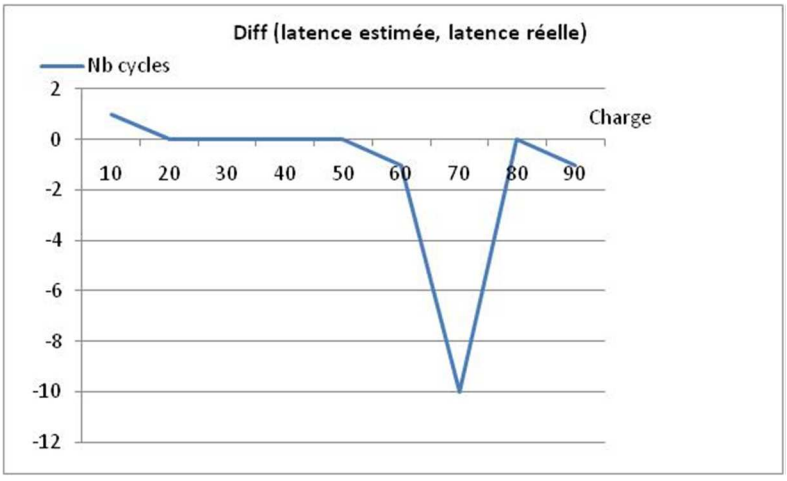


Figure 4-32 : Différence entre la latence du NoC extraite de latence multi-FPGA version 1par rapport à l'implémentation du mono-FPGA.

En retirant la latence de la communication inter-FPGA calculée précédemment, la latence d'implémentation sur mono-FPGA et la latence du NoC évaluée à partir de la version 1 du multi-FPGA sont présentées en Figure 4-31. On constate que les deux courbes sont presque confondues sauf aux alentours de charge de réseau de 70%. La différence du nombre de cycles pour ces deux versions est donnée en Figure 4-32. La latence estimée du NoC est très proche de la latence réelle,

la différence maximale concernant le nombre de cycles étant d'une dizaine de cycles pour une charge de 70%.

4.5 Analyse des résultats pour le dimensionnement du NoC

Toutes les expérimentations nous permettent d'apporter quelques conclusions pertinentes sur le dimensionnement du NoC sur plusieurs points de vue :

D'un point de vue ressources sur le FPGA :

- Le dimensionnement du NoC a une influence directe sur les ressources du FPGA : l'élévation du nombre de nœuds et de la taille des flits augmente de manière quasi-linéaire le nombre de ressources, les LUTs constituant les ressources limitantes du FPGA.
- Le choix d'un algorithme de routage de type Turn-Model n'a aucune influence sur les ressources du FPGA.
- Il est envisageable de déployer le NoC sur plusieurs FPGAs si les ressources des blocs de calcul sont importantes ou si la taille du NoC est élevée.

D'un point de vue performances de la communication :

- Il est possible d'accélérer les transferts de paquets dans le NoC grâce à l'augmentation de la taille des flits du NoC ou à l'augmentation du nombre de flits par paquet.
- Le nombre de paquets n'a aucune influence sur les performances temporelles (la latence calculée en général est la latence d'un paquet).
- Le choix de l'algorithme de routage peut avoir une influence significative sur les performances temporelles. Il n'existe pas un algorithme de routage qui donne de meilleures performances. Une analyse des communications permet de faire un choix sur l'algorithme de routage à utiliser.
- Les performances temporelles concernant les transferts de paquets dépendent du nombre de TGs et TRs ainsi que de leur position, de la charge du réseau, de l'algorithme de routage choisi. Des explorations sur l'ensemble des scénarii est nécessaire afin d'estimer les solutions les plus adaptées à l'algorithme.
- Il est nécessaire de définir le point de saturation pour chaque scénario. Ce point de saturation permet d'extraire la zone de fonctionnement et la zone de saturation et de définir une charge de réseau située dans la zone de fonctionnement.

D’un point de vue plateforme d’émulation :

- L’utilisation de blocs d’émulation permet d’émuler sur le circuit les scénarii des applications de traitement d’images. Les ressources nécessaires pour l’insertion de ces blocs représentent 10% de ressources en plus pour des petites taille de NoC.
- Dans ce cas, il est facile d’estimer les performances temporelles et en ressources du NoC indépendamment. L’estimation en ressources se fait via une communication multi-routeur. L’estimation temporelle (incluant le point de saturation) se fait via une communication routeur-routeur.
- La génération de la plateforme d’émulation déployée sur multi-FPGA est rapide grâce au flot de conception proposé.

L’algorithme d’authentification d’œuvres d’art en imagerie spectrale implantée sur les architectures embarquées utilise les paramètres d’entrée suivant :

- Nombre de régions = 2 000.
- Nombre de longueurs d’onde pour chaque région = 128, 480, 960, 992.
- Tailles de la fenêtre (pour la moyenne) =1x1, 2x2, 4x4, 8x8, 16x16, 32x32, 64x64 pixels.

A partir de toutes les observations réalisées sur le NoC et les paramètres de l’application, les caractéristiques de l’architecture de NoC proposée sont :

- La taille des flits est de 8 bits ou 16 bits (selon les performances visées),
- Le NoC est de taille 4x4,
- Le nombre de nœuds de calcul est de 13, voire 14,
- Deux nœuds d’acquisition permettent de récupérer respectivement les données originales et les données à comparer,
- Les calculs de moyenne, de projection XYZ, Lab et RGB sont implantés en double sur des routeurs différents, pour exécuter en parallèle les calculs sur les données originales et les données à comparer,
- Un nœud de calcul exécute un calcul de distance, en couleur ou en multispectral,
- A partir des scénarii, une charge de réseau inférieure à 66% permet de se trouver en dessous du point de saturation.

L’architecture finale du NoC sur laquelle sont implantées les différentes fonctions est donnée en Figure 4-33. Nous considérons deux blocs d’acquisition, un pour les images originales qui sont mémorisées en format brut dans une mémoire (**acqui_2**), un pour l’acquisition des images à

comparer provenant directement d'une caméra spectrale (**acqui_1**). Les flux de données à transférer peuvent être des flux pour une région ou plusieurs régions (noté **R**) et sont fonction du nombre de longueurs d'onde (noté **W**) ou/et de la taille de la fenêtre (notée **S**). Les évaluations de la communication se faisant sur un FPGA, le format des paquets utilisés est celui représenté en Figure 3-8. Les données d'une région sont envoyées via un paquet auquel est associé un entête de taille de 5 flits. Ainsi, pour des spécifications d'application de W=128 et S=64, la taille du paquet entre acqui_1 et moy_1 est de taille 8187 flits (=5+128x64= et entre moy_1 et XYZ_1 la taille des paquets est de 132 (= 5+128).

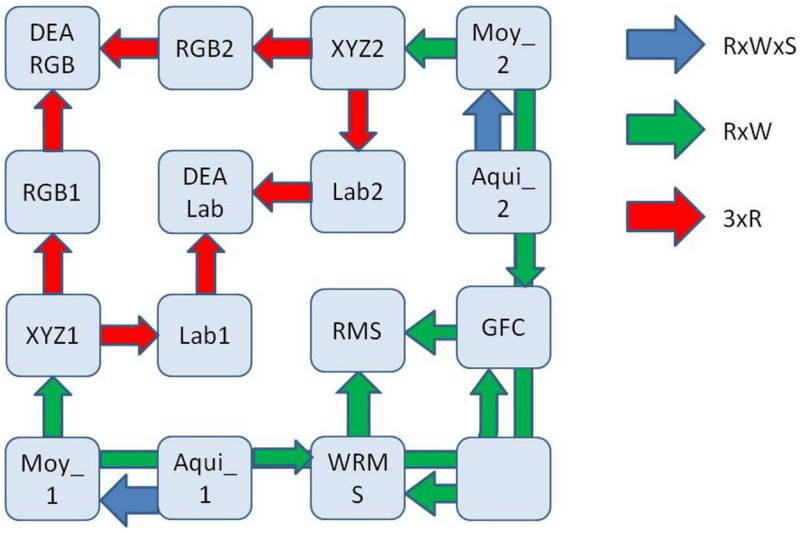


Figure 4-33 : Architecture du NoC Hermes pour l'application d'imagerie spectrale.

A partir du NoC proposé, plusieurs expérimentations portant sur les latences des paquets sur les différents nœuds du NoC sont réalisées au moyen de NoCGen. Les résultats des latences pour chacune des fonctions lors de l'émulation sur le FPGA sont donnés dans le Tableau 4-6,

Tableau 4-6 : Latences des différentes scénarii de l'algorithme d'authentification par émulation sur FPGA Virtex 5 pour R=1, W=128, S=64.

	Latence
acquisition1 => moyenne1	16407
moyenne1 => XYZ1	279
XYZ1 => RGB1	29
XYZ1 => Lab1	29
RGB1 & RGB2 => DEA RGB	78
Lab1 & Lab2 => DEA Lab	78
RGB1 & RGB2 => WRMS	841
moyenne1 & moyenne2 => GFC	841

Les Figure 4-34 et Figure 4-35 représentent respectivement la latence de la communication entre le bloc acquisition_1 et le nœud de la fonction moyenne pour respectivement différentes tailles de fenêtre et différentes longueurs d'onde.

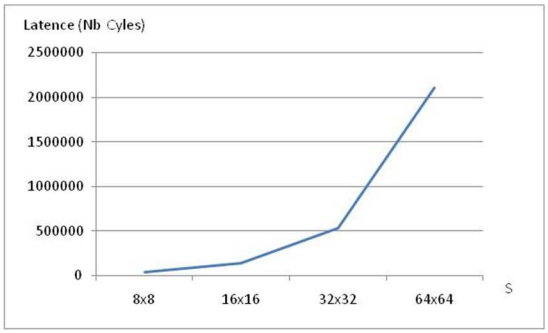


Figure 4-34 : Latence de la communication acquisition=> moyenne pour différentes tailles de fenêtre (w=128, R=1).

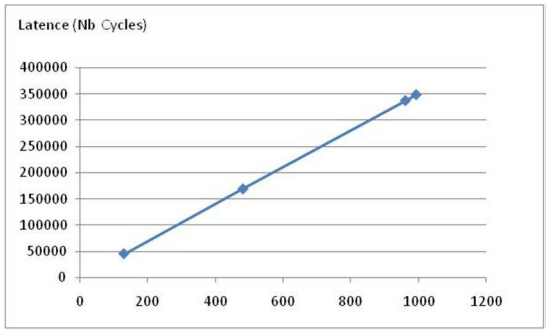


Figure 4-35 : Latence de la communication acquisition=> moyenne pour différentes valeurs de longueurs d'onde (S=64, R=1).

La latence entre le nœud moy_1 et XYZ1 est donnée en Figure 4-36. On observe de la même manière que le temps de communication entre ces deux nœuds est proportionnelle au nombre de longueurs d'onde spécifié.

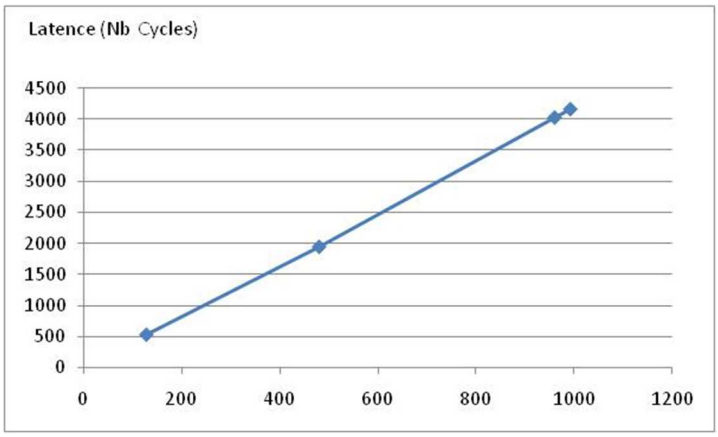


Figure 4-36 : Latence de la communication moy_1=> XYZ_1 pour différentes valeurs de longueurs d'onde (S=64, R=1) pour une charge de réseau de 50%.

D'un point de vue général sur l'analyse des communications sur le NoC Hermes pour l'algorithme d'authentification d'œuvres d'art, on constate que les transferts de données sont linéaires et dépendent des paramètres de l'application tels que le nombre de longueurs d'onde et la taille de la fenêtre. Ainsi l'utilisateur peut estimer les temps de communication pour les différents paramètres de l'application grâce à quelques implantations judicieusement choisies (implantations qui sont

réalisées rapidement grâce au flot de conception proposé).Les implantations judicieusement choisies sont :

- Une taille de NoC possédant un nombre de nœuds égal au nombre de fonctions dans l'application (privilégiant massivement le pipeline des applications). La taille des flits utilisée est de préférence 8 ou 16 bits et la charge du réseau de 50%. Les données sont envoyées via un paquet.
- Une implantation avec des charges de réseaux adaptées pour chaque TRs en dessus ou dessous de 50%. L'utilisateur exploitera les résultats de la 1ère implantation et le scénario utilisé pour définir la zone de saturation par rapport à une charge de 50%.
- Ensuite, l'utilisateur peut faire varier la taille des flits, la taille des paquets (en découpant ou regroupant certains paquets), modifier la position des fonctions sur le NoC. Les adaptations dépendront fortement de l'application ciblée.

4.6 Conclusion

Dans ce chapitre, plusieurs expérimentations portant sur le paramétrage du NoC Hermes ainsi qu'un panel de scénarii ont permis de dimensionner rapidement l'ensemble des solutions du NoC envisageables sur un FPGA Virtex 5 et de définir un placement des tâches de l'algorithme sur ce NoC. Les plateformes d'émulation ont été générées rapidement, permettant de couvrir l'ensemble de l'espace de conception du NoC. Ensuite, les scénarii correspondant aux transferts des données dans le NoC pour l'algorithme d'authentification d'œuvres d'art ont été testés et évalués. Ces évaluations peuvent être prises en compte dans l'analyse des performances de l'architecture complète qui se fait dans le chapitre suivant.

Il est possible pour n'importe quel utilisateur d'avoir à sa disposition une plateforme d'émulation pour un algorithme aussi complexe soit-il de traitement de signal et d'images et de tester une architecture de communication sur différents FPGAs avec un grand ensemble de scenarii à évaluer. L'émulation s'avère nécessaire et indispensable car la simulation effectuée sur PC et hors circuit de prototypage ne permet pas d'obtenir des évaluations précises des temps de communication.

Chapitre 5. Expérimentation : Implantation d'algorithmes d'imagerie multi-spectrale sur une architecture multiprocesseur

L'objectif de ce chapitre est l'implantation de l'algorithme d'imagerie multi-spectrale sur une architecture multiprocesseurde type MPSoC basée sur une architecture NoC. Des simulations sont réalisées autour de cette architecture multiprocesseur basée sur le NoC Hermes et des processeurs RISC choisie. Le dimensionnement de l'architecture pour l'algorithme d'authentification est exploré ainsi qu'une étude de faisabilité d'implantation sur une plateforme d'émulation sur FPGA. Ce chapitre aborde également l'analyse de la précision obtenue sur une architecture embarquée et une exploration des temps de calcul par rapport aux paramètres de l'application. Ce chapitre, principalement expérimental, aborde une analyse précise des architectures MPSoC utilisant un NoC et présente des perspectives sur la conception d'une plateforme d'émulation MPSoC et NoC complète.

L'architecture MPSoC est de plus en plus utilisée dans les systèmes embarqués. Cependant la conception de MPSoC pose le problème de l'optimisation des performances [77][78].Leur conception ouvre un espace de conception très large, chaque cœur possédant son propre ensemble de paramètres. Les mesures de performances d'un MPSoC ne sont pas toujours faciles. En raison de ces facteurs, les outils de CAO (Conception Assistée par Ordinateur) et les plateformes de simulation sont inévitables pour réaliser la conception et l'exploration d'architectures. Plusieurs plateformes MPSoCs utilisant une communication de type bus ou NoC existent dans le monde académique ou industriel.

5.1 Différents plateformes de génération et simulation de MPSoC

Il existe de nombreux outils de simulation ou de génération de MPSoC. Un ensemble de paramètre permet de définir l'architecture externe de processeurs (nombre et type), d'IPs, mémoire et interconnexion. Une rapide description de plusieurs plateformes connues dans le domaine est faite. L'une d'entre elle offre un bon rapport entre simplicité d'utilisation et la possibilité de simulation et génération sur FPGA. Cette plateforme est choisie pour la suite des expérimentations.

5.1.1 SocLib

SoCLib est une plateforme ouverte pour le prototypage virtuel de systèmes MPSoC dans le domaine de l'embarqué. SoCLib permet le développement de plateformes de prototypage virtuel et facilite l'exploration d'architectures au moyen d'outils et de bibliothèques :

- Une bibliothèque Open Source est composée de modèles de simulation de haut niveau écrits en SystemC pour les composants matériels, qui permettent des simulations rapides. Les IPs disponibles sont des processeurs avec mémoires cache, interconnexions et des blocs mémoire.

- Les outils logiciels pour concevoir des applications embarquées : compilateur, linker pour chaque processeur, accélérateur de simulation, outil de génération automatique de modèles de simulation et débogueur.

SoCLib peut être utilisé pour créer un système matériel complet, ainsi que pour tester et évaluer des applications. Pourtant, l'architecture MPSoC conçue par SoCLib n'est pas synthétisable et ne peut pas être implantée sur FPGA.

5.1.2 STARSoC

STARSoC (Synthesis Tool for Adaptive and Reconfigurable System-on-Chip) [79]est une plateforme de codesign et exploration d'espace de conception pour des architectures matériel/logiciel. Cette plateforme est développée au LE2I. Pour cette plateforme, la description d'entrée du flot de conception (Figure 5-1) est composée d'un ensemble de processus logiciel/matériel décrits en C. Après avoir indiqué le nombre de processeurs reconfigurables (OpenRISC), le partitionnement logiciel et matériel est spécifié. Pour ce flot de conception, la partie matérielle est synthétisée en architecture RTL et la partie logicielle est distribuée sur les différents processeurs. La communication basée sur un bus« whisbone » est synthétisée pour l'interconnexion

entre les coprocesseurs matériels et les processeurs. De cette façon, à partir d'une spécification d'applications de haut niveau, STARSoC génère une plateforme MPSoC en utilisant le bus pour communication.

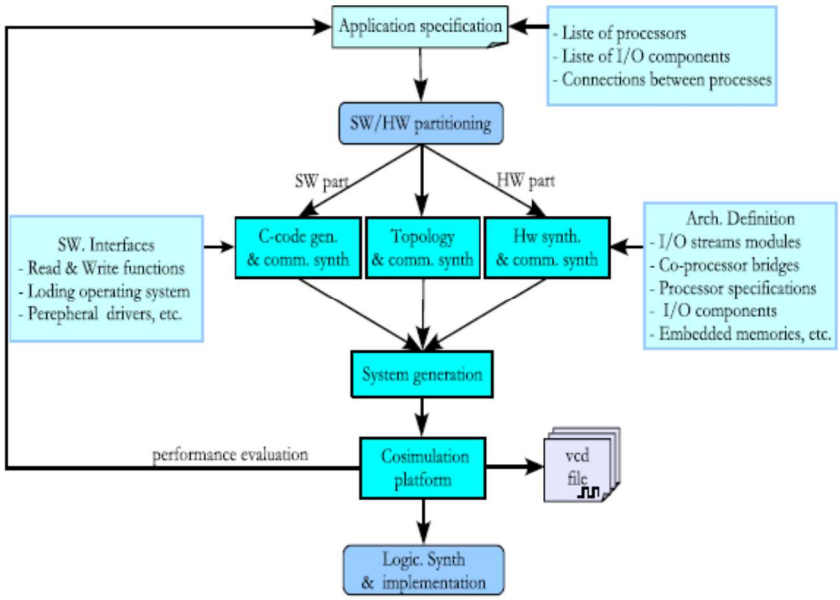


Figure 5-1 : Flot de conception de STARSoC[79]

Afin de faciliter et de réduire le temps d'exploration de plusieurs solutions, STARSoC propose un environnement de simulation de type TLM (« Transaction Level Modeling »). Une fois les composants définis, la description RTL est générée par la synthèse de haut niveau avec le système de communication qui peut être implantées sur FPGA.

5.1.3 SESAM

Le SESAM [81]est proposée par le CEA, LIST afin de faciliter la conception et l'exploration d'architecture multiprocesseurs(MIPS, PowerPC, Sparc) basée sur une communication de type NoC (Figure 5-2).

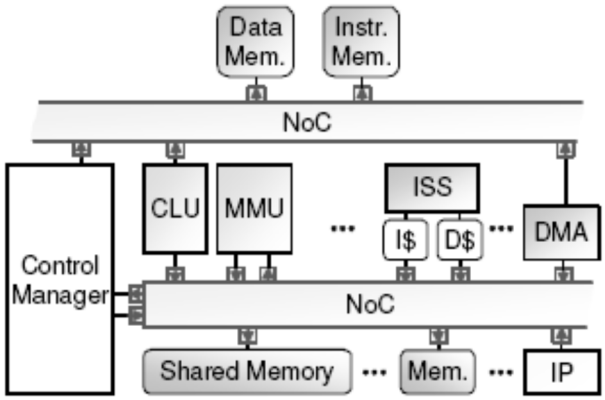


Figure 5-2 : L'infrastructure de SESAM

SESAM peut être utilisé pour analyser et optimiser le parallélisme de l'application. L'architecture est décrite en SystemC et permet l'exploration au niveau TLM avec des simulations rapides au niveau cycle. Les composants et les paramètres du système sont indiqués en temps réel à partir d'un fichier de paramètres sans recompilation de la plateforme.

L'architecture générée par l'outil SESAM n'est pas synthétisable et ne supporte pas l'implémentation sur FPGA.

5.1.4 La plateforme HeMPS

La plateforme Hermes MultiProcessor System, HeMPS[82] proposée par l'université PUCRS est une plateforme hétérogène de MPSoC basée sur une communication de type NoC. Les principaux composants matériels sont le NoC HERMES et les processeurs plasma « mostly-MIPS » [83].

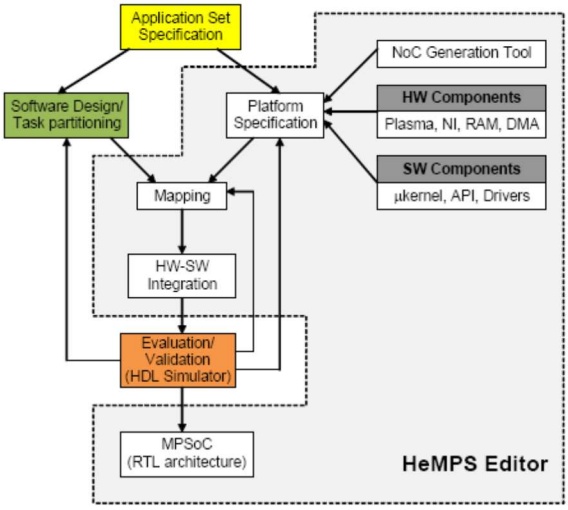


Figure 5-3 : Flot de conception de la plateforme HeMPS

Le flot de conception proposé par la plateforme HeMPS est présenté dans la Figure 5-3. Il commence par les spécifications des applications qui vont être exécutées sur l'architecture MPSoC. En utilisant une interface API (« Application Programming Interface ») prédéfinie de HeMPS, l'utilisateur spécifie les fonctions de l'application et définit l'architecture cible. Le développement logiciel consiste à décrire le graphe des tâches de chaque application et le partitionnement de ces tâches. Les spécifications de l'architecture consistent à paramétrer le NoC et à définir les mémoires locales. L'architecture est générée en VHDL synthétisable avec un modèle de simulationC/SystemC pour le processeur et la mémoire. Le simulateur de type ISS (« Instruction Set Simulator ») permet de réduire de 91% le temps de simulation par rapport à la simulation RTL. L'architecture MPSoC basée sur un NoC générée par HeMPS peut être implantée sur FPGA.

Cette plateforme semble posséder toutes les bonnes caractéristiques pour une utilisation pour la suite des expérimentations. Elle est basée autour du NoC Hermes qui constitue une partie de ce travail de thèse et elle offre plusieurs niveaux de simulation et la génération de VHDL synthétisable pour une implantation sur FPGA.

5.2 Expérimentation de l'architecture MPSoC pour l'application.

Dans cette partie d'expérimentation, la plateforme HeMPS est utilisée pour générer et simuler l'architecture MPSoC pour l'application d'authentification de l'œuvre d'art. Notre choix s'est porté vers cet outil par rapport aux autres outils présentés précédemment. HeMPS génère une architecture utilisant un NoC (StarSoC utilisant un bus traditionnel) au niveau RTL ou ISS et cette architecture RTL est implantable sur FPGA. De plus, HeMPS possède une interface graphique facile à utiliser et l'environnement de HeMPS est disponible en « opensource ».

Nous présentons l'architecture matérielle MPSoC, l'interface graphique pour le placement de tâches et le débogage ainsi que la plateforme d'émulation de l'architecture HeMPS implantée sur FPGA.

5.2.1 L'architecture HeMPS

Les composants de base de l'architecture HeMPS sont le NoC Hermes et le processeur plasma (Figure 5-4).

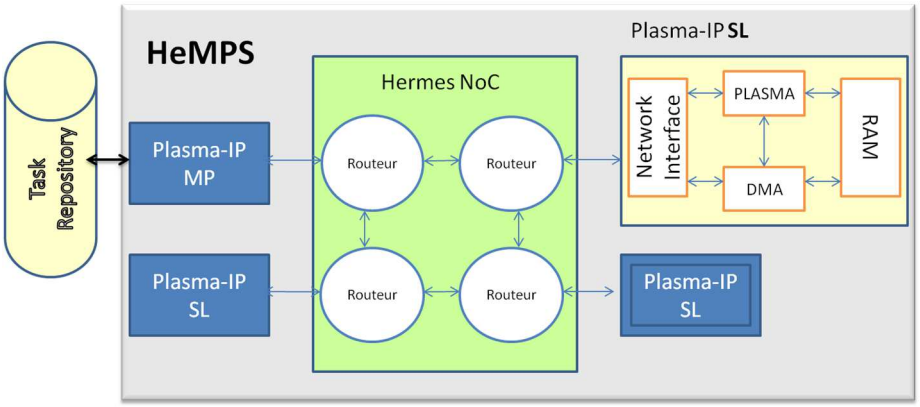


Figure 5-4 : Architecture MPSoC-NoC HeMPS

Le processeur plasma est encapsulé dans un bloc IP appelé plasma-IP pour permettre la connexion au NoC. Ce bloc IP contient également une mémoire privée, une interface réseau, et un module DMA (Direct Memory Access). Pour les applications exécutées sur l'architecture, les applications doivent être modélisées sous forme de graphes de tâches.

Les tâches placées sur un même processeur doivent être regroupées dans la fonction main() du processeur. Les tâches non affectées à un processeur sont stockées dans une mémoire externe, nommé « task repository ».

L'architecture contient un processeur maître (Plasma-IP MP), qui est responsable de la gestion des ressources du système. Il est le seul à avoir accès à la mémoire « task repository ». Au lancement de l'exécution, le processeur maître distribue de manière dynamique les tâches non affectées aux processeurs esclaves qui l'entourent.

Pour chaque processeur esclave, l'interface réseau et le DMA gèrent les envois et les réceptions des paquets, tandis que le processeur Plasma effectue les traitements de tâches et la mise au format des paquets. Chaque processeur possède une mémoire locale distribuée (RAM). Cette mémoire double-port permet de communiquer simultanément avec le processeur et le DMA pour une communication directe via un bloc NI pour ce dernier. L'organisation de la mémoire locale du processeur est divisée en plusieurs pages (Figure 5-5). Le « microkernel », placé en page 0, supporte le multitâche et les communications entre les tâches sur la page 0. Les autres pages de la mémoire sont réservées pour les « microkernels » dédiés à l'exécution des tâches. La taille de la mémoire et le nombre de pages par processeur sont paramétrables.

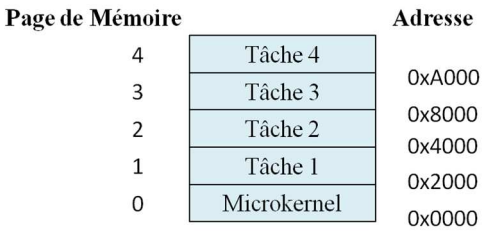


Figure 5-5 : Configuration de la mémoire avec 5 pages.

5.2.2 L'outil de génération HeMPS

L'outil de génération HeMPS couvre les différentes étapes de la conception présentées dans la section 5.1.4, et génère automatiquement l'architecture. La Figure 5-6 présente l'interface graphique principal de l'outil HeMPS.

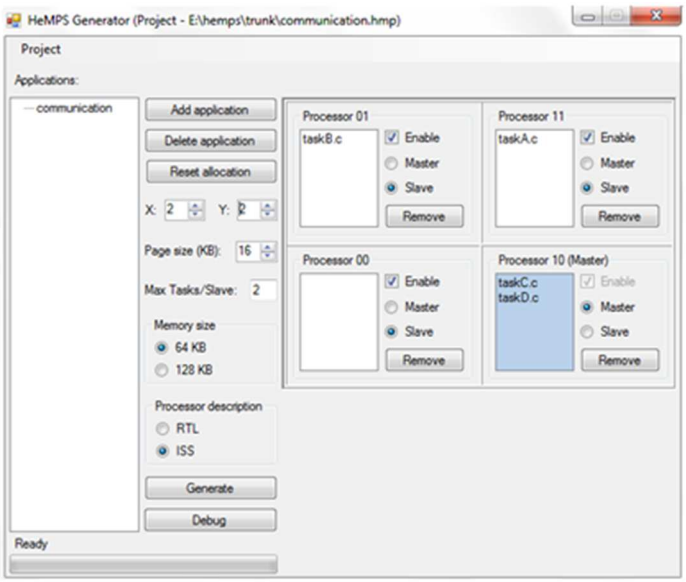


Figure 5-6 : Interface graphique de l'outil HeMPS

L'utilisateur définit le nombre de processeurs connectés au NoC Hermes (la taille du NoC étant spécifiée par les paramètres X et Y) ainsi que le placement des tâches sur les processeurs. Le nombre de tâches pour chaque processeur esclave est limité par la taille de la mémoire et le nombre de pages de mémoire. Les tâches non placées doivent être affectées au processeur maître qui réalisera un placement dynamique sur les processeurs esclaves voisins lors de l'exécution de l'application. Un exemple est donné dans la Figure 5-6 avec deux tâches (task A et task B) placées respectivement sur les processeurs esclaves 01 et 11. Le processeur 10 est défini comme le processeur maître et les tâches restantes (task C et task D) sont affectées à ce processeur de contrôle.

Afin d'évaluer les performances, les descriptions RTL et ISS du processeur sont disponibles. L'intégration de l'application et de l'architecture se fait à la fin via l'instruction « generate ».

Chaque processeur exécute des fonctions écrites en C sur des formats de données de 32 bits en virgule fixe et regroupées dans une tâche nommée « main ». La tâche possède trois parties principales : 1) réception des données venant d'autres tâches, 2) exécution de la (ou des) fonction(s) de la tâche, 3) envoi des résultats vers les tâches descendantes. Le nombre de tâches varie en fonction du nombre de processeurs et du placement des fonctions sur les différents processeurs. Il est donc nécessaire d'écrire les tâches en fonction de l'architecture.

Pour les expérimentations menées dans la suite de ce manuscrit, l'outil de simulation Modelsim a été utilisé. Le débogage consiste à envoyer vers le processeur maître des informations de timing pour chaque processeur via une applet system « print() ». Ces informations sont associées à la tâche exécutée et à l'identification de processeur. L'interface de débogage est présentée en Figure 5-7.

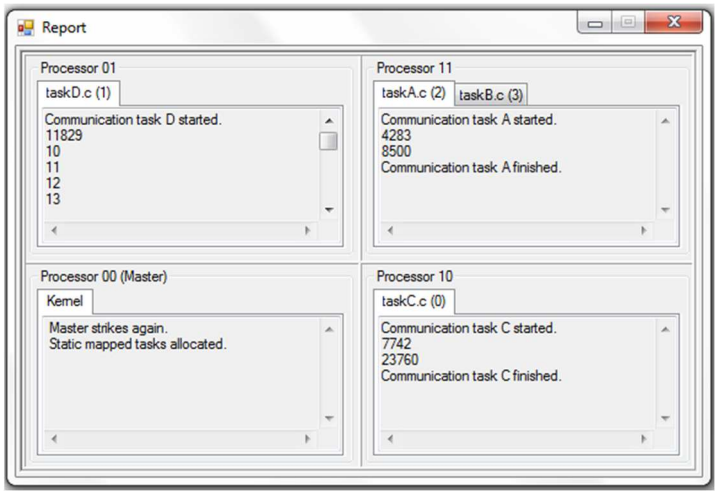


Figure 5-7 : Interface de débogage HeMPS.

5.2.3 Etude de performances de l'architecture HeMPS pour l'application d'authentification d'œuvre d'art.

Les performances sont évaluées pour la même structure de NoC (NoC de taille 4x4) donnée dans le chapitre précédent. Un redimensionnement de l'architecture est effectuée afin d'analyser l'impact des performances par rapport aux paramètres de l'architecture.

5.2.3.1 Contraintes liées à la plateforme HeMPS

Le portage de l'application sur cette plateforme a nécessité quelques modifications importantes dûes aux contraintes de HeMPS.

- Le format des données traitées sur le processeur est un format de données 32 bits en virgule fixe. La précision des données expérimentales et des calculs nous amène à utiliser un format de données sur 32 bits dont 16 bits pour la partie entière et 16 bits pour la partie décimale. La communication vers le processeur étant de 16 bits, le paquet contient donc 2 flits pour chaque donnée.

- L'environnement HeMPS ne permet pas de faire du multitâche sur les processeurs esclaves étant donnés les paramètres choisis. Les processeurs exécutent donc les fonctions en séquentiel. Ainsi le placement de plusieurs tâches sur un processeur modifie le parallélisme de flux potentiel de l'application. L'ordonnancement des tâches est donc statique et décrit dans le main.

- La taille des flits du NoC Hermes 4x4 est de 16 bits.

- Le paramétrage des processeurs est identique quant à la taille de la mémoire et le nombre de tâches maximum par processeur.

- Les fonctions d'acquisition ne sont pas considérées dans cette partie. On considère que les données sont placées dans la mémoire des nœuds de calcul de moyenne. Les paramètres utilisés pour l'algorithme sont une région de taille 8x8 sur 16 longueurs d'ondes.

Il est également important de préciser que l'ensemble des fonctions de calcul n'étaient pas disponibles dans les bibliothèques de l'outil. Nous avons donc écrit l'ensemble des opérations de calcul nécessaire pour l'application d'imagerie spectrale.

5.2.3.2 Expérimentation sur le NoC Hermes 4x4

La structure du NoC maillé 4x4 et le placement des tâches utilisés pour cette expérimentation sont ceux présentés en Figure 4-33. Les performances sont donc étudiées sur un NoC 4x4 possédant 16 processeurs (15 esclaves et 1 maître). De part la configuration choisie, chaque processeur esclave ne peut exécuter qu'une seule tâche décrite dans son main(). Plusieurs tâches sont possibles mais doivent être ordonnancées de façon statique dans le main().

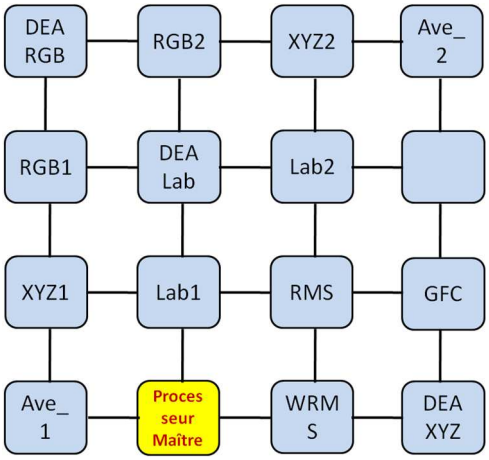


Figure 5-8 : Placements des fonctions sur une architecture MPSoC 4x4.

A partir du placement de chacune des fonctions sur le NoC 4x4 illustré dans la Figure 5-8, nous avons écrit une tâche pour chaque fonction et implanté les fonctions sur les processeurs correspondants (Figure 5-9). Dans cette configuration, le processeur maître (le processeur10) ne distribue aucune tâche car toutes les tâches ont été placées.

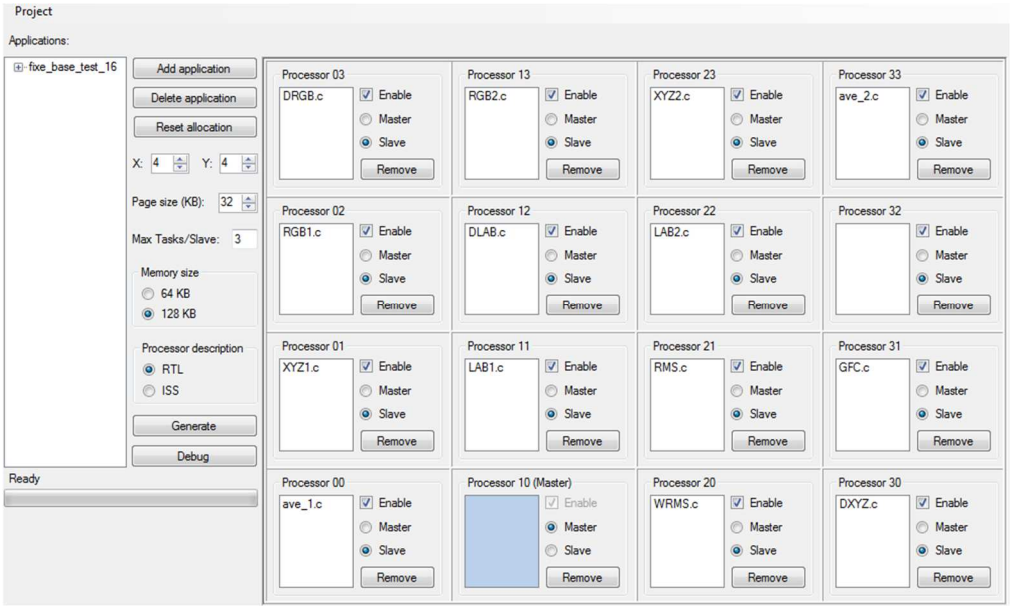


Figure 5-9 : Placement des tâches de l'application sur l'outil HeMPS

Tableau 5-1 : Temps d'exécution pour chaque fonction de l'application (16 longueurs d'ondes, 1 région, taille de fenêtre de 64 pixels)

Fonction	Nombre de cycles de l'horloge
Ave_1	30287
XYZ1	24523
RGB1	5 838
LAB1	94005
Ave_2	31 096
XYZ2	24922
RGB2	7 401
LAB2	93 698
Dist_XYZ	29 595
Dist_RGB	29 518
Dist_LAB	34 722
Dist_WRMS	1 117 462
Dist_RMS	89 743
Dist_GFC	118022
Temps total	1 325 966

Les latences pour chaque fonction exécutée sur le processeur ainsi que le temps de calcul total (hors communication entre les nœuds) sont données dans le Tableau 5-1. Nous constatons que le temps de calcul pour la distance WRMS est beaucoup plus long par rapport aux autres fonctions, environs 84% du temps total d'exécution. La latence totale de l'ensemble des traitements hors communication est de 1 325 966 cycles d'horloge, soit un temps de calcul de 13,2 ms pour une fréquence de 100 MHz. Ce temps correspond à l'exécution des fonctions en séquentiel sur un seul et même processeur. Le temps de calcul sur GPU de fréquence 576 MHz est de 11,5 ms. Ainsi une architecture embarquée de type MPSoC permet d'obtenir des temps de calcul proche sur des calculs se faisant sur 1 région à une fréquence de fonctionnement plus faible (et donc moins consommatrice en puissance). L'élévation de la fréquence d'horloge du FPGA à 576 MHz permettrait d'obtenir des temps de calcul de l'application de 2,29 ms. Il est important de repréciser que le portage de l'application sur un processeur embarqué a nécessité de diminuer la précision des données et résultats, le GPU travaillant sur des données en virgule flottante 64 bits et le processeur embarqué opérant sur des données en virgule fixe 32 bits.

Le nombre de longueurs d'ondes a été modifié de 16 à 64 pour observer la fonction de calcul de distance WRMS, fonction la plus couteuse en temps de calcul (Tableau 5-2).

Tableau 5-2 : Temps d'exécution (nb cycles) pour la fonction WRMS en fonctions du nombre de longueurs d'ondes.

Nb Longueurs d'onde	Dist_WRMS	Temps total	
16	1 117 462	1 325 966	84,2%
64	4 218 255	4 796 822	87,9%

L'augmentation du nombre de longueurs d'onde augmente le nombre de cycles de la fonction distance WRMS d'un facteur proche. Ainsi multiplier le nombre de longueurs d'onde par 4 a augmenté le temps de calcul de la fonction d'un facteur 3,775. Cette différence s'explique par le fait que les fonctions de calcul possèdent une phase de réception des données et une phase d'envoi des données. Ainsi il parait plus efficace de traiter un grand nombre de longueurs d'onde, le facteur d'accélération devenant meilleur. On observe que cette fonction reste contraignante sur le temps d'exécution total, il représente entre 84,2 et 87,9 % du temps de calcul total. Il conviendrait alors de découper la fonction WRMS sur plusieurs processeurs en exploitant le parallélisme de donnée massif de cette fonction ou de réaliser un bloc d'accélération de calcul pour cette fonction. Les contraintes supplémentaires pour ces deux solutions seraient de modifier le placement des tâches sur les différents processeurs (augmentant probablement la taille de l'architecture MPSoC) ou d'utiliser une plateforme MPSoC hétérogène constitué de processeurs et d'accélérateurs matériel. Ce type de plateforme n'est pas supporté dans l'environnement HeMPS.

5.2.4 Expérimentations sur une architecture MPSoC de taille 2x3

Nous proposons de modifier les paramètres de l'architecture pour analyser l'impact sur les performances de l'application. L'architecture choisie est une architecture MPSoC avec une taille de NoC de 2x3. Le nombre de fonctions étant supérieur au nombre de processeurs, il est nécessaire de placer plusieurs fonctions sur un seul et même processeur. Le choix du placement se fait sur une analyse du graphe flot de données de l'application représenté en Figure 5-10. Ce graphe permet de voir les dépendances de données entre les tâches et d'extraire les sources de parallélisme potentiel.

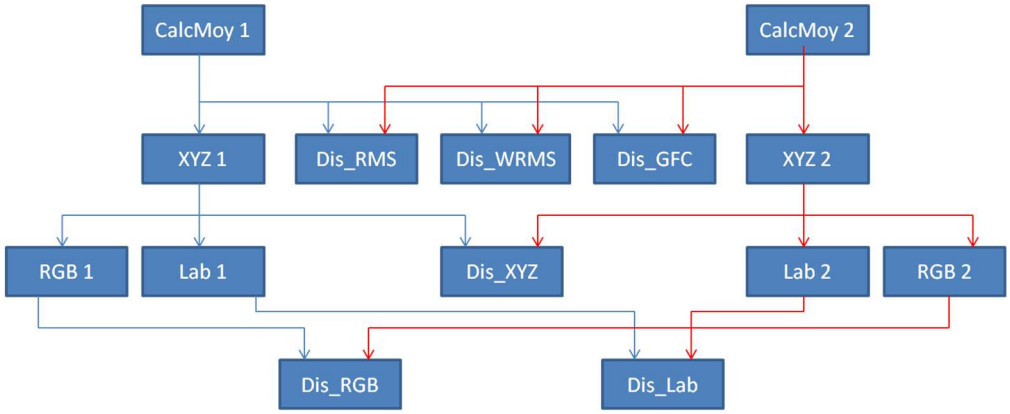


Figure 5-10 : graphe flot de données de l'application.

Les fonctions numérotées en « 1 » présentent les fonctions pour l'image originale et « 2 » pour l'image à comparer. On voit que de nombreuses dépendances sont présentes pour les calculs sur l'image originale et sur l'image à comparer. Le graphe flot de données possède 14 fonctions et un maximum de 5 fonctions potentiellement parallèles (Dis_RMS, Dis_WRMS, Dis_GFC, XYZ1, XYZ2) hormis le pipeline de l'application. Sur une architecture MPSoC 2x3, le nombre de tâches maximum par processeur est de 3 avec une taille maximum de mémoire instruction de 32KB et une taille de mémoire de données de 128KB. Les fonctions sont regroupées comme dans la Figure 5-11 avec les tâches correspondant à :

- La tâche P contient le calcul de la moyenne, le calcul XYZ et le calcul RGB.
- La tâche RDW contient le calcul de distance RMS, le calcul de distance WRMS ainsi que le calcul de distance GFC.
- La tâche LAB est la fonction de projection Lab.
- La tâche DLAB est le calcul de distance Lab.
- La tâche DRGB est le calcul de distance RGB.
- La tâche DXYZ est le calcul de distance XYZ.

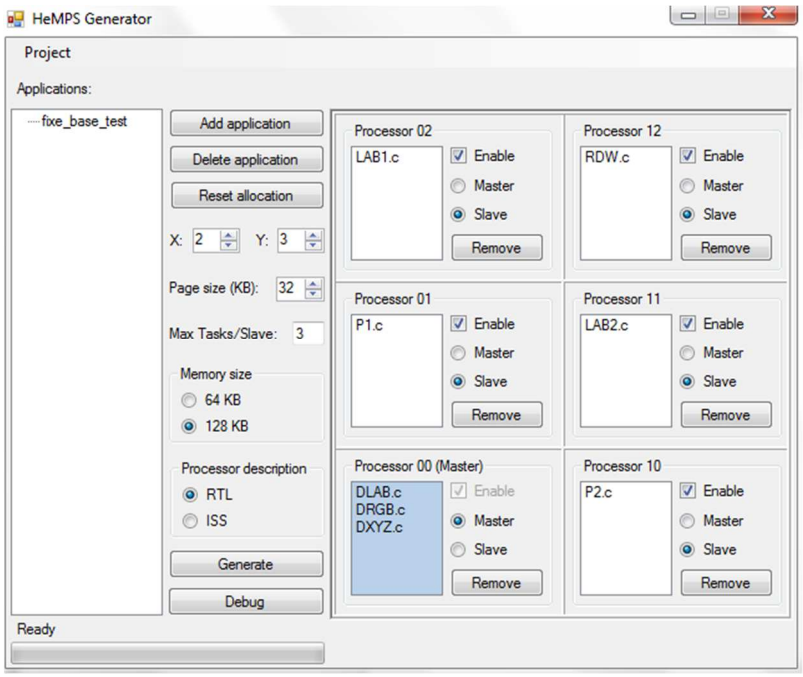


Figure 5-11 : Placements des tâches sur l'architecture 2x3 par l'utilisateur

Le processeur 00 est défini comme le processeur maître dans cette architecture. Les calculs de distance sont placés sur le processeur maître pour permettre un placement dynamique sur un processeur esclave. L'outil de débogage de HeMPS, Figure 5-12, indique que les calculs de distance RGB et LAB sont placés sur le processeur (01) alors que le calcul de distance XYZ a été placé sur le processeur (10) pendant l'exécution.

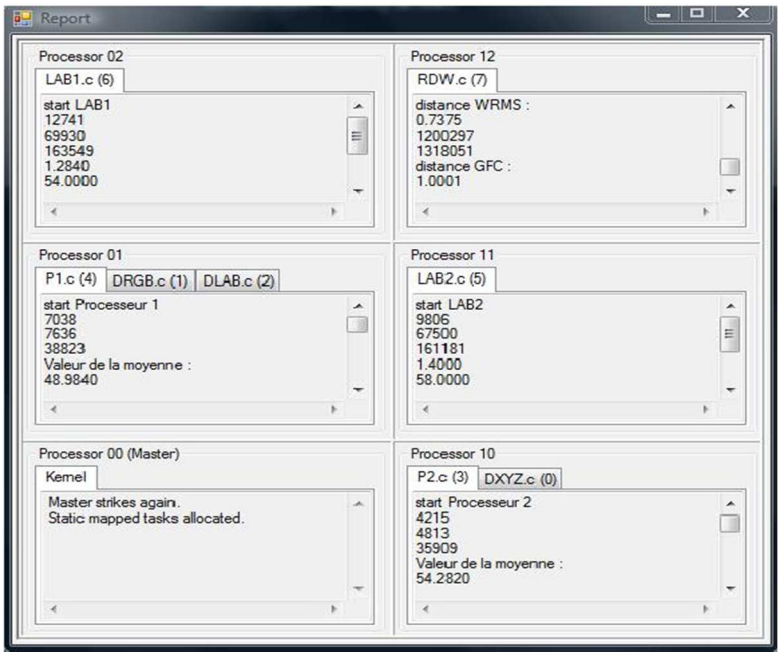


Figure 5-12 : Rapport de débogage indiquant le placement final de l'ensemble des tâches.

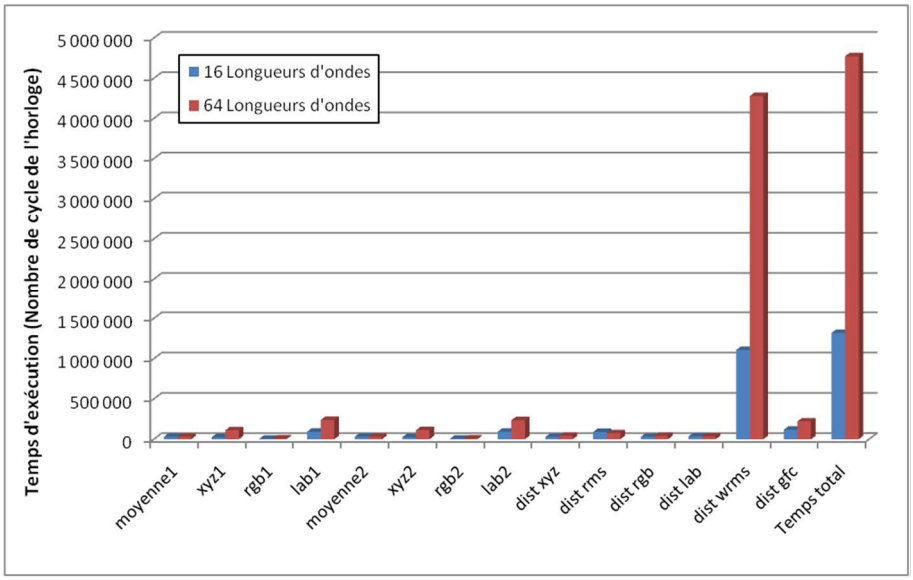


Figure 5-13 : Les temps d'exécution pour chaque fonction en fonction du nombre de longueurs d'ondes

La Figure 5-13indique les temps d'exécution (en nombre de cycles) de chacune des fonctions avec une variation du nombre de longueurs d'ondes pour une taille de région de 8x8 pixels. On observe également que la fonction de distance WRMS est une latence de calcul très élevée par rapport aux autres fonctions. De la même manière la multiplication par 4 du nombre de longueurs d'onde entraîne une augmentation des latences des autres fonctions d'un facteur proche de 4.

La Figure 5-14 présente les temps d'exécution pour chaque calcul en fonction de la taille de la région (8x8 et 16x16) pour 64 longueurs d'onde. Seule le calcul de la moyenne varie en fonction de la taille de la région, ce qui reste cohérent puisque la taille de la région est uniquement utilisée pour le calcul de la moyenne.

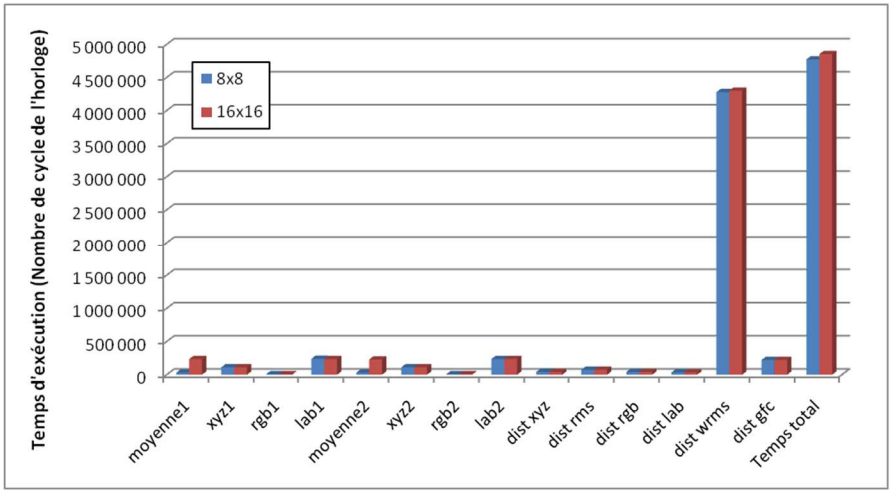


Figure 5-14 : Les temps d'exécution pour chaque fonction selon la taille de la région.

En plus des performances temporelles, nous nous sommes également intéressés à la précision des résultats pour l'application d'authentification. Les données sont traitées en virgule fixe sur le processeur embarqué. La précision est analysée par rapport à l'exécution de l'algorithme sur un CPU qui a considéré les données en virgule flottante 64 bits.

Tableau 5-3 : Comparaison des valeurs des résultats de références sur CPU et les valeurs des résultats obtenus sur l'architecture HeMPS

	Fonction	Référence	Résultat sur l'architecture HeMPS	erreur (%)
Données d'origine	moyenne	48,9847	48,984	0,0014
	X	465,713	464,8841	0,1780
	Y	60,38	60,3451	0,0578
	Z	2291,4663	2290,498	0,0423
	R	1085,1636	1082,2704	0,2666
	G	-366,7739	-364,4709	0,6279
	B	2099,1743	2077,3411	1,0401
	L	82,0601	82,02	0,0489
	A	426,1137	426	0,0267
	B	-382,5499	-383	0,1177
Données à comparer	moyenne	54,2826	54,282	0,0011
	X	516,0825	515,1657	0,1776
	Y	66,9103	66,8722	0,0569
	Z	2539,3015	2538,234	0,0420
	R	1202,5302	1199,3267	0,2664
	G	-406,4426	-403,8911	0,6278
	B	2326,2119	2302,0273	1,0397
	L	85,4722	85,384	0,1032
	A	440,938	441	0,0141
	B	-395,858	-396,2	0,0864
Calcul de distance	XYZ	252,9862	252,85	0,0538
	RGB	258,64	256,3455	0,8871
	RMS	0,6622	0,662	0,0302
	WRMS	0,7378	0,7375	0,0407
	LAB	20,2119	20,2623	0,2494
	GFC	1	1,0001	0,0100

Les paramètres de l'application sont 64 longueurs d'ondes, une fenêtre de 8x8 pixels pour une seule région. Les données utilisées pour l'expérimentation ont des valeurs largement inférieures à 0, la

précision de calcul utilisée en virgule fixe est de 4 décimales (16 bits pour la partie entière et 16 pour la partie décimale). Sur le Tableau 5-3 sont représentées les données obtenues sur un CPU servant de référence et sur un processeur plasma de la plateforme HeMPS ainsi que les erreurs de précision. Les erreurs sur les précisions sont inférieures à 1,1%, avec une erreur moyenne totale de l'application aux alentours de 0.21%, ce qui permet d'affirmer que l'architecture MPSoC sur FPGA peut être utilisée dans notre contexte d'authentification.

5.2.4.1 Expérimentations sur la variation des paramètres de l'architecture

L'expérimentation suivante consiste à faire varier les paramètres de l'architecture, plus précisément le nombre de processeurs. Pour chaque taille d'architecture, les tâches sont réécrites pour être placées sur les processeurs. Ces placements sont manuels, aucun algorithme permettant de définir des optimisations de placement n'a été pris en compte. L'application utilise comme paramètres une taille de région de 8x8 pixels pour 16 longueurs d'ondes. La taille d'architecture est varie de 1x2 jusqu'à 4x4 processeurs.

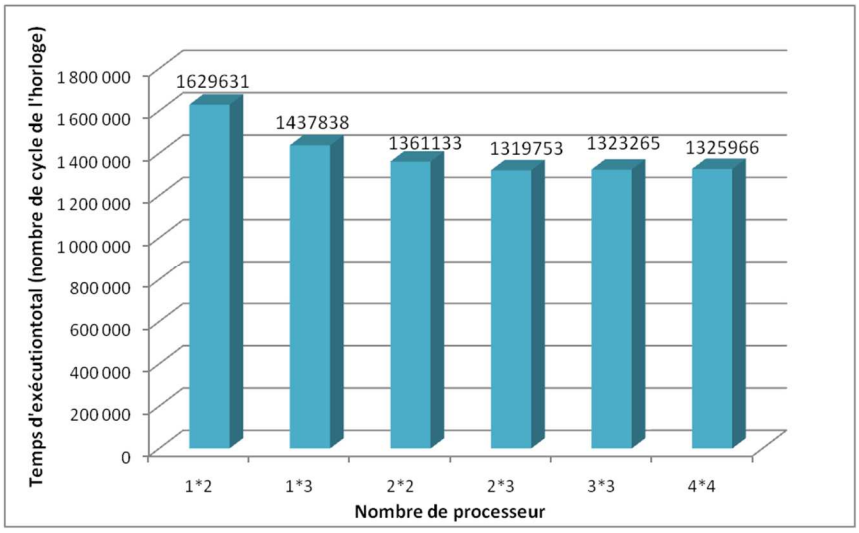


Figure 5-15 : Temps d'exécution en fonction du nombre de processeurs.

Le temps d'exécution total (en nombre de cycles) pour différentes tailles d'architecture est donné dans la Figure 5-15. Le nombre de cycles se réduit nettement pour un nombre de processeurs allant de 2 à 6. A partir de 6 processeurs, les temps d'exécutions deviennent stables. Ceci s'explique par le fait que paralléliser plus ralentit le temps total car les fonctions d'envoi et de réception deviennent nombreuses et gourmandes en temps de cycles par rapport aux fonctions elles-mêmes. L'architecture MPSoC avec 6 processeurs générée par HeMPS semble être un des meilleurs compromis d'architectures pour l'application d'authentification d'œuvres d'art. Ainsi, dans les

sections suivantes, l'architecture MPSoC avec 2x3 processeurs est utilisée pour les expérimentations.

5.2.5 Implantation de la plateforme d'émulation HeMPS MPSoC sur FPGA.

L'architecture MPSoC générée par HeMPS est synthétisable et une plateforme d'émulation associée peut être implantée sur FPGA [82]. Cependant, la plateforme d'émulation nécessite des ressources logicielles et matérielles en plus de l'architecture générée par HeMPS. Une mémoire DDR2 supplémentaire est nécessaire pour mémoriser les tâches non affectées sur un processeur esclave « task repository »

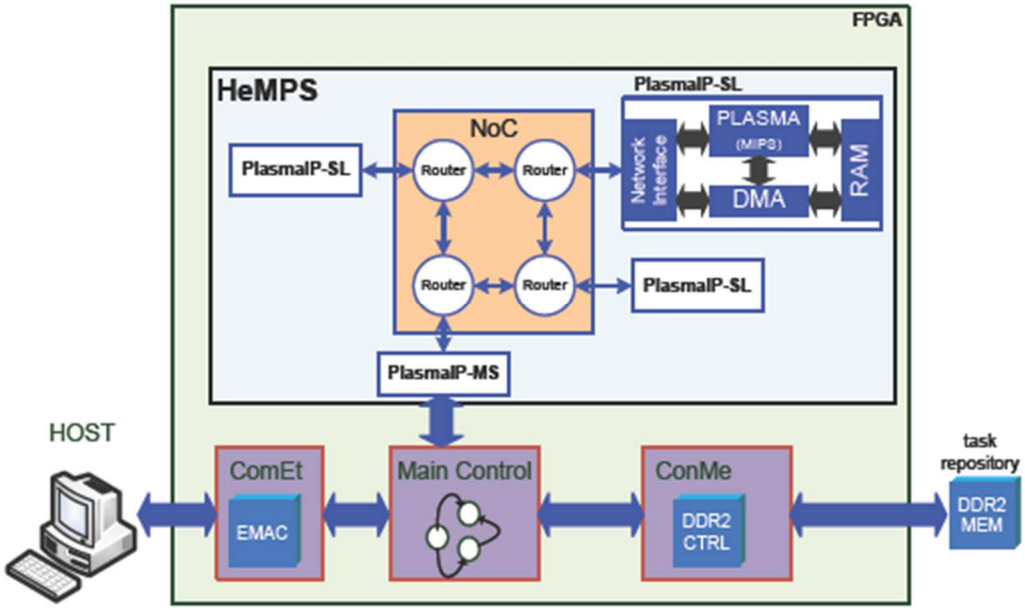


Figure 5-16 : La plateforme d'émulation de HeMPS sur FPGA.

L'architecture d'émulation sur FPGA contient en plus une unité de communication pour la mémoire DDR2 (ConME), un contrôleur principal, une unité de communication entre l'ordinateur hôte et le contrôle principal (ComET).Toutes les applications devant être exécutées par HeMPS sont initialement stockées dans la mémoire externe DDR2.

- L'unité ConME est composée de deux blocs : 1) une interface qui traduit les commandes, les adresses ou des données à partir du contrôle principal pour le contrôleur mémoire DDR2, 2) un contrôleur de DDR2généréepar l'outil Coregen de Xilinx.
- L'unité ComET est responsable de la communication entre le MPSoCet l'ordinateur hôte. Le protocole TCP/IP est un composant matériel. Ce module ComET est composé d'un

module de transmission, un module de réception et d'un noyau Ethernet MAC. Les modules communiquent via l'Ethernet MAC et le module de contrôle principal.

- Le contrôle principal est responsable du contrôle des interactions entre les unités. Il supporte les fonctions suivantes : 1) un processus de commandes reçues venant de ComET, 2) une écriture / lecture de données sur/ de ConME, 3) une transmission des codes des tâches 4) un transfert de message pour le débogage de HeMPS sur ComET.

5.3 Analyse des expérimentations

Les placements-routages de l'architecture sont réalisés par l'outil Xilinx PlanAhead. L'architecture MPSoC est tout d'abord implantée sur la plateforme Xilinx ML506 utilisée précédemment. La synthèse d'architecture, Figure 5-17 nous permet d'analyser les ressources utilisées sur FPGA.

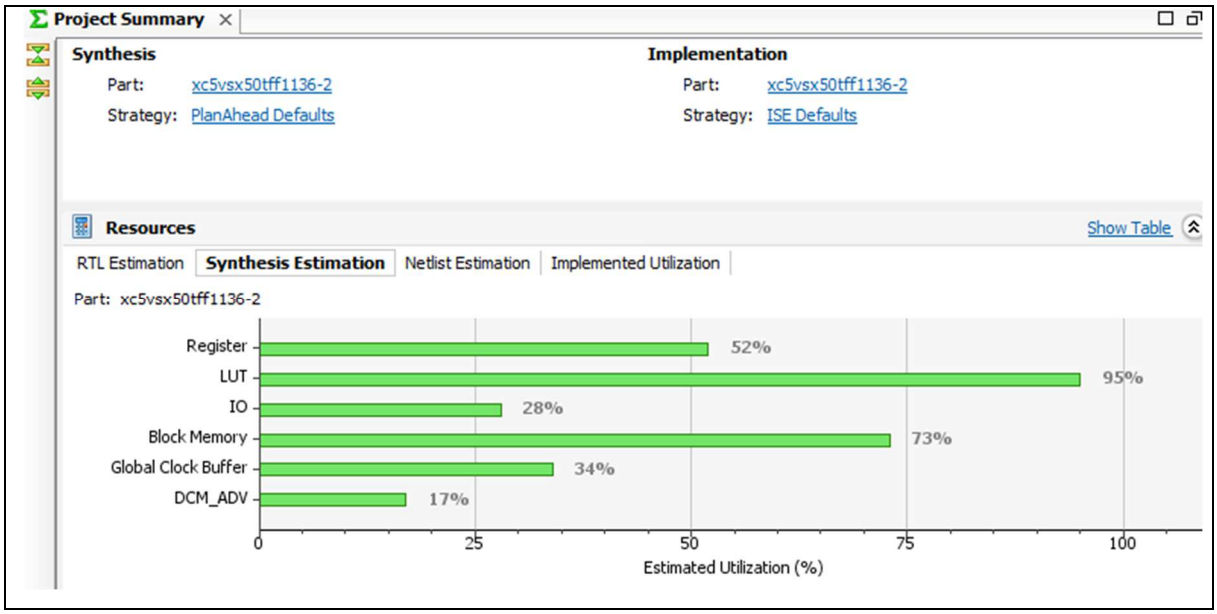


Figure 5-17 : Rapport de synthèse pour FPGA xc5vsx50t

Une architecture MPSoC 2x3 utilise 95% des LUTs disponibles (soit environ 31008 LUTs sur les 32640 disponibles) après synthèse. Le pourcentage du nombre de registres utilisé est de 52% soit 1696 registres. En considérant la taille du NoC uniquement, les ressources étaient de 2681 registres (8.21%) et de 9858 LUTs (30%). Les 16 processeurs utilisent donc environ 44% des registres et 65% des LUTs nécessaires. Cette architecture MPSoC n'est cependant pas implantable sur la ML506, le nombre de blocs RAM est insuffisant comme le montre la Figure 5-18. Il manque 2 blocs mémoires.

```
ERROR:Place:836 - Not enough free sites available for the components of the
   following type(s).
      BLOCKRAM      Number of Components 266    Number of Sites 264
```

Figure 5-18 : Rapport du placement routage de la plateforme MPSoC NoC sur Virtex 5.

L'utilisation d'une plateforme ML605 intégrant un FPGA XC6VLX240T a été réalisée. Ce FPGA virtex 6 possède 150720LUTs et 301440 registres, soit beaucoup plus de ressources que le Virtex 5. Le rapport de synthèse, Figure 5-19 indique que l'architecture MPSoC utilise 3% des registres et 16 % de LUTs. D'un point de vue mémoire, il possède suffisamment de blocs RAM pour supporter la plateforme complète.

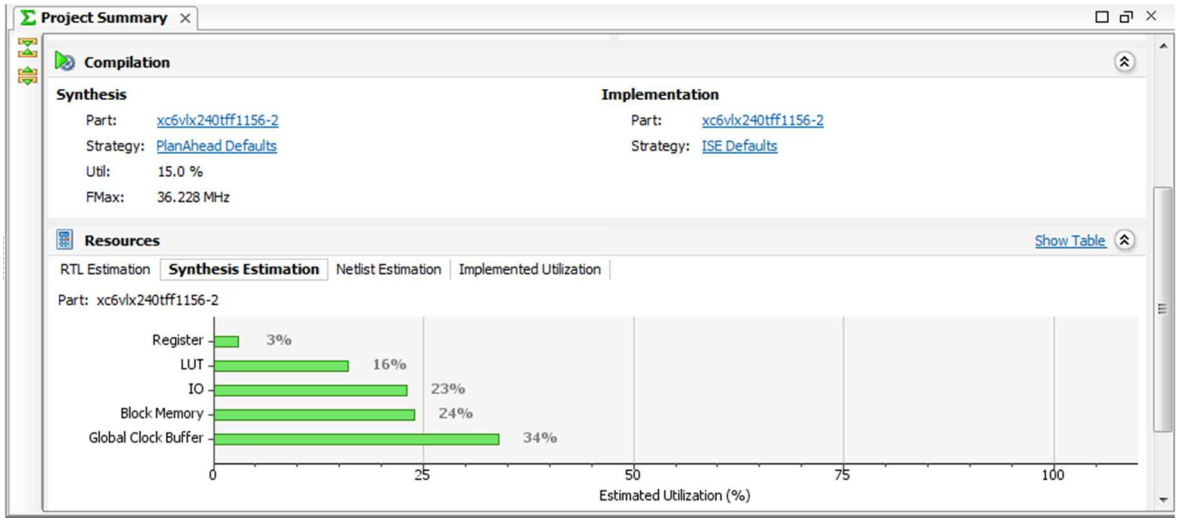


Figure 5-19 : Rapport de synthèse pour la plateforme MPSoC NoC sur FPGA xc6vlx240T.

5.4 Analyse des résultats

Nous avons montré qu'il existe de nombreuses architectures et plateformes intégrant des processeurs homogènes et hétérogènes. Cependant peu d'architectures intègrent un NoC pour la communication entre les cœurs de calcul. A partir de toutes ces expérimentations sur une plateforme existante MPSoC à base de NoC, plusieurs analyses peuvent être faites :

- Il n'existe pas de plateforme d'émulation pour des architectures MPSoC hétérogènes et intégrant une architecture de communication de type NoC. L'hétérogénéité concerne des types de processeurs différents ainsi que des blocs IP dédiés. Les algorithmes de traitement de signal et d'images intègrent des fonctions bas niveau, moyen et haut niveau qui ne nécessitent pas le même type de cœur de calcul. Dans notre cas, l'insertion de blocs

d’accélération pour la fonction WRMS serait un ajout considérable dans les performances espérée de la plateforme.

- L’évaluation des performances concerne principalement des fonctions de traitement, la communication n’est pas prise en compte pour permettre l’évaluation complète de l’application. Dans notre cas, nous avons conçu une plateforme d’émulation pour estimer les performances de la communication sur FPGA et utilisé une plateforme d’émulation pour estimer les temps de calcul des fonctions. Une plateforme complète permettant une évaluation complète reste indispensable.
- HeMPS ne possède pas d’algorithme de placement des tâches sur les différents processeurs permettant d’aider l’utilisateur. Le parallélisme de flux et de tâches sont des sources de parallélisme facilement exploitables pour une application de traitement d’images. Le parallélisme de données est plus difficile à exploiter sur une telle plateforme (contrairement aux GPU).
- Le portage de l’application nécessite un changement de format des données fixe sur 16 bits. Cette limitation restreint l’espace de conception du NoC et de la plateforme complète. La structure du NoC permet de transférer des données de type et format différent. Cet avantage n’est pas exploité par la structure des processeurs et de la plateforme restreignent à un seul format de données.

Chapitre 6. Conclusions et perspectives

6.1 Conclusion

Les applications temps réel embarquées du signal et de l'image nécessitent à l'heure actuelle des architectures performantes de type MPSoC à base de NoC généralement ad hoc pour répondre aux besoins des applications. Les CPUs ou GPUs présentent des caractéristiques de traitement de calcul intéressantes pour certaines applications mais ils possèdent tous deux des inconvénients importants qui ne les rendent pas bien adaptés aux algorithmes évolués de traitement d'images développés au jour d'aujourd'hui. Dans une architecture MPSoC à réseau NoC, les cœurs de calcul ainsi que le réseau d'interconnexion intègrent un grand nombre de paramètres qui ouvrent de manière très large l'espace de conception de l'architecture finale. L'émulation de ces plateformes devient inévitable pour permettre les analyses et les explorations en un temps réduit avec un niveau fin de précision pour obtenir des résultats pertinents.

Dans cette thèse, nous proposons des plateformes d'émulation sur FPGA répondant aux contraintes citées précédemment. La plateforme d'émulation du NoC permet d'explorer les paramètres du NoC au moyen d'un nombre restreint d'implantations sur FPGA pour des scénarii d'une application de traitement du signal et de l'image. La plateforme d'émulation du NoC se base sur la structure du NoC à laquelle sont connectés un ensemble de blocs d'émulation conçus en VHDL synthétisable. Le flot proposé est ouvert et permet d'émuler différentes structures de NoCs sans nécessiter une reconception complète du flot. Quelques adaptations portant notamment au niveau des blocs d'adaptation d'interface (NI) peuvent être nécessaires et rajoutées facilement dans le flot. Les blocs d'émulation développés permettent l'analyse des performances pour des scénarii d'une application et ils permettent également l'exploration des différents paramètres de la plateforme pour définir une des architectures les plus appropriées aux contraintes de l'application et de la plateforme de prototypage. Pendant l'exploration, les blocs d'émulation font varier de manière automatique certains paramètres concernant les transferts de données pour aider l'utilisateur à paramétrer l'architecture de communication pour ensuite émuler la plateforme dimensionnée dans un contexte d'évaluation.

Les diverses expérimentations réalisées sur une structure de NoC, notamment autour du NoC Hermes, montrent la nécessité d'une exploration complète et d'une évaluation précise sur FPGA.

Nous avons montré que la taille des flits, la taille du NoC, la taille des paquets, le choix de la charge du réseau et de l'algorithme de routage ont tous une influence non négligeable sur les performances temporelles et les ressources du FPGA. Une combinaison précise de ces paramètres est appropriée pour un scénario donné mais chaque scénario possède sa propre combinaison « gagnante ». Il n'existe pas une combinaison idéale de ces paramètres et il est donc nécessaire d'évaluer l'ensemble des scénarii de l'application pour trouver le meilleur compromis sur tous ces paramètres pour dimensionner au plus juste l'architecture finale. Nous avons également démontré que l'émulation concerne des plateformes de taille importante avec un nombre de ressources élevé. Il s'avère nécessaire dans de nombreux cas d'utiliser plusieurs FPGAs pour supporter toute la plateforme et ce type de plateforme sera de plus en plus utilisé avec l'augmentation massive des architectures dites « manycore » car possédant plusieurs centaines de cœurs.

Le flot de conception dédié à la génération automatique de la plateforme d'émulation a permis ensuite de définir les performances des communications de l'application d'imagerie spectrale sur FPGA pour l'authentification d'œuvres d'art de manière précise et rapide.

Nous avons également étudié les plateformes d'émulation d'architectures MPSoC existantes dans la communauté et réalisé des expérimentations sur une plateforme d'émulation de MPSoC à base du NoC Hermes. Ces évaluations ont permis de définir une architecture MPSoC adaptée à l'application d'imagerie spectrale et de simuler les temps de calcul des fonctions sur le processeur plasma embarqué. Cependant le placement des tâches est une étape manuelle dépendant de l'expérience de l'utilisateur et la description des tâches sur les processeurs nécessite de réécrire la structure générale de l'application en fonction du découpage de cette application sur les processeurs. Nous avons également observé qu'une architecture homogène n'est pas la solution la plus adaptée à des applications complètes de traitement d'image. Ces applications d'image contiennent pour la majorité des fonctions de bas, moyen et haut niveau. Chacun de ces trois types possède des sources de parallélismes différents ce qui a pour conséquence d'utiliser soit des composants matériels soit des composants logiciels et ceci en fonction du type de la fonction. La définition d'une architecture mixte logicielle matérielle pour ces applications a été établie depuis plusieurs décennies et doit être considérées dans ces architectures MPSoC à base de NoC.

6.2 Perspectives

Les perspectives de ces travaux sont conséquentes en terme d'applications et de conception de plateforme d'émulation MPSoC basée sur un NoC.

Dans un premier temps, une étude autour des architectures mémoire pour ce type d'application est nécessaire. En effet, les traitements et les échanges de données sont importants entre les différentes unités de calcul, et on peut légitimement prévoir qu'un placement efficace des données en mémoire améliorerait les performances générales. Cette problématique n'a pu être abordée faute de temps.

Ensuite, un travail d'exploration de l'espace de conception de l'architecture complète finale est envisagé ainsi que la proposition d'une plateforme d'émulation de l'architecture MPSoC complète. La plateforme permettra d'émuler le comportement complet de l'application, à savoir le temps de calcul des fonctions exécutés sur les processeurs embarqués en identifiant les temps de chargement et le temps de calcul lui-même ainsi que les temps de communication entre les cœurs. L'émulation de la plateforme finale se fera en fonction des différents paramètres de l'algorithme. Cette plateforme devra également permettre de réaliser des explorations de l'espace de conception de l'architecture finale. Ces blocs doivent permettre de réaliser des variations sur les scénarii de communication (comme la variation de la charge du réseau qui est faite automatiquement) ainsi que des variations relatives au placement des tâches et de divers degré de parallélisme. On peut envisager d'orienter le travail sur un placement dynamique partiel ou total des tâches.

Cette plateforme d'émulation devra considérer la possibilité d'un déploiement sur des circuits 3D ou des plateformes multi-FPGAS permettant toujours de réaliser des évaluations ou des explorations pertinentes.

Les plateformes MPSoC hétérogènes étant les plus adaptées aux applications ciblées, l'évolution de la plateforme d'émulation finale se fera en considérant la possibilité d'intégration de blocs d'accélération de calcul (câblés ou non) ou des structures de processeurs différentes comme cela est supporté dans de nombreuses plateformes MPSoC actuelles.

La mise au point d'un environnement logiciel pour la prédiction et l'exploration des performances de timing et de ressources sur FPGA est également envisagée.

D'un point de vue plus général sur l'utilisation des NoCs, on constate une limitation de ces architectures NoC. On commence à être confronté aux premières limitations de ce type d'architecture dû au nombre croissant d'unités de calcul et à leur hétérogénéité, ce qui augmente la quantité et l'hétérogénéité des flux dans le réseau. Le Noc étant extrait du modèle OSI des réseaux informatiques, il serait intéressant de suivre l'analogie avec les réseaux informatiques vers les réseaux du futur. Une perspective serait de tester l'adéquation des paradigmes des réseaux informatiques du futur (Next Generation Networks) pour les réseaux sur puces.

D’un point de vue applicatif, nous souhaitons valider les plateformes avec d’autres algorithmes en imagerie couleur ou spectrale. Une application qui nous semble pertinente et intéressante est la reconstruction 3D couleur. Cela permet de vérifier que l’ensemble des possibilités applicatives ont été considérées lors de la conception des plateformes d’émulation.

Des perspectives de collaborations nationales et internationales ont été lancées lors de cette thèse grâce à l’implication de chercheurs et de doctorants, notamment sur la partie internationale :

- l’équipe de F. Moraes au PURC, Porte Allegre pour la partie plateforme d’émulation MPSoC,
- l’équipe de N. Filali Merchaoui, Monastir Tunisie pour le développement des applications de reconstruction 3D couleur,
- l’équipe de S. Yao, Tianjin Chine pour la conception d’architecture de NoC émergentes.

Liste des Publications

Publications internationales

- J. Tan, L. Zhang, V. Fresse, A. C. Legrand, D. Houzet, "A predictive and parametrized architecture for image analysis algorithm implementations on FPGA adapted to multispectral imaging".The international Workshops on Image Processing Theory, Tools and Applications, Sousse, Tunisia, November 23-26, 2008.Page(s): 1-8.
- V. Fresse, J. Tan, F. Rousseau, "Exploration of an adaptive NoC architecture on FPGA dedicated to multi and hysperspectral algorithm for art authentication". The international conference on Image Processing Theory, Tools and Applications, Paris, FRANCE, July 7-10, 2010.Page(s): 529-534.
- J. Tan, V. Fresse, F. Rousseau, "Generation of emulation platforms for NoC exploration on FPGA". International Symposium on Rapid System Prototyping (RSP), in Karlsruhe, Germany on the Karlsruhe Institute of Technology (KIT), May 24-27, 2011.Page(s): 186-192.
- J. Tan, V. Fresse, F. Rousseau, "From Mono-FPGA to Multi-FPGA Emulation Platform for NOC Performance Evaluations" International Conference on Parallel Computing 30 August - 2 September 2011Ghent, Belgium.

Colloques nationaux

- J. Tan, V. Fresse, F. Rousseau, "Vers la génération automatique d'architectures efficaces pour des applications d'imagerie multispectrale". Ecole d'hiver Francophone sur les Technologies de Conception des Systèmes embarqués Hétérogènes, Hôtel Préalpina, Chexbres, Suisse, 12-14 janvier 2009.
- J. Tan, V. Fresse, F. Rousseau, "Vers une architecture autour du NoC Hermes pour des applications d'imagerie multispectrale". Ecole d'hiver Francophone sur les Technologies de Conception des Systèmes embarqués Hétérogènes, 11-13 janvier 2010 Hôtel Alpina, Chamonix - Mont Blanc, France
- J. Tan, V. Fresse, F. Rousseau, "Exploration d'une architecture NoC adaptable sur FPGA pour des applications d'imagerie multispectrale et hyperspectrale". Le quatrième colloque du GDR SOC-SIP du CNRS les 09-10-11 Juin à Paris-Cergy (ENSEA), 2010.

Bibliographie

[1] ITRS Technical Report “ITRS Assembly & Packaging ReportMore than Moore Initiative”. Juillet 200 8, www.itrs.net/

[2] Raj,J.; , "More Moore or More Than Moore?”.SEMATECH Symposium Taiwan, September 7, 2010.

[3] Gupta,R.K.; , "Hardware-software co-design: Tools for architecting systems-on-a-chip," *Design Automation Conference 1997. Proceedings of the ASP-DAC '97. Asia and South Pacific*, vol., no., pp.285-289, 28-31 Jan 1997. Doi: 10.1109/ASPDAC.1997.600157

[4] Buchenrieder, K.; Sedelmeier, A.; Veith, C.; , "Industrial HW/SW codesign,"*Hardware/Software Co-Design*, G. De Micheli and M. Sami, Eds. Amsterdam: Kluwer, 1996, pp. 453-466.

[5] Bolsens, I.; De Man, H.J.; Lin, B.; Van Rompaey, K.; Vercauteren, S.; Verkest, D.; , "Hardware/software co-design of digital telecommunication systems ," *Proceedings of the IEEE* , vol.85, no.3, pp.391-418, Mar 1997. Doi: 10.1109/5.558713

[6] Chou, Pai H.;Ortega, Ross B.; Borriello, G.; ,"The Chinook hardware/software co-synthesis system,"*The 8th international symposium on System synthesis, Proceedings ofISSS ’95. New York,NY, USA: ACM, 1995*, pp.22-27. Doi: 10.1145/224486.224491

[7] Chiodo, M.; Giusto, P.;Jurecska, A.; Hsieh, H.; Lavagno, L.; Sangiovanni, A.; , "A formal methodology for hardware/software co-design of embedded systems,"*IEEE Micro*, vol. 14, no. 4, pp: 26-36, Aug 1994.

[8] Wolf, W.; Jerraya, A.A.; Martin, G.; , "Multiprocessor System-on-Chip (MPSoC) Technology," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.27, no.10, pp.1701-1713, Oct. 2008. Doi: 10.1109/TCAD.2008.923415

[9] http://en.wikipedia.org/wiki/Color_image

[10] http://fr.wikipedia.org/wiki/Espace_colorim%C3%A9trique

[11] http://www.linuxgraphic.org/grokking/node50.html

[12] http://helios.univ-reims.fr/Labos/LERI/membre/luc/ENSEIGNEMENT/COURS/TR_IMG/node38.html

[13] http://www.tsi.enst.fr/tsi/enseignement/ressources/mti/median_couleur/col.html

[14] http://www.tsi.enst.fr/tsi/enseignement/ressources/mti/RVB_ou_LAB/html/colorspace.html

[15] Hernández-Andrés, J.; Romero, J.; Lee Jr, R.L.; ,"Colorimetric and spectroradiometric characteristics of narrow-field-of-view clear skylight in Granada, Spain,"*Journal of the Optical Society of America A*, Vol. 18, No. 2, pp:412-420, Feb 2001.

[16] Viggiano,J.A.S.; ,"The Comparison of Radiance Ratio Spectra: Assessing a Model's Goodness of Fit,"*Advanced Printing of Conference Summaries: SPSE's 43rd Annual Conference, 1990. Springfield, VA: SPSE – The Society for Imaging Science and Technolog,*pp: 222-225.

[17] Viggiano, J.A.S.; ,"Perception-referencedmethod for comparison of radianceratiospectra and its application as an index of metamerism,"*Proceedings of SPIE Vol. 4421, 9th Congress of the International Colour Association, Eds*, pp: 701-704.

[18] Imai, F.; Berns, R.; Tzeng, Di-Y.; ,"A Comparative Analysis of Spectral Reflectance Estimated in Various Spaces Using a Trichromatic Camera System," Munsell Color Science Laboratory, Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology, Rochester, New York.

[19] Zhao, Y.; Berns,R.S.; Okumura, Y.;Taplin, L.A.;, "Improvement of Spectral Imaging by Pigment Mapping,"*Thirteenth Color Imaging Conference: Color Science and Engineering Systems, Technologies, and Applications, Scottsdale, Arizona; November 2005*; pp. 40-45; ISBN / ISSN: 0-89208-259-3

[20] http://www.theverge.com/2011/11/3/2535607/cpu-and-gpu-the-convergent-technology

[21] http://en.wikipedia.org/wiki/File:AMD_A64_Opteron_arch.svg

[22] http://www.presence-pc.com/tests/GeForce-GTX-260-280-22792/7/

[23] http://microcontrollershop.com/product_info.php?products_id=2355

[24] Cordan, B.; , "An efficient bus architecture for system-on-chip design," *Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999*, vol., no., pp.623-626, 1999. Doi: 10.1109/CICC.1999.777358

[25] Guerrier, P.; Greiner, A.; , "A generic architecture for on-chip packet-switched interconnections ," *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*, vol., no., pp.250-256, 2000. Doi: 10.1109/DATE.2000.840047

[26] Dally,W.J.; Towles, B.; , "Route packets, not wires: on-chip interconnection networks," *Design Automation Conference, 2001. Proceedings*, vol., no., pp. 684- 689, 2001. Doi: 10.1109/DAC.2001.156225.

[27] Fu, Z.; Ling, X.; , "The design and implementation of arbiters for Network-on-chips," *Industrial and Information Systems (IIS), 2010 2nd International Conference on* , vol.1, no., pp.292-295, 10-11 July 2010. Doi: 10.1109/INDUSIS.2010.5565854

[28] Naeem, A.; Jantsch, A.; Xiaowen Chen; Zhonghai Lu; , "Realization and Scalability of Release and Protected Release Consistency Models in NoC Based Systems," *Digital System Design (DSD), 2011 14th Euromicro Conference on* , vol., no., pp.47-54, Aug. 31 2011-Sept. 2 2011. Doi: 10.1109/DSD.2011.11

[29] RISO,S. "Evaluation des paramètres architecturaux des réseaux sur puce," PhD Thèse. Pp.172.

[30] Moraes, F.; Mello, A.; Möller, L.; Ost, L.; Calazans, N.; , "A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping," *Integration, the VLSI Journal, Special issue: Networks on chip and reconfigurable fabrics,*Volume 38 Issue 1, October 2004, pp.69-93. Doi : 10.1016/j.vlsi.2004.03.003

[31] Benini, L.; De Micheli, G.; , "Powering networks on chips," *System Synthesis, 2001. Proceedings. The 14th International Symposium on*, vol., no., pp. 33- 38, 2001. Doi: 10.1109/ISSS.2001.156528

[32] NGAC, N.; Mancini, S.; Desvignes, M.; Houzet, D.; , "High Speed 3D Tomography on CPU, GPU, and FPGA,"*EURASIP Journal on Embedded Systems 2008, 2008:930250*. Doi:10.1155/2008/930250

[33] http://www-ccrt.cea.fr/fr/collaborations/fichiers/journee_thematique_110408/GPGPU-hydro-110408.pdf

[34] Vieira de Mello, A.; Ost, L.C.;Moraes, F.; Laert, N.; Calazans, V.; , "Evaluation of Routing Algorithms on Mesh Based NoCs," *Technical report series, Mai 2004.*

[35] Adriahantenaina, A.; Charlery, H.; Greiner, A.; Mortiez, L.; Zeferino, C.A.; , "SPIN: a scalable, packet switched, on-chip micro-network," *Design, Automation and Test in Europe Conference and Exhibition, 2003*, vol., no., pp. 70- 73 suppl., 2003. Doi: 10.1109/DATE.2003.1253808

[36] Siguenza-Tortosa, D.; Nurmi, J.; , "VHDL-based simulation environment for Proteo NoC," *High-Level Design Validation and Test Workshop, 2002. Seventh IEEE International*, vol., no., pp. 1- 6, 27-29 Oct. 2002. Doi: 10.1109/HLDVT.2002.1224419

[37] Liang, J.; Swaminathan, S.; Tessier, R.; , "ASOC: a scalable, single-chip communications architecture," *Parallel Architectures and Compilation Techniques, 2000. Proceedings. International Conference on*, vol., no., pp.37-46, 2000. Doi: 10.1109/PACT.2000.888329

[38] Dally, W.J.; Towles, B.; , "Route packets, not wires: on-chip interconnection networks," *Design Automation Conference, 2001. Proceedings*, vol., no., pp. 684- 689, 2001. Doi: 10.1109/DAC.2001.156225

[39] Pande, P.P.; Grecu, C.; Ivanov, A.; Saleh, R.; , "Design of a switch for network on chip applications," *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 Interna-*

tional Symposium on, vol.5, no., pp. V217- V-220 vol.5, 25-28 May 2003. Doi: 10.1109/ISCAS.2003.1206235

[40] Durand, Y.; Bernard, C.; Lattard, D.; , "FAUST: On-Chip Distributed Architecture for a 4G Baseband Modem SoC," *Proceedings of Design and Reuse IP-SOC'2005, Grenoble, France,*pp.51-55.

[41] Clermidy, F.; Bernard, C.; Lemaire, R.; Martin, J.; Miro-Panades, I.; Thonnart, Y.; Vivet, P.; Wehn, N.; , "MAGALI: A Network-on-Chip based multi-core system-on-chip for MIMO 4G SDR," *IC Design and Technology (ICICDT), 2010 IEEE International Conference on* , vol., no., pp.74-77, 2-4 June 2010. Doi: 10.1109/ICICDT.2010.5510291

[42] Ngan,N.; , "Etude et conception d'un réseau sur puce dynamiquement adaptable pour la vision embarquée," *PhD thesis, ESIEE Paris*.

[43] Ye,T.T.; ,"On-Chip Multiprocessor Communication Network Design And Analysis," *PhD thesis, Standford University, 2003.*

[44] Salminen, E.; Kulmala, A.; Hamalainen, T.D.; , "On network-on-chip comparison," *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, vol., no., pp.503-510, 29-31 Aug. 2007. Doi: 10.1109/DSD.2007.4341515

[45] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L.; ,"Hermes : an infrastructure for low area overhead packet-switching networks on chip,"*Integration, the VLSI Journal, Volume 38, Issue 1*. Doi : DOI:10.1016/j.vlsi.2004.03.003

[46] Hilton, C.; Nelson, B.; , "A flexible circuit switched NOC for FPGA based systems," *Field Programmable Logic and Applications, 2005. International Conference on*, vol., no., pp. 191- 196, 24-26 Aug. 2005. Doi: 10.1109/FPL.2005.1515721

[47] Jovanovic, S.; Tanougast, C.; Weber, S.; Bobda, C.; , "CuNoC: A Scalable Dynamic NoC for Dynamically Reconfigurable FPGAs," *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, vol., no., pp.753-756, 27-29 Aug. 2007. Doi: 10.1109/FPL.2007.4380761

[48] Jovanovic, S. ; , "Architecture reconfigurable de systèmes embarqué auto-organisé," PhD thesis, Université Henry Poincaré - Nancy 1, Laboratoire d'Instrumentation Electronique de Nancy (LIEN), 2009.

[49] Majer, M.; Bobda, C.; Ahmadinia, A.; Teich, J.; , "Packet Routing in Dynamically Changing Networks on Chip," *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, vol., no., pp. 154b, 04-08 April 2005. Doi: 10.1109/IPDPS.2005.323.

[50] Mello, A.; Tedesco, L.; Calazans, N.; Moraes, F.; , "Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC," *Integrated Circuits and Systems*

Design, 18th Symposium on, vol., no., pp.178-183, 4-7 Sept. 2005. Doi: 10.1109/SBCCI.2005.4286853

[51] Marescaux, T.; Bartic, A.; Verkest, D.;Vernalde, S.; Lauwereins, R.; ,"Interconnection networks enable fine-grain dynamic multi-tasking on FPGAs,"*FPL '02 Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications,*vol., no., pp.795-805. ISBN:3-540-44108-5

[52] Zeferino,C.A.; Susin, A.A.; , "SoCIN: a parametric and scalable network-on-chip," *Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on*, vol., no., pp. 169- 174, 8-11 Sept. 2003. Doi: 10.1109/SBCCI.2003.1232824

[53] Hilton, C.; Nelson, B.; , "PNoC: a flexible circuit-switched NoC for FPGA-based systems," *Computers and Digital Techniques, IEE Proceedings*, vol.153, no.3, pp. 181-188, 2 May 2006. Doi: 10.1049/ip-cdt:20050175

[54] Salminen, E.; Kulmala, A.; Hamalainen, T.D.; , "HIBI-based multiprocessor SoC on FPGA," *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, vol., no., pp. 3351- 3354 Vol. 4, 23-26 May 2005. Doi: 10.1109/ISCAS.2005.1465346.

[55] Chan, J.; Parameswaran, S.; , "NoCGEN:a template based reuse methodology for Networks On Chip architecture," *VLSI Design, 2004. Proceedings. 17th International Conference on*, vol., no., pp. 717- 720, 2004. Doi: 10.1109/ICVD.2004.1261011

[56] Benini, L.; , "Application Specific NoC Design," *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol.1, no., pp.1-5, 6-10 March 2006. Doi: 10.1109/DATE.2006.243857

[57] Genko, N.; Atienza, D.; De Micheli, G.; Mendias, J.M.; Hermida, R.; Catthoor, F.; , "A complete network-on-chip emulation framework," *Design, Automation and Test in Europe, 2005. Proceedings*, vol., no., pp. 246- 251 Vol. 1, 7-11 March 2005. Doi: 10.1109/DATE.2005.5

[58] Krasteva, Y.E.; Criado, F.; de la Torre, E.; Riesgo, T.; , "A Fast Emulation-Based NoC Prototyping Framework," *Reconfigurable Computing and FPGAs, 2008. ReConFig '08. International Conference on*, vol., no., pp.211-216, 3-5 Dec. 2008. Doi: 10.1109/ReConFig.2008.74

[59] Liu, P.; Xiang, C.; Wang, X.; Xia, B.; Liu, Y.; Wang, W.; Yao, Q.; , "A NoC Emulation/Verification Framework," *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on*, vol., no., pp.859-864, 27-29 April 2009. Doi: 10.1109/ITNG.2009.197

[60] Ogras, U.Y.; Marculescu, R.; Hyung Gyu Lee; Naehyuck Chang; , "Communication architecture optimization: making the shortest path shorter in regular networks-on-chip,"

Design, Automation and Test in Europe, 2006. DATE '06. Proceedings, vol.1, no., pp.6 pp., 6-10 March 2006. Doi: 10.1109/DATE.2006.244068

[61] Mahadevan, S.; Virk, K.; Madsen, J.; , "Arts: A systemc-based framework for modelling multiprocessor systems-on-chip,"*Proceedings of Design Automation of Embedded Systems, 2006.* Vol. 11, Nr. 4 (2007), pp. 285-311. ISSN：0929-5585

[62] Chan, J.; Parameswaran, S.; , "NoCGEN:a template based reuse methodology for Networks On Chip architecture," *VLSI Design, 2004. Proceedings. 17th International Conference on*, vol., no., pp. 717- 720, 2004. Doi: 10.1109/ICVD.2004.1261011

[63] Jalabert, A.; Murali, S.; Benini, L.; Micheli, G.D.; , "Xpipes Compiler: a tool for instantiating application specific network on chip," *Proceedings of The Design, Automation, and Test in Europe*, 2004.

[64] Kouadri-Mostefaoui, A-M.; Senouci, B.; Petrot, F.; , "Large Scale On-Chip Networks : An Accurate Multi-FPGA Emulation Platform," *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*, vol., no., pp.3-9, 3-5 Sept. 2008. Doi: 10.1109/DSD.2008.130

[65] Li,X.; Hammami, O.; , "Multi-FPGA emulation of a 48-cores multiprocessor with NOC," *Design and Test Workshop, 2008. IDT 2008. 3rd International*, vol., no., pp.205-208, 20-22 Dec. 2008. Doi: 10.1109/IDT.2008.4802498

[66] Seiculescu, C.; Murali, S.; Benini, L.; De Micheli, G.; , "SunFloor 3D: A Tool for Networks on Chip Topology Synthesis for 3-D Systems on Chips," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol.29, no.12, pp.1987-2000, Dec. 2010. Doi: 10.1109/TCAD.2010.2061610

[67] B.S. Feero,B.S.; Pande, P.P.; , "Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation," *Computers, IEEE Transactions on*, vol.58, no.1, pp.32-45, Jan. 2009. Doi: 10.1109/TC.2008.142

[68] Rahmani, A.-M.; Latif, K.; Liljeberg, P.; Plosila, J.; Tenhunen, H.; , "Research and practices on 3D networks-on-chip architectures," *NORCHIP, 2010*, vol., no., pp.1-6, 15-16 Nov. 2010. Doi: 10.1109/NORCHIP.2010.5669453

[69] Xu, T.C.; Yin, A.W.; Liljeberg, P.; Tenhunen, H.; , "A study of 3D Network-on-Chip design for data parallel H.264 coding," *NORCHIP, 2009*, vol., no., pp.1-6, 16-17 Nov. 2009. Doi: 10.1109/NORCHP.2009.5397851

[70] Stepniewska, M.; Luczak, A.; Siast, J.; , "Network-on-Multi-Chip (NoMC) for Multi-FPGA Multimedia Systems," *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, vol., no., pp.475-481, 1-3 Sept. 2010. Doi: 10.1109/DSD.2010.106

[71] Duato, J.; Yalamanchili, S.; Ni,L.; , "Interconnection Networks, An Engineering Approach,"*Ed, Elseveir Science, USA, 2003*.

[72] Bertozzi, D.; Jalabert, A.; Srinivasan Murali; Tamhankar, R.; Stergiou, S.; Benini, L.; De Micheli, G.; , "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *Parallel and Distributed Systems, IEEE Transactions on* , vol.16, no.2, pp. 113- 129, Feb. 2005. Doi: 10.1109/TPDS.2005.22

[73] Kavaldjiev, N.; Smit, G. J. M.; Wolkotte, P. T.; Jansen, P. G.;, "Routing of guaranteed throughput traffic in a network-onchip,"*Report Acquisitions Computer Hardware, November 2005*.

[74] Feero, B.; Pande, P.P.; , "Performance Evaluation for Three-Dimensional Networks-On-Chip," *VLSI, 2007. ISVLSI '07. IEEE Computer Society Annual Symposium on*, vol., no., pp.305-310, 9-11 March 2007. Doi: 10.1109/ISVLSI.2007.79

[75] http://www.xilinx.com/

[76] Lyonnard, D.; Yoo, S.; Baghdadi, A.; Jerraya, A.A.; , "Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip," *Design Automation Conference, 2001. Proceedings*, vol., no., pp. 518- 523, 2001. Doi: 10.1109/DAC.2001.156194

[77] Baghdadi, A.; Zergainoh, N.; Cesario, W.; Roudier, T.; Jerraya, A.A.; , "Design space exploration for hardware/software codesign of multiprocessor systems," *Rapid System Prototyping, 2000. RSP 2000. Proceedings. 11th International Workshop on*, vol., no., pp.8-13, 2000. Doi: 10.1109/IWRSP.2000.854975

[78] Lahiri, K.; Raghunathan, A.; Dey, S.; , "Design space exploration for optimizing on-chip communication architectures," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol.23, no.6, pp. 952- 961, June 2004. Doi: 10.1109/TCAD.2004.828127.

[79] Samahi, A.; Bourennane, E.-B.; , "Automated Integration and Communication Synthesis of Reconfigurable MPSoC Platform," *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, vol., no., pp.379-385, 5-8 Aug. 2007. Doi: 10.1109/AHS.2007.35

[80] Boukhechem,S.; Bourennane, E.-B.; , "TLM Platform Based on SystemC for STARSoC Design Space Exploration," *Adaptive Hardware and Systems, 2008. AHS '08. NASA/ESA Conference on*, vol., no., pp.354-361, 22-25 June 2008. Doi: 10.1109/AHS.2008.17

[81] Ventroux, N.; Guerre, A.; Sassolas, T.; Moutaoukil, L.; Blanc, G.; Bechara, C.; David, R.; , "SESAM: An MPSoC Simulation Environment for Dynamic Application Processing," *Computer and Information Technology (CIT), 2010 IEEE 10th International*

Conference on, vol., no., pp.1880-1886, June 29 2010-July 1 2010. Doi: 10.1109/CIT.2010.322

[82] WACHTER, E.W.; Biazi, A.; Moraes, F.G.; , "HeMPS-S: A homogeneous NoC-based MPSoCs framework prototyped in FPGAs," *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, vol., no., pp.1-8, 20-22 June 2011. Doi: 10.1109/ReCoSoC.2011.5981498

[83] http://opencores.com/project,plasma