UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences

Département d'informatique

THESE DE DOCTORAT
Spécialité : Informatique

AIDE A LA CONCEPTION D'UN RESEAU SUR PUCE POUR UN SYSTEME INTEGRE

Par

Nesrine TOUBALINE

devant le jury composé de :

N. BENBLIDIA	Professeur, Université Blida 1	Présidente
A. GUESSOUM	Professeur, Université Blida 1	Examinateur
Y. CHALLAL	Professeur, ESI, Alger	Examinateur
N. BOUSTIA	MCA, Université Blida 1	Examinatrice
D. BENNOUAR	Professeur, Université de Bouira	Directeur de thèse
A. MAHDOUM	Maître de Recherche C.D.T.A., Alger	Co-Directeur de thèse

Blida, Mai 2018

RESUME

Le paradigme des réseaux sur puce (Networks on Chip-NoC) est né de la nécessité de nouveaux moyens de communication respectant des exigences de plus en plus grandes en performances et coûts, principalement lorsqu'il s'agit de systèmes sur puce (Systems on Chip-SoCs) complexes, intégrant un grand nombre de composants communiquant. Les travaux de cette thèse s'intéressent aux problèmes de performance et de coût des SoCs à base de NoCs. Comme le NoC offre beaucoup de paramètres (topologie, fréquence, chemins, stratégie de routage, horloge etc.), sa conception entraîne un certain nombre de défis à relever afin de répondre aux besoins des applications et satisfaire les exigences des clients. L'espace de conception étant extrêmement large, il est donc nécessaire de disposer d'outils d'aide à la conception afin d'assister et guider le concepteur dans ses choix et l'aider à concevoir le NoC adapté à ses besoins.

La conception d'un réseau sur puce étant une problématique nécessitant beaucoup de ressources humaines, nous avons développé des techniques (basées sur des heuristiques) pour contribuer à la conception de tels réseaux. Ces techniques concernent précisément une réduction et un placement efficace de liens verticaux sur une architecture 3D déjà générée (en vue de réduire la surface et les paramètres électriques parasites) ainsi que l'optimisation du nombre de PLLs générant les fréquences d'horloge relatives aux composants du système. D'autant plus qu'une PLL consomme beaucoup de puissance, ce qui est néfaste pour un système contenant un nombre conséquent de composants fonctionnant à des fréquences différentes.

Mots clés : systèmes sur puce**,** réseaux sur puce, performance, optimisation combinatoire, heuristiques, topologie, génération d'horloges.

ABSTRACT

The Networks on Chip-NoC paradigm arose from the need of new means of communication meeting increasingly high demands on performance and costs, especially when it comes to complexes systems on chip (SoCs), integrating a large number of communicating components. The works of this thesis are interested in the performance and cost problems of SoCs based on NoCs. Since NoC offers many parameters (topology, frequency, paths, routing strategy, clock, etc.), the design of the NoC provides a number of challenges to meet the needs of the applications and satisfy customers requirements. Since the design space is extremely large, it is necessary to have design assistance tools in order to assist and guide the designer in his choices and to help him to design a NoC adapted to his needs.

Since the design of a network on a chip is a problem requiring a lot of human resources, we have developed techniques (based on heuristics) to contribute to the design of such networks. These techniques relate precisely to a reduction and an efficient placement of verticals links on a 3D architecture already generated (in order to reduce the area and the electrical parasitic parameters) as well as the optimization of the number of PLLs generating the clock frequencies relative to the components of the system. Especially since a PLL consumes a lot of power, which is detrimental to a system containing a consequent number of components operating at different frequencies.

Keywords: systems on chip, networks on chip, performance, combinatorial optimization, heuristics, topology, clocks generation.

ملخص

نموذج الشبكات على شريحة إلكترونية ولد من الحاجة إلى وسائل جديدة للتواصل تحترم المطالب العالية على نحو متزايد على الكفاءة و التكاليف، و خصوصا عندما يتعلق الأمر بالأنظمة على شريحة معقدة، تدمج عدد كبير من المكونات المتصلة .أشغال هذه الأطروحة مهتم بمشكلات الكفاء ة و التكلفة في الأنظمة على شريحة القائمة على الشبكات على شريحة .كما الشبكة على شريحة لديها الكثير من المعلمات (طوبولوجيا، والتردد، والمسارات، واستراتيجية التوجيه ، الساعة الخ)، تصميمها لديه عدد من التحديات لتلبية متطلبات التطبيق والعملاء .فضاء التصميم كبير للغاية، ولذلك فمن الضروري توفير أدوات دعم التصميم لتوجيه المصمم في الاختيار ومساعدته في تصميم الشبكة على شريحة مصممة خصيصا لاحتياجاته.

تصميم شبكة على شريحة مشكلة تتطلب الكثير من الموارد البشرية، قمنا بتطوير تقنيات (على أساس الاستدلال) للمساهمة في تصميم هذه الشبكات. هذه التقنيات تتعلق على وجه التحديد بخفض و إيداع فعال للروابط العمودية على بنية 3D منتجة قبلا (للحد من المساحة و المعلمات الكهربائية الطفيلية). و كذلك تحسين عدد PLLs التي تقوم بتوليد ترددات الساعة. وخاصة أن PLL تستهلك الكثير من الطاقة، ما هو ضار للنظام الذي يحتوي على عدد كبير من العناصر التي تعمل على ترددات مختلفة.

الكلمات الجوهرية :أنظمة على شريحة، الشبكات على شريحة، والكفاء ،والتحسين التوافقي، الاستدلال، والطوبولوجيا، توليد الساعات.

REMERCIEMENTS

Cette thèse est le fruit de la patience et la générosité de mes directeurs de thèse, Monsieur Djamel Bennouar, professeur à l'université de Bouira, et Monsieur Ali Mahdoum, chercheur au centre de développement des technologies avancées. Ils ont activement contribué à la réalisation de mes travaux, par leurs conseils efficaces et leurs remarques pertinentes. Qu'ils trouvent ici l'expression de ma reconnaissance sincère.

Je voudrais aussi remercier l'ensemble des membres du jury pour avoir accepté de participer à ma soutenance et d'évaluer mon travail. Un remerciement particulier à Madame N. Benblidia la directrice de notre laboratoire (Laboratoire de Recherche pour le Développement des Systèmes Informatisés) pour ses efforts et animation. Je remercie également le Professeur A. Guessoum du département d'électronique pour ses conseils et encouragements. J'adresse un remerciement à Monsieur Y. Challal professeur de l'Ecole Supérieur d'Informatique d'Alger pour sa disponibilité et son déplacement. Ainsi, j'aimerai remercier Docteur N. Boustia mon exemple de réussite depuis mes études en ingéniorat ; pour son enseignement de qualité, puis sa collaboration et son partage professionnel et enfin pour sa participation au membre de jury de ma soutenance.

Je remercie les membres du Laboratoire de recherche LRDSI ainsi que tous les gens qui ont contribué à l'ouverture de l'habilitation universitaire au niveau de notre département. Je cite aussi les doctorants et doctorantes du département d'informatique, pour le partage de connaissances et les diverses discussions enrichissantes.

Je suis extrêmement reconnaissante envers ma famille, amies et surtout mes collègues de m'avoir soutenu, encouragé pour aller jusqu'au bout de mes objectifs.

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

INTRODUCTION

Suivant la loi de Moore affirmée par le cofondateur de la société Intel Gordon Moore en 1965 qui prévoit l'augmentation de la densité des transistors au sein des circuits électroniques (le nombre de transistors doublera tous les dix-huit mois par circuit de même taille), le développement des technologies des circuits intégrés a permis l'intégration de millions de transistors sur une même puce. Cette croissance d'intégration a donc poussé les concepteurs de circuits intégrés à s'orienter vers la réutilisation de composants matériels déjà conçu appelés IPs (*Intellectual Property*) qui, comme des cellules de bibliothèque peuvent être utilisés comme éléments de base pour concevoir de nouveaux circuits afin d'accélérer le développement et ainsi réduire le temps de mise sur le marché (time-to-market).

En effet, le canal du transistor est devenu de plus en plus court. Ceci permet de placer plusieurs composants sur la même puce pour concevoir un système sur puce (SoC-System on chip). Aussi, la tension de seuil du transistor est tellement faible que le délai des portes logiques a été considérablement réduit. Malheureusement, le délai de transfert de données sur les interconnexions se pose encore. Du fait que les systèmes actuels se caractérisent par de fortes communications, les architectures à base de bus sont devenues inefficaces.

Un nouveau paradigme d'interconnexion est alors apparu (le réseau sur puce - Network on Chip-) afin de surmonter les problèmes des méthodes d'interconnexion classiques. Ce nouveau paradigme est inspiré des réseaux informatiques en les ramenant au niveau des circuits intégrés. Un réseau sur puce permet de connecter plusieurs composants communiquant simultanément via des routeurs et de courts fils d'interconnexion.

Cependant, il n'existe pas d'outils permettant le développement d'un réseau sur puce. C'est pourquoi les outils d'aide à la conception de réseaux sur puce sont des allées de recherche très actives.

C’est donc dans cette optique que la question de recherche se pose : Comment aider les concepteurs de circuits intégrés à concevoir des réseaux sur puce à des coûts raisonnables et dans un temps limité tout en tirant parti des possibilités offertes par les nouvelles technologies ?

La conception d'un réseau sur puce faisant intervenir un nombre considérable de tâches et donc l'implication de toute une équipe (une dizaine de personnes) pour une réalisation en un temps raisonnable, notre travail s'est limité à quelques tâches.

Plus précisément, notre travail a consisté dans une première phase en une activité intense de recherche détaillée sur les réseaux sur puce (en particulier la tâche de mapping) et la proposition d’une solution permettant l'optimisation du nombre de liens verticaux et leur placement dans le contexte une architecture 3D (circuit intégré multicouche). L'autre partie de notre travail a visé l'optimisation du nombre de PLLs (Phase Locked Loop pour boucle à verrouillage de phase) pour générer les fréquences d'horloge requises pour le fonctionnement des différents composants du système sur puce.

Contributions:

Afin d’atteindre nos objectifs de recherche, nous avons entamé la démarche suivante:

Une recherche avancée sur les réseaux sur puce et la génération de topologies pour les réseaux sur puce.

Comme résultat de cette étude nous avons proposé un framework de classification et d’évaluation des stratégies de mapping dans les réseaux sur puce.

Une réduction et un placement efficace de liens verticaux sur une architecture 3D déjà générée.

Enfin, nous avons proposé une solution pour confronter le problème de génération d'horloges pour les différents composants fonctionnant a différentes fréquences d'horloges au sein du réseau sur puce.

Organisation de la thèse :

Chapitre 1 : Les réseaux sur puce. Dans ce chapitre nous présenterons les concepts de base des réseaux sur puce, leurs phases de conception, les réseaux sur puce existants et nous terminerons par les nouveaux challenges dédiés à la conception d'outils de conception assistée par ordinateur pour les réseaux sur puce.

Chapitre 2 : Le mapping dans les réseaux sur puce. Le deuxième chapitre sera consacré à la phase de mapping. Nous détaillerons la proposition du framework de classification et d'évaluation des stratégies de mapping dans les réseaux sur puce.

Chapitre 3 : Placement efficace de liens verticaux sur une architecture 3D. Ce chapitre introduira le concept d'une architecture 3D dans la conception des circuits et son impact sur les performances des réseaux sur puce. Nous parlerons de la technique pour le placement de liens verticaux afin de tirer un maximum de profit des opportunités qu'offre la troisième dimension dans la conception des réseaux sur puce.

Chapitre 4 : La génération d'horloges dans le réseau sur puce. Le problème de génération d'horloges sera traité au chapitre 4. Nous exposerons la démarche suivie et les solutions apportées pour résoudre ce problème.

Enfin, nous terminerons avec une analyse des résultats de ces travaux et présenterons les conclusions et les perspectives de ce travail de thèse.

CHAPITRE 1
LES RESEAUX SUR PUCE (*Networks on Chip –NoCs-*)

1.1 Introduction

Les structures d'interconnexion classiques utilisant le bus partagé rencontrent beaucoup de problèmes lorsque les systèmes se complexifient (*un nombre important de composants sur la même puce*). Particulièrement, le parallélisme, la limitation du débit, la consommation d'énergie et la synchronisation globale sont autant de problèmes dans la conception de tels systèmes.

La tendance actuelle dans la conception des systèmes sur puce est de remplacer les interconnexions classiques par les réseaux sur puce. Cette nouvelle technique d'interconnexion permet d'améliorer les performances du système. Quand le bus est utilisé comme moyen de communication, la bande passante est partagée par tous les composants du système [1]. Les NoCs sont préférables aux bus parce qu'ils offrent une plus grande bande passante. En plus, ils peuvent prendre en charge plusieurs communications simultanées [2].

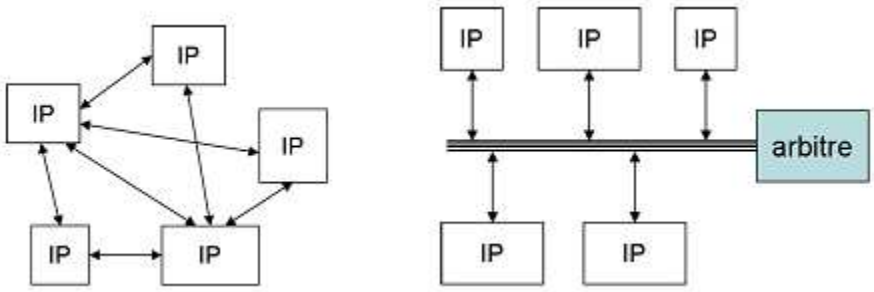
Plusieurs travaux de recherche dans la littérature sur les NoCs ont montré que ceux-ci rencontrent beaucoup de problèmes pendant leurs phases de conception. Citons le choix de la topologie du réseau, la limite de la surface, la consommation d'énergie, etc.

Le but de ce chapitre est d'introduire le paradigme des réseaux sur puce. A cet effet, nous allons décrire les différents types d'interconnexions utilisés dans les systèmes sur puce. Ensuite, nous définirons les concepts des réseaux sur puce. Enfin, nous présenterons quelques travaux existants et les nouveaux apports liés à la conception d'un NoC dans le cadre de la thèse.

1.2 Les différents types d'interconnexions

Le paragraphe précédent a introduit le réseau sur puce comme étant une nouvelle solution pour l'interconnexion des systèmes sur puce. Dans celui-ci, nous allons présenter la communication dans les systèmes sur puce.

1.2.1. La communication basée sur la topologie point à point (figure 1.1 (a)) Entre chaque paire d'IPs communiquant il existe une interconnexion dédiée uniquement à l'échange de données entre ces deux IPs [3]. Ce type d'interconnexion entre les IPs d'un système est simple de réalisation et permet des échanges de données très rapides, mais limite toute flexibilité et extensibilité lors de la conception de systèmes sur puce [4]. L'utilisation d'un nombre important de liens permet d'avoir la meilleure performance en termes de bande passante. Cependant, le taux d'utilisation des liens est très bas (*environ 10% selon une recherche de Dally et al [2]*). Cette topologie de communication peut être utilisée dans les systèmes qui comportent peu de composants communicants et nécessite des débits de transfert de données très élevés.



(a) Point-à-point (b) Bus partagé

Figure 1.1 : Architectures d'interconnexion traditionnelles dans les SoCs

1.2.2. La communication par un bus partagé ou hiérarchique : Elle propose un moyen de communication partagé par tous les IPs d'un système. Un bus est composé de lignes de données, de contrôle et d'une unité d'arbitrage. Les lignes de données transportent les informations de la source vers la destination. Les lignes de contrôle portent les informations sur les signaux de contrôle (*signal d'acquittement*). L'unité d'arbitrage gère la mise en place d'une connexion entre une source et une cible. Avec ce type de structure de communications, la

communication ne peut se faire qu'à l'alternat. A un instant donné une seule opération de communication entre les composants connectés est possible.

Une structure de communication ayant une topologie de bus présente un goulot d'étranglement des données dès que le nombre de composants interconnectés augmente significativement [4] : De plus, le nombre de modules connectés à un bus doit être limité a cause des effets parasites (*effets capacitifs et résistifs*) et la consommation d'énergie associés principalement à la longueur importante des lignes et au nombre de modules connectés.

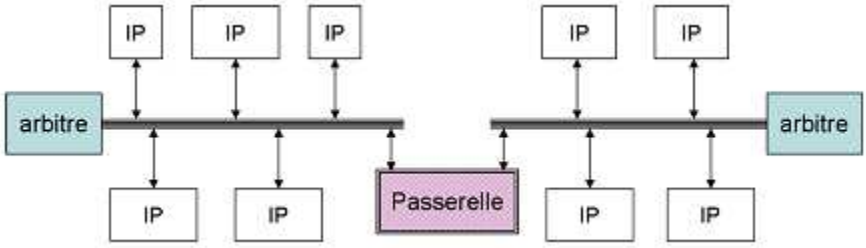


Figure 1.2 : Bus hiérarchique

L'augmentation du nombre de modules implique aussi l'augmentation de la taille du bus, et de ce fait augmente les délais de transmission des données [5]. Le bus est donc par nature limité dans sa capacité à interconnecter un grand nombre d'IPs. Une structure de bus hiérarchique (*figure 1.2*) a été proposée pour réduire le nombre de composants qui sont connectés sur le même bus et réduire ainsi la longueur des fils du bus partagé. Son principe est de relier plusieurs bus entre eux par l'intermédiaire de passerelle et donc chaque bus possède son propre arbitre. Cependant, cette solution ne peut éviter les problèmes naturels des bus partagés [6].

1.2.3. La communication par réseau sur puce : Les NoCs apparaissent comme une solution pour surmonter les problèmes de bus. Dans les NoCs, le transport des données d'une source vers une destination se fait à travers des routeurs et des canaux d'interconnexion. La figure 1.3 illustre une structure de NoC de type maillé 2D (*mesh*), composée de plusieurs éléments de calcul IPs ou PE (*Processing Element*) connectés les uns aux autres via des routeurs et des fils d'interconnexion. Un IP est connecté à un routeur via un module d'interface NI

(*Network Interface*) dont la fonction principale est la mise en paquets de toutes les données qui circulent sur le réseau (*le sous-paragraphe 1.3.1 détaille les rôles des composants d'un NoC*).

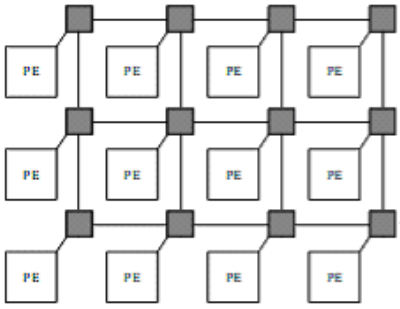


Figure 1.3: Exemple d'une structure de NoC de type mesh [4]

Le tableau suivant (*tableau 1.1 [4]*) résume les diverses caractéristiques des trois grands types de communication dans les systèmes sur puce.

Tableau 1.1: Comparaison des interconnexions [4]

	Point à point	**Bus**	**NoC**
Parallélisme	Très bon	Mauvais	Très bon
Scalabilité	Très mauvais	Bon	Très bon
Réutilisation	Très mauvais	Très bon	Très bon
Consommation	Bon	Mauvais	Très bon

Parmi les différents moyens de communication, le réseau sur puce est le meilleur. Il permet de prendre en charge plusieurs communications simultanément. Il est capable d'être étendu : on peut ajouter des IPs sans altérer les performances globales du système. En effet, dans les systèmes utilisant le bus, le nombre d'IPs connectés se partagent le même média de transmission et donc limitent la bande passante et l'extensibilité du système. Chaque IP est relié au routeur par une interface-réseau qui permet de connecter cet IP au réseau et de normaliser la communication. Enfin, les fils d'interconnexions sont plus courts dans les NoCs et donc consomment moins.

Dans ce qui suit, nous détaillons les concepts de base des réseaux sur puce.

1.3 Concepts des réseaux sur puce

1.3.1. Les composants d'un réseau sur puce

Un réseau sur puce est composé de routeurs, d'interfaces-réseaux et de liens de communication (figure 1.4 [7]). Les routeurs sont chargés d'aiguiller les données qu'ils reçoivent vers leurs destinations. Les interfaces-réseaux permettent aux IPs d'accéder au réseau sur puce à travers les routeurs. Enfin, les liens de communication assurent le transfert des données entre les routeurs.

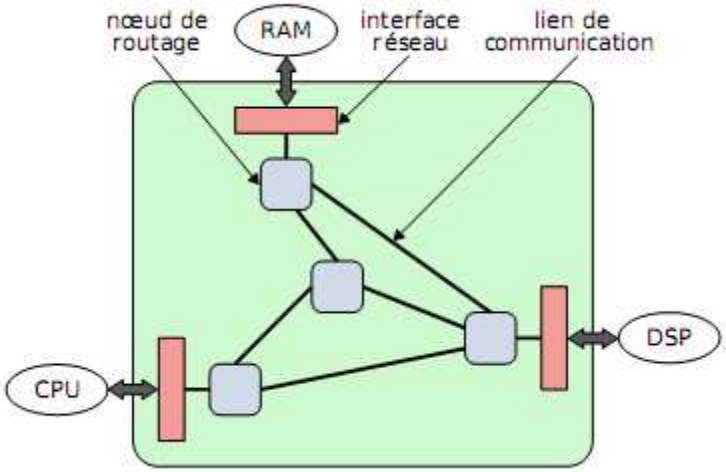


Figure 1.4: Structure d'un réseau sur puce [7]

1.3.1.1. Les interfaces-réseaux [7]:

Une interface réseau fournit un ensemble de services de communication (*paquétisation, contrôle de flux, ordonnancement des communications, …*) aux composants qui lui sont connectés. Elle se sert des primitives de communication de base du réseau sur puce pour répondre aux appels à ses services de communication. Comme l'illustre la figure 1.5, une interface réseau est divisée en deux parties :

- une partie frontale (*front end*) qui est reliée à un composant fonctionnel du circuit,
- une partie arrière (*back end*) qui est reliée au réseau sur puce.

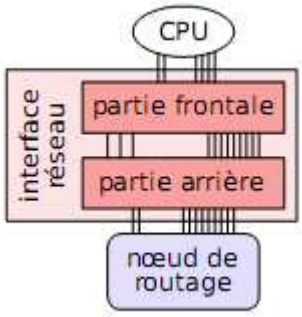


Figure 1.5 : Architecture d'une interface-réseau [7]

1.3.1.2. Le routeur :

Son rôle est d'acheminer les données à traves le réseau. Il est constitué des éléments suivants (figure 1.6 [7]) :

- **Les files d'attente** qui stockent les paquets circulant.
- **Le commutateur** connecte les files d'attente d'entrée aux files d'attente de sortie.
- **L'unité de routage et d'arbitrage** qui assure l'aiguillage des paquets et gère les conflits d'accès (*lorsque plusieurs files d'attente d'entrée doivent être connectées à la même file de sortie*).

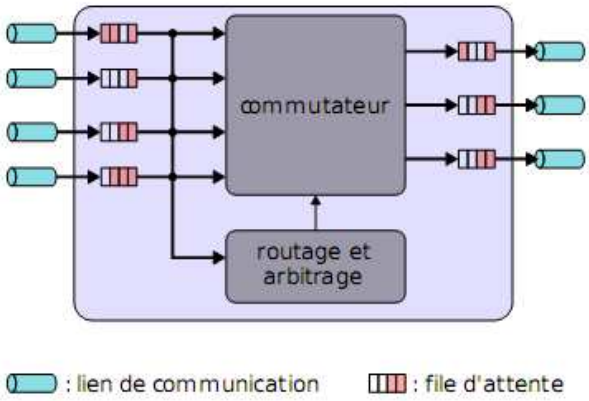


Figure 1.6 : Structure d'un nœud de routage [7]

1.3.1.3. Les liens de communication

Leur rôle est de transférer des données entre deux composants d'un réseau sur puce. Les fils utilisés pour réaliser les liens de communication d'un réseau sur puce sont souvent sensibles aux phénomènes du bruit [7] : une solution permettant de résoudre ce problème est d'utiliser des codes correcteurs et/ ou

détecteurs d'erreurs. Des répéteurs ou des FIFOs (*First In First Out*) peuvent être ajoutés pour segmenter les fils ayant de longues distances à parcourir.

La topologie d'un réseau sur puce définit comment sont interconnectés ces différents composants. Comme pour les réseaux informatiques, de nombreuses topologies sont envisageables pour construire un NoC. Elle spécifie l'organisation du réseau et elle est souvent modélisée par un graphe où les nœuds de routage et les liens de communication sont respectivement associés aux sommets et aux arrêtes de ce graphe. Le paragraphe suivant décrit les topologies des réseaux sur puce.

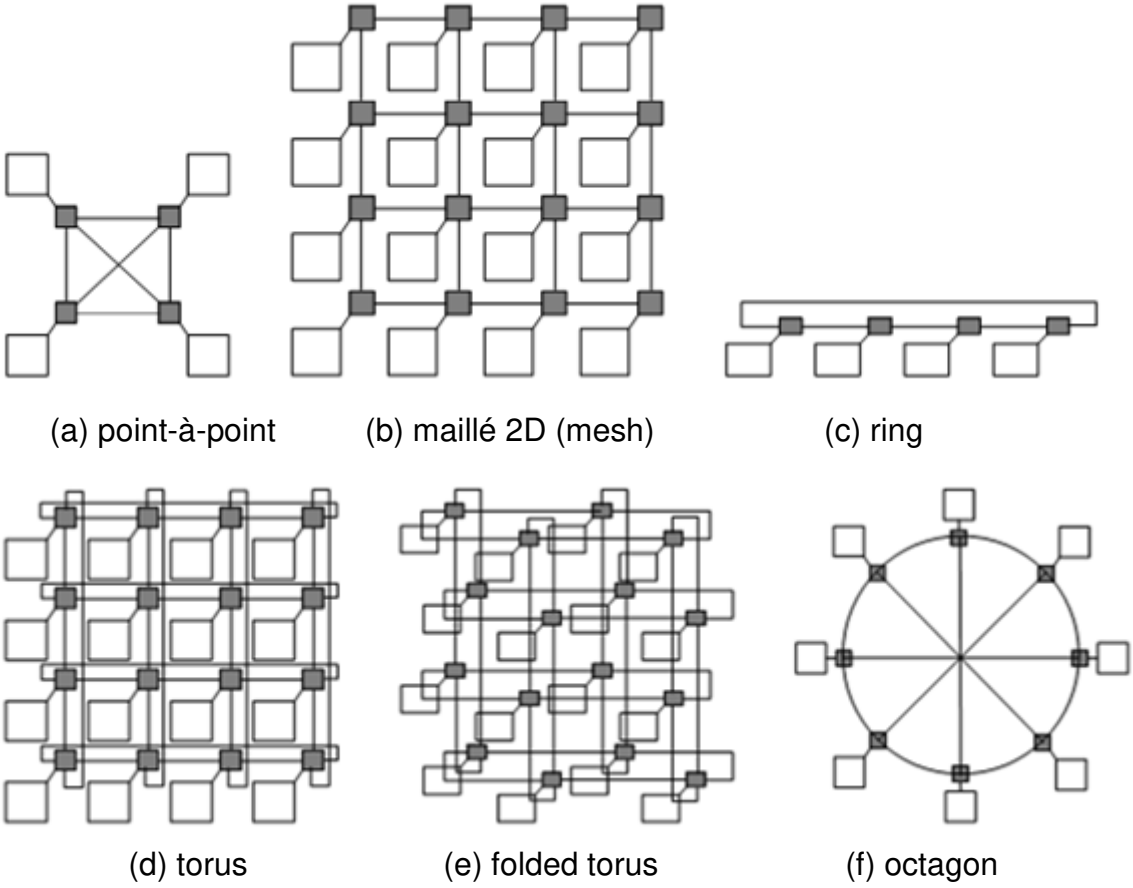
1.3.2 Topologies

Les réseaux sur puce peuvent être classés en 3 grands groupes [8] : les réseaux directs, indirects et irréguliers.

Les réseaux directs : Dans les réseaux directs (figure 1.7), chaque routeur est connecté à un IP et aux autres routeurs voisins. Les réseaux sur puce directs les plus utilisés sont les réseaux maillés 2D *mesh, torus, folded torus* et *octagon* [9, 2, 10, 11]. La topologie matricielle *mesh* semble être la topologie la plus répandue dans le domaine des réseaux sur puce. Ceci est dû à la même longueur de tous les canaux dans le réseau et à l'extensibilité de sa structure qui croît linéairement avec le nombre de nœuds connectés [4].

Les réseaux indirects : Pour les réseaux sur puce de type indirect (figure 1.8), chaque nœud est connecté à un routeur qui est connecté à d'autres routeurs par des liaisons points à point. Dans la topologie de type *tree*, structure de type « arbre », les éléments de calcul sont connectés uniquement aux extrémités de la structure topologique que l'on nomme « feuilles » de l'arbre. Ce type de structure présente des gros inconvénients en terme d'embouteillage de trafic des paquets de données [4]. En effet, souvent le routeur *root* est surchargé de trafic et limite donc fortement les performances du réseau. La topologie de type *fat tree* permet de réduire les encombrements et les embouteillages qui peuvent se produire dans ce type de réseau, où le nombre des canaux entre les routeurs adjacents augmente en se rapprochant du routeur *root*. Une variation de topologie "*fat tree*"

est la structure papillon (*butterfly*). Ces solutions permettent d'améliorer le débit de ces réseaux tout en restant moins coûteuses en terme de connectivité [4].

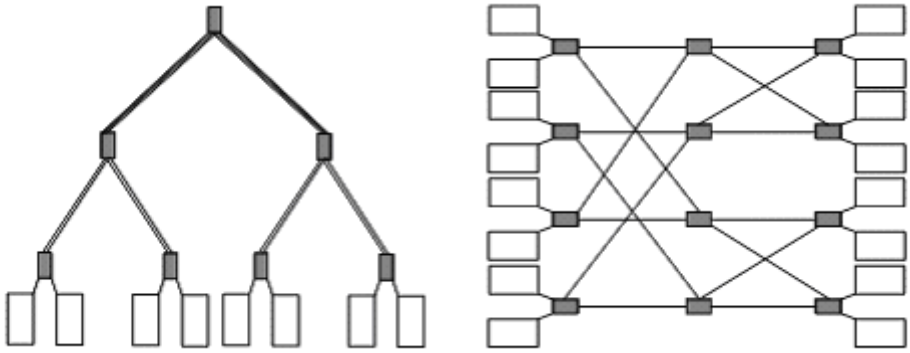


(a) point-à-point (b) maillé 2D (mesh) (c) ring

(d) torus (e) folded torus (f) octagon

Figure 1.7 : Exemples de topologies des réseaux sur puce de type direct.

De nombreux autres exemples de NoC indirects peuvent être trouvés dans la littérature [12, 1, 13].



(a) fat tree (b) butterfly

Figure 1.8 : Exemples de topologies des réseaux sur puce de type indirect.

Les réseaux irréguliers : Le troisième type de réseau sur puce est de topologie irrégulière. Ce sont des structures d'interconnexion qui combinent généralement les topologies de type direct et indirect vues précédemment. Un exemple d'une topologie irrégulière est la topologie de type *mesh* où les routeurs non utilisés sont simplement supprimés du réseau. Parmi ce type de topologie de réseau sur puce irrégulière, nous pouvons citer les réseaux Xpipes [14]. Ce genre de topologie est souvent associé à des applications spécifiques.

Après avoir présenté les différentes topologies, nous allons présenter, dans le paragraphe suivant, les algorithmes de routage déterminant le cheminement employé pour la transmission des messages entre les nœuds d'une topologie d'un réseau sur puce.

1.3.3 Algorithme de routage

Le routage permet de déterminer le chemin qui sera employé pour acheminer les paquets entre des nœuds source aux nœuds destination. Plusieurs approches pour le routage on été décrites dans la littérature et peuvent être classés selon leurs techniques de routage. On cite les catégories telles que le routage statique ou dynamique, distribué ou source, minimal ou non-minimal, tolérant aux fautes ou non.

Les algorithmes de routage statique sont des algorithmes définissant des chemins fixes et identiques pour transférer les paquets de données d'un nœud source vers un nœud destinataire. Cette méthode de routage ne tient pas en compte les conditions du réseau telles que sa charge actuelle, l'occupation des canaux de transmission, etc. Des exemples d'algorithmes de routage statiques sont les algorithmes de routage *XY* (figure 1.9 [15]) et west-first. Par contre, dans les algorithmes de routage dynamique, les décisions d'acheminement des paquets son prises au cours du temps en fonction des conditions du réseau. Les algorithmes *odd-even* [16] et *look-ahead* [17] sont deux exemples d'algorithmes de routage dynamique.

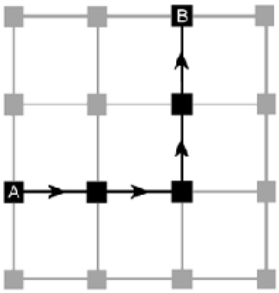


Figure 1.9 : L'algorithme de routage XY[15]

Les algorithmes de routage statique et dynamique (*appelés aussi déterministe ou adaptatif*) peuvent être également classés selon la façon dont les informations de routage sont stockées et l'endroit où les décisions de routage sont prises (*au niveau routeur local ou routeur source*). On distingue les algorithmes de routage source et distribué : pour les algorithmes de routage source, des informations de routage (*appelées feuille de route*) sont additionnées dans le paquet à transmettre et les autres routeurs consultent ces informations pour acheminer le paquet. Dans le type distribué, les décisions de routage sont prises par chaque routeur qui dispose à son niveau d'une table de routage ou d'une fonction de routage qui lui permet de calculer la destination du paquet.

Les algorithmes de routage peuvent être aussi classifiés selon la distance parcourue par un paquet : les algorithmes de routage minimaux et non-minimaux. Pour un algorithme minimal, le chemin de routage d'un paquet entre deux nœuds est le chemin le plus court (*l'algorithme de routage XY*). Par contre, dans les algorithmes de routage non-minimaux un paquet peut suivre n'importe quel chemin libre sans contrainte de distance.

Comme une dernière classification, les algorithmes de routage peuvent être considérés comme tolérants aux fautes ou non. On dit qu'un algorithme est tolérant aux fautes lorsque malgré la défaillance d'un des routeurs du réseau, l'acheminement des paquets peut être effectué. Plusieurs exemples d'algorithmes de routage tolérants aux fautes peuvent être trouvés dans la littérature [18, 19, 20].

Un autre point important dans le routage des paquets est l'identification des nœuds, ce qu'on appelle adressage dans les réseaux informatiques. Dans les réseaux sur puce, il est assez courant d'utiliser un adressage assigné en fonction du nombre d'éléments interconnectés ou alors en fonction de la position relative de ces éléments dans le réseau [21].

1.3.4 Les paramètres des réseaux sur puce

Pour évaluer les caractéristiques d'un réseau sur puce, de nombreux critères peuvent être pris en compte. On souhaite un réseau sur puce proposant des débits de transferts élevés, avec une faible latence, en minimisant la consommation d'énergie, le tout occupant une faible surface de silicium. En plus, ce réseau doit assurer la diversité de chemin, l'extensibilité, la flexibilité, etc. Dans ce qui suit, nous résumons ces caractéristiques.

a. Débit de données : C'est la quantité de données transitant dans le réseau par unité de temps. Le débit dépend des caractéristiques physiques des liens (*largeur des liens)* et prend en compte aussi les temps de routage et de bufférisation [22].

b. La latence : elle correspond aux délais (*généralement exprimés en nombre de cycles*) nécessaires à l'acheminement d'un message depuis un nœud émetteur jusqu'au nœud destinataire. Elle dépend de la longueur du message, des délais de routage et de la distance entre la source et la destination [22].

c. La surface : dans un réseau sur puce, elle représente principalement l'aire occupée par les composants du réseau, c'est-à-dire les routeurs (*buffers*), les liens de communications (*longueur et largeur*) et les blocs IPs insérés dans le réseau.

d. La consommation : l'énergie consommer dans un réseau sur puce est la somme d'énergie consommer par l'ensemble des composants du réseau, donc, elle est relative au nombre de ressources du réseau. Une optimisation de la consommation d'un réseau sur puce passe par une optimisation des ressources du réseau (*tailles des buffers, nombre de routeurs, …)* [22].

e. La diversité de chemin : c'est l'existence de plusieurs chemins entre deux nœuds du réseau. Cette caractéristique permet au réseau d'être tolérant aux fautes [21].

f. L'extensibilité : Elle représente la possibilité d'étendre le réseau en nombre de composants communiquant sans altérer les performances générales du système. En particulier, la bande passante doit augmenter proportionnellement lorsque de nouveaux éléments de traitement sont rajoutés au réseau [21].

g. La flexibilité : il s'agit à sa capacité de réutilisation dans de futurs systèmes [23].

1.4 La conception des réseaux sur puce

Un NoC est principalement caractérisé par son architecture définie par sa topologie et sa fonction de routage permettant la gestion des communications. Il existe plusieurs propositions de NoC dans la littérature dont quelques unes sont citées ci-dessous.

1.4.1. Exemples de réseaux sur puce

Le réseau SPIN (*Scalable Programmable Integrated Network) :* est un réseau sur puce développé par le laboratoire LIP6 (Laboratoire d'Informatique de Paris 6) de l'université Pierre et Marie Curie en 2000. Il dispose d'une topologie en arbre élargi (*fat tree)* avec un algorithme de routage adaptatif et distribué.

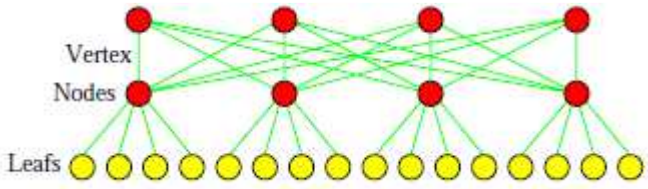
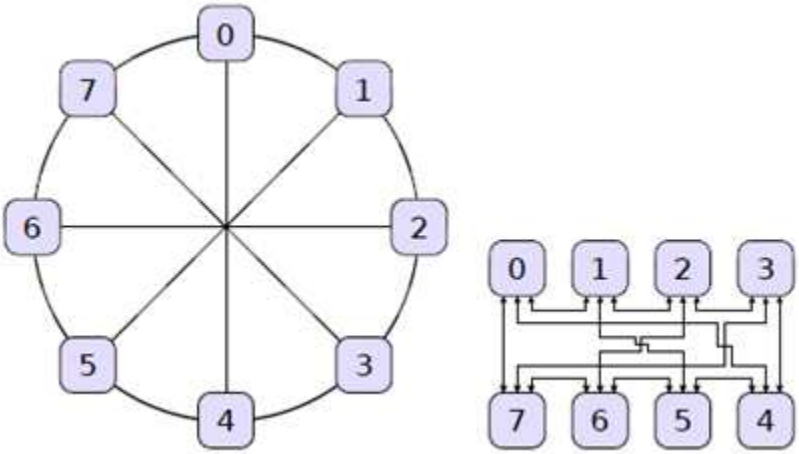


Figure 1.10: topologie en arbre élargi [13]

Le réseau SPIN a été réalisé en utilisant la technologie CMOS 0.13µm de STMicroelectronics. Le routeur RSPIN occupe une surface de 0.24 mm^2 et un réseau SPIN32 correspondant à un réseau de 16 routeurs avec 32 ports de connexion aux ressources nécessite 4.6 mm^2 de silicium, dont les FIFOs occupent 30% de la surface [13].

Le réseau OCTAGON : il a été proposé par STMicroelectronics et l'Université de Californie à San Diego en 2001 [11]. La topologie du réseau est un anneau octogonal avec des liens supplémentaires entre routeurs diamétralement opposés

(figure 1.11 : La topologie en octogone est constituée de huit nœuds et douze liens physiques de connexion. L'envoi d'un message nécessite deux sauts au maximum). L'algorithme de routage est distribué et adaptatif.



Vue structurelle **Vue physique**

Figure 1.11: Topologie du réseau sur puce Octagon [7]

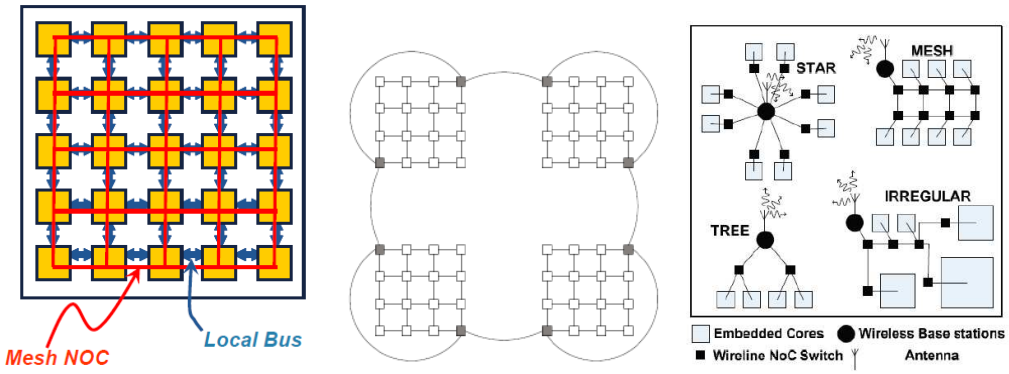
Le réseau Mango : a été proposé par l'Université Technique du Danemark [24] utilise un algorithme de routage de type source, déterministe. La topologie du réseau est une maille à deux dimensions.

Le réseau ÆTHEREAL [25] : il a été proposé par le laboratoire de recherche Philips Research repose sur une topologie de grille 2D et son algorithme de routage est déterministe et de type source.

On constate que la topologie 2D maillée est la plus utilisée dans les réseaux sur puce (Mongo [24], ÆTHEREAL [25], aSOC [26], Hermes [27], QNOC[28] et autres). Ceci est dû à la facilité de son implémentation et son indépendance de la taille du réseau. En plus, le routage dans la topologie 2D maillée est simple [29].

Des architectures hybrides combinant deux architectures au sein d'un même réseau sont aussi proposées [30]. On peut avoir un réseau qui combine l'architecture de grille avec l'architecture de bus : le bus est utilisé pour faire communiquer les nœuds voisins directement (*sans passer par un routeur*) et la

topologie de grille est utilisée pour transmettre les paquets aux autres nœuds (figure 1.12 (a)). Une autre architecture (figure 1.12 (b)) combine la topologie de grille (*utilisée pour le réseau local*) avec la topologie anneau (*pour le réseau global*). La figure 1.12 (c) montre une architecture de réseau sans fil.



(a) (b) (c)

Figure 1.12: Architecture Hybride [30]

La conception d'un réseau sur puce nécessite plusieurs niveaux d'abstraction partant de la modélisation haut-niveau de l'application jusqu'à son implémentation physique. Le paragraphe suivant d'écrit les phases de conception d'un réseau sur puce.

1.4.2. Les phases de conception d'un réseau sur puce

Les composants (IPs) exécutent une application qui est spécifiée comme un ensemble de tâches avec la communication entre eux. Ceci est représenté par le graphe de tâche d'application. À partir d'un graphe de tâches, de nombreux modèles de NoC traditionnels appliquent généralement les étapes suivantes:

- l'ordonnancement et l'assignation des tâches aux IPs;
- le mapping de ces IPs dans le réseau;
- définir la stratégie de routage; et
- simuler la conception physique du réseau.

L'ordonnancement vise à assigner les tâches aux IPs pour être exécutées à un moment donné. Plusieurs critères peuvent être considérés dans cette phase: le temps total d'exécution, le temps moyen passé par une tâche dans le système,

l'énergie consommée par une tâche dans un IP, l'attribution de certaines tâches au même IP et l'architecture du système (*homogène ou hétérogène*). Le mapping consiste à placer les composants IPs du graphe d'application sur l'architecture cible (*aux ressources ou tuiles*) tout en minimisant les coûts de communication, la consommation d'énergie ou la latence du système. La phase de routage prend en considération aussi le coût de communication : par le choix de l'algorithme de routage, la conception de routeur spécifique (*définir la taille des buffers, etc*), traitant le problème d'interblocage (*l'interblocage se produit lorsque certains paquets sont mutuellement en attente parce qu'ils réservent quelques ressources et attendent les uns les autres pour libérer les ressources*), etc. Enfin, la phase de conception physique (*floorplanning*) consiste à définir le placement physique (*position*) des différents composants du circuit.

Avec la miniaturisation continuelle de la technologie des semi-conducteurs et la complexité de conception de grands circuits, les concepteurs de circuits électroniques sont ramenés à utiliser des outils informatiques adéquats. Cependant, il n'existe pas d'outils d'automatisation de l'ensemble des phases de conception d'un réseau sur puce. C'est pourquoi les outils d'aide à la conception de réseaux sur puce sont des avenues de recherche très actives.

1.4.3. Les nouveaux challenges liés à la conception d'un NoC

La recherche en C.A.O. (*Conception Assistée par Ordinateur*) permet de proposer de nouvelles solutions aux concepteurs de circuits afin de développer des circuits à des coûts raisonnables et dans un temps limité tout en tirant parti des possibilités offertes par les nouvelles technologies.

Dans ce contexte, la division Microélectronique et Nanotechnologie du centre de développement des technologies avancées propose le développement d'outils CAO-VLSI pour la conception de réseaux sur puce. La technique en cours d'investigation concerne la conception d'un réseau sur puce pour des applications spécifiques.

A partir des travaux proposés dans [31,32], nous avons, dans le cadre de cette thèse, développé d'autres outils d'aide à la conception de réseaux sur puce:
- placement efficace (surface, temps, consommation de puissance) de liens verticaux à partir d'un placement d'IPs sur une architecture 3D élaboré dans [33, 34],
- optimisation efficace de PLLs (les PLLs sont des diapositives électronique permettent la génération des fréquences d'horloge des composants du système) pour la réduction de la surface et de la consommation de puissance dans un NoC.

1.5 Conclusion

Dans ce premier chapitre, nous avons introduit le concept des réseaux sur puce ainsi que les notions associées. En effet, partant du contexte dans lequel les réseaux sur puce ont émergé, nous avons décrit les caractéristiques de ces réseaux. Nous avons cité quelques exemples de réseaux sur puce existants suivis des phases de conception. Afin de permettre d'automatiser ces phases, de nouvelles techniques et méthodologies de conception sont proposées dans plusieurs travaux de recherche.

Des problèmes peuvent surgir lors de la conception d'un réseau sur puce du fait du nombre de paramètres qui sont à prendre en compte, car chacun d'entre eux peut influencer les performances globales du système. Le choix des paramètres d'un NoC est un problème très complexe. Citons le coût en consommation, le coût en surface, le choix de la topologie du réseau, etc. Trouver une combinaison optimale de ces paramètres afin de maximiser les performances d'un NoC est un problème d'optimisation combinatoire NP-difficile. Les outils que nous avons développés traitent pour la plupart des problèmes d'optimisation combinatoire. Ceux-ci n'étant pas de complexité algorithmique polynomiale, il devient obligatoire de développer des méthodes à base d'heuristique ou de méta-heuristiques. Dans le chapitre suivant nous exposons les méthodes d'optimisation proposées pour résoudre le problème de mapping dans les réseaux sur puce.

CHAPITRE 2
LE MAPPING DANS LES RESEAUX SUR PUCE

2.1Introduction

La conception d'un réseau sur puce efficace qui satisfait les contraintes de performance de l'application est un processus complexe. Il passe par plusieurs niveaux d'abstraction, partant de la modélisation de l'application jusqu'à son implémentation physique (*voir sous-paragraphe 1.4.2 du chapitre 1*). La phase de mapping est une phase centrale dans la conception des NoCs dont le résultat a un impact direct sur les performances du système. Ce chapitre sera consacré à l'étude des principes techniques de mapping existantes où nous proposons un framework d'évaluation et de comparaison des différentes techniques de mapping.

2.2 Le problème de mapping

Le problème de mapping est abordé dans la littérature sous plusieurs formalismes. La définition la plus adopté est la suivante :

Le mapping consiste à placer chaque élément de l'application sur l'architecture du réseau sur puce de sorte à maximiser les performances du système. Le problème de mapping est un problème d'optimisation combinatoire NP-difficile. L'espace de recherche croit avec la taille du système. Théoriquement, placer N IPs dans M nœuds du réseau ($N \leq M$) implique $M!/(M-N)!$ arrangements d'IPs possibles dans les nœuds du NoC. La solution optimale d'un problème d'optimisation peut rarement être déterminée en un temps polynomial. Il est nécessaire de développer une heuristique ou méta-heuristique afin de déterminer une solution optimale ou pré-optimale dans un temps CPU raisonnable.

La modélisation de l'application sous forme d'IPs exécutant des tâches et la topologie du NoC sous forme de ressources est réalisée sous forme de graphes orientés. Le Mapping consiste à placer les composants du graphe d'application (IPs) sur chaque sommet du graphe représentant la topologie (Tuile) tout en

minimisant les chemins de communications entre IPs. Les deux graphes utilisés sont les suivants :

-Un graphe d'application G = (V, A) où chaque vertex (nœud) $v_i \in V$ représente l'IP sélectionnée, et chaque arc directionnel $a_{ij} \in A$ représente une communication du vertex v_i au vertex v_j.

-Un graphe de topologie A= (T, L) où chaque vertex $t_i \in T$ représente une tuile de la topologie, et chaque arc directionnel $l_{i,j} \in L$ représente le chemin de routage entre t_i et t_j.

En utilisant les deux graphes, le problème de mapping consiste à affecter chaque IP de l'application à une et une seule tuile de l'architecture de sorte à minimiser le coût de communications de l'application. De ce fait, les IPs qui communiquent le plus entre elles doivent être placées sur des tuiles proches.

Pour faire face aux nouveaux défis, d'autres flot de conception d'un NoC ont vu le jour. Différentes approches de mapping ont été proposées dans la littérature :

- Une approche qui suppose que les IPs sont déjà placés dans le NoC. Dans ce cas, le mapping consiste uniquement à joindre une ou plusieurs tâches de l'application aux tuiles de l'architecture ou IPs. Cette caractéristique permet la réalisation de mapping dynamique, c'est-à-dire, au moment de l'exécution de l'application, les tâches sont mappées aux IPs.

-Une approche qui intègre le mapping avec l'ordonnancement, le routage ou le floorplanning et considère plusieurs critères en même temps (optimise les mêmes performances).

-Une autre approche consiste à personnaliser la topologie pour une application spécifique (ASIC-Application Specific Integrated Circuit). Cette approche vise à générer une topologie sur mesure de type irrégulière pour une application spécifique.

Pour comparer et évaluer les différentes approches de mapping existantes, nous proposons un framework de caractérisation permettant la classification et l'évaluation des stratégies de mapping dans les réseaux sur puce. Cette classification permet aux futurs chercheurs de bien comprendre le problème de placement, ses différentes approches et les guident vers une approche spécifique selon leurs objectifs visés [35].

2.3 Un Framework pour l'évaluation et la classification des stratégies de mapping [35]

La figure 2.1 présente les critères de classification des différentes techniques de mapping :

-Les méthodes développées : heuristiques ou métaheuristiques et l'objectif visé (mono ou multi-objectif).

-L'approche suivie : dynamique ou statique, durant l'exécution de l'application ou pendant la conception.

-Les phases combinées avec le mapping : ordonnancement, routage ou floorplanning.

-La topologie envisagée : régulière ou personnalisée.

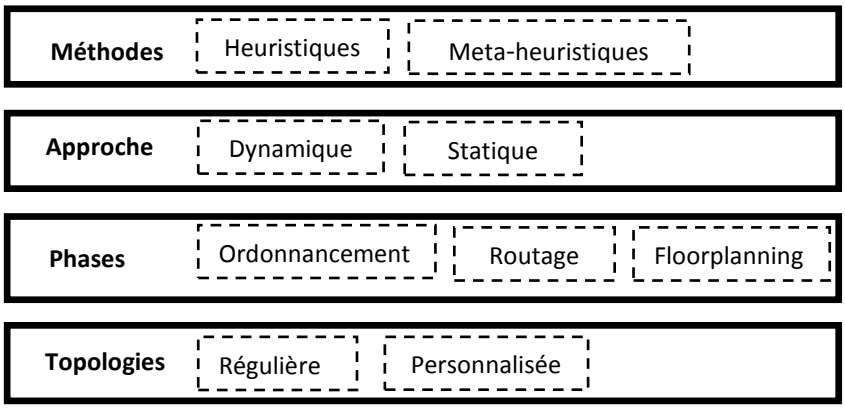


Figure 2.1 : Les critères de classification des stratégies de mapping

Ces quatre critères seront détaillés dans les paragraphes suivants. Un tableau récapitulatif sera ensuite donné en 2.4.

2.3.1 Métaheuristiques et heuristiques

On distingue les métaheuristiques des heuristiques (Figure 2.2) : les métaheuristiques sont basées sur des idées générales inspirées par des systèmes naturels et les heuristiques sont des stratégies spécifiquement développées pour résoudre un problème particulier. Les métaheuristiques et les heuristiques sont classifiées comme des méthodes constructives et transformatrices [36] : L'heuristique transformatrice transforme certaines solutions de mapping existantes afin de les améliorer. Dans les heuristiques constructives, des solutions partielles sont générées successivement, et à la fin la solution de mapping complète est obtenue. L'heuristique constructive peut être constructive sans amélioration itérative ou constructive avec amélioration itérative. Une heuristique constructive sans amélioration mappe les IPs sélectionnés (un à la fois) d'un graphe d'application sur le graphe de l'architecture du NoC. Il n'y aura pas de changement de position d'un IP une fois le placement effectué. Dans une heuristique constructive avec amélioration itérative, les IPs du graphe de d'application sont mappés sur le graphe de l'architecture ; cela génèrera une solution initiale. Ensuite, une amélioration itérative est effectuée sur la solution construite précédemment pour trouver de meilleures solutions candidates. Le tableau 2.1 montre des exemples de chaque classe.

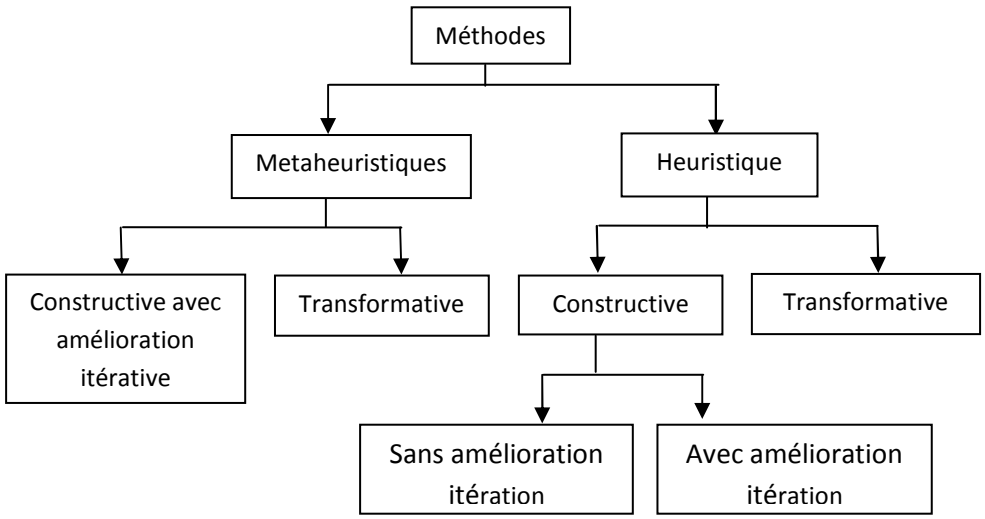


Figure 2.2: Classification selon la méthode choisie: Heuristiques et Metaheuristiques

Tableau 2.1: Exemples d'heuristiques et méta-heuristiques de chaque classe

<table>
<tr><th colspan="2">Métaheuristique</th><th colspan="2">Heuristique</th><th></th></tr>
<tr><td>Constructive avec amélioration itérative</td><td>Transformative</td><td colspan="2">Constructive</td><td>Transformative</td></tr>
<tr><td rowspan="2">-SA [37, 38]</td><td rowspan="2">-GA[39, 40, 41]
-SPEA [42, 43, 44]
-NSGA [45]
-PSO [46, 47]</td><td>Sans amélioration itérative</td><td>Avec amélioration itérative</td><td rowspan="2">-CGMAP [55]
-GAMR [56]</td></tr>
<tr><td>-CMAP [48]
-CHMAP [49]
-Spiral [50]
-Crinkle [54]</td><td>-Onyx [51]
-NMAP[52]
-MOCA [53]</td></tr>
</table>

3.3.1.1. Métaheuristiques

L'algorithme du recuit simulé a été proposé dans [36], où il a été comparé à l'algorithme de *branch- and-bound*. Leur but était de minimiser le coût des communications dans une architecture régulière 2D maillée (Figure 2.3). Les mêmes auteurs proposent dans [57] un algorithme de *branch- and-bound* afin de résoudre les problèmes de mapping et de routage pour une architecture régulière 2D maillée en minimisant le coût des communications.

La figure 2.3 présente une architecture régulière à base de tuiles où on peut placer les IPs d'une application. A partir du graphe de communications entre les tâches (chaque sommet représente un IP et chaque arc représente une communication entre deux IPs), le mapping détermine sur quelle tuile (par exemple (3,0), (2,1), (1,3), etc.) doit être placée chaque IP (par exemple, un circuit intégré propre à une application (ASIC), DSP3, CPU1, etc.) en minimisant le coût de la communication.

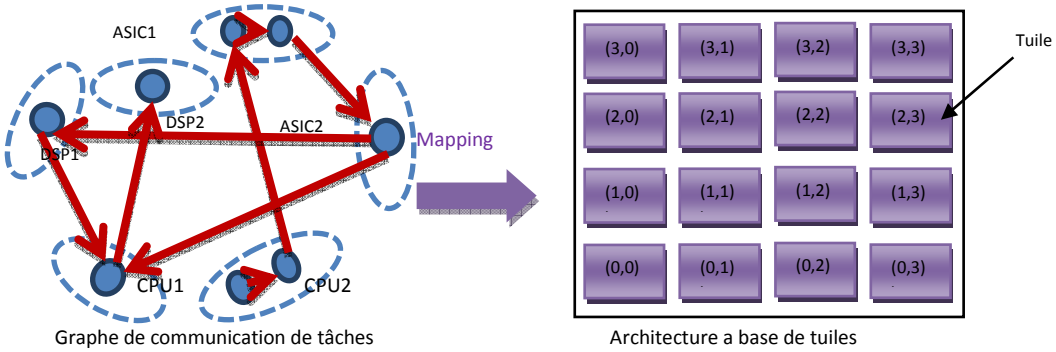


Figure 2.3 : Architecture à base de tuiles et le problème de mapping [37]

Afin d'accélérer l'algorithme du recuit simulé pour le mapping des IPs sur les tuiles, les auteurs dans [38] proposent de le combiner avec la technique de clusters (CSA : Cluster-based Simulated Annealing). Les IPs qui communiquent sont regroupés ensemble dans des clusters (groupes). Pour chaque cluster, une région du réseau sur puce est allouée.

Inspiré par les techniques évolutives, plusieurs travaux ont proposé un algorithme génétique. Ceci, pour résoudre le problème de mapping et d'ordonnancement [39] dans une topologie régulière 2D maillée en minimisant le temps total d'exécution. Dans [42**,** 43**,** 44] l'algorithme génétique SPEA (Strength Pareto Evolutionary Algorithm) permet d'effectuer une exploration multi-objectif (puissance et performance) de l'espace de mapping de réseaux sur puce basé sur une topologie 2D maillée. La méthode SPEA maintient un archive externe contenant les meilleurs fronts de compromis rencontrés durant la recherche. Une approche utilisant l'algorithme génétique multi-objective (NSGA : *Non Dominated Sorting Genetic Algorithm)* est développée dans [45]. L'optimisation multi-objective ne fournit pas seulement une solution unique (un mapping), mais plutôt un ensemble de solutions (des mappings). Chaque solution est un mapping au sens de Pareto qui donne un compromis entre les objectifs à optimiser (puissance et performance).

Dans [46], un algorithme d'optimisation multi-objectif évolutionnaire est proposé (Particle Swarm Optimization-PSO) en hybridation avec l'algorithme de Dijkstra (déterminer le plus court chemin). Le mapping est réalisé sur une topologie 2D maillée. Une comparaison (Tableau 2.2) entre l'algorithme PSO et l'algorithme génétique (AG) montre que les résultats donnés par AG sont meilleurs que ceux donnés par PSO.

Tableau 2.2: Comparaison entre PSO et GA [46]

Nombre d'IPs	Nombre de Tâches	Temps d'exécution		Energie	
		PSO	AG	PSO	AG
2	4	348	164.61	14053	8560
4	6	3152	1718	24025	20619
6	8	13659	75985	302430	29693

Les auteurs en [40] confirment dans leur étude que l'algorithme génétique donne les meilleurs résultats. Ils ont testé plusieurs versions de l'algorithme génétique (en appliquant différentes méthodes pour les opérateurs de croisement et de mutation), avec l'algorithme de recuit simulé, l'algorithme PSO et d'autres. Les tests sont évalués sur une topologie 2D maillée. Par contre dans [47], l'algorithme PSO est prouvé meilleur que l'heuristique proposée dans [52] ainsi que l'algorithme génétique proposé par [41] (PSO réduit environ 50% de la consommation d'énergie par rapport à [52] et environ 10% par rapport à [41]).

Alors que les méta-heuristiques sont des idées algorithmiques générales qui peuvent être appliquées à une large gamme de problèmes, les heuristiques exploitent les informations dépendantes du problème pour trouver une solution à un problème spécifique. Dans la section suivante, certaines heuristiques appliquées au problème de mapping sont citées.

2.3.1.2. Heuristiques

CHMAP (Chain-Mapping), un algorithme de mapping des IPs dans une architecture de réseau sur puce 2D maillée est proposé dans [49]. Le principe est de construire des chaînes d'IPs connectés afin d'introduire une méthode pour prioriser les IPs à mapper. Initialement, les chaînes sont construites à partir du graphe de communications (Figure 2.4 (a) et (b)). Ensuite, le nombre de chaînes est réduit en optimisant la bande passante totale (Figure 2.4 (c)). Les chaînes finales sont mappés sur la topologie en commençant par la chaîne de plus grande bande passante afin d'optimiser le mapping.

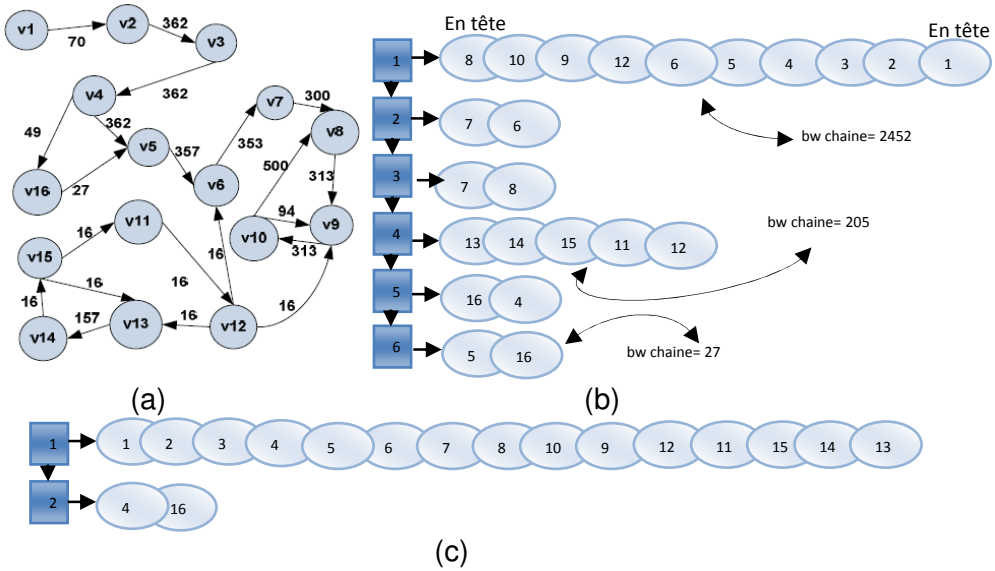


Figure 2.4 : (a) Graphe de communication VOPD (b) chaîne initiale et (c) chaîne finale [49]

La technique SPIRAL [50] consiste à placer de façon optimale les IPs d'une application donnée sur un réseau sur puce en topologie 2D maillée. Cette méthode affecte les IPs (ordonnées selon le degré communication de leurs tâches) de l'application aux ressources de l'architecture suivant la forme d'une spirale (Figure 2.5) en démarrant du centre pour atteindre les tuiles frontières.

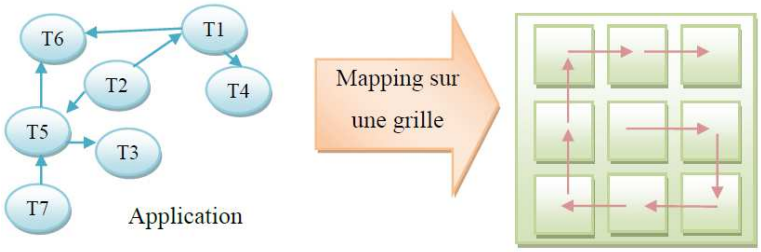


Figure 2.5 : Mapping avec la technique SPIRAL

Onyx [51] est une heuristique basée sur les priorités (des IPs ainsi que des tuiles) pour le mapping : La détermination de la priorité de chaque IP par la fonction *highest-priority* permet le placement des IPs qui communiquent le plus en premier. L'heuristique Onyx simule les différentes possibilités du placement d'un IP (Figure 2.6-a).

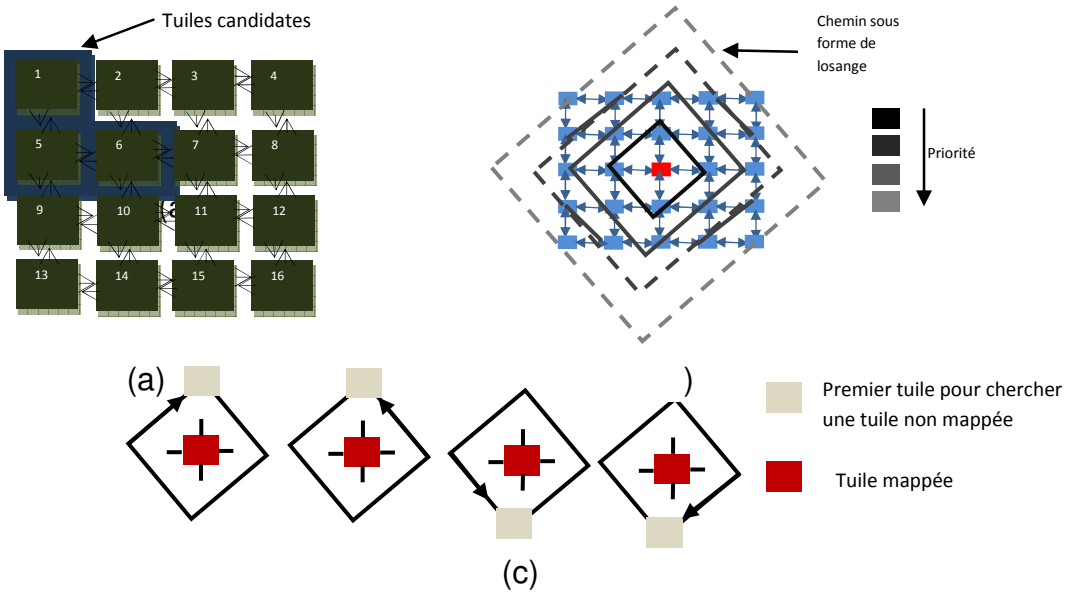


Figure 2.6 : (a) Tuiles candidates pour le placement du premier IP, (b) Concept de chemin sous forme losange, (c) Définition de 4 mouvements afin de se déplacer dans le chemin sous forme losange pour déterminer les priorités des tuiles pour le mappage [51].

L'IP qui communique le plus avec celui déjà placé, est mis dans une tuile libre la plus proche (avec une distance d'un saut, si non deux sauts, etc.). La recherche d'une tuile libre se fait dans un chemin sous forme losange (Figure 2.6-b) suivant 4 mouvements (Figure 2.6-c). Le type du mouvement donne des priorités différentes aux tuiles.

Crinkle [54] est une heuristique qui se base sur une liste de priorités et un placement des IPs en zigzage commençant par le coin de la topologie 2D (Figure 2.7).

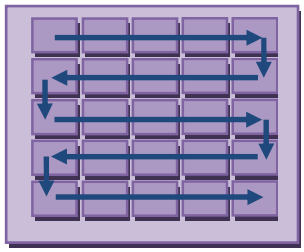


Figure 2.7 : chemin en zigzage pour le mapping des IPs [54]

Les travaux de cette section peuvent être aussi classés en deux classes : ***Statique*** ou ***Dynamique***. On détaille dans le paragraphe suivant ces types de mapping.

2.3.2 Mapping statique et dynamique

On distingue deux types de mapping statique et dynamique: Dans le mapping **statique,** tous les IPs sont affectés aux tuiles de l'architecture avant que l'application ne soit exécutée. Ce type de mapping est adapté à des plateformes spécifiques. A l'opposition, le mapping **dynamique** est effectué durant l'exécution de l'application. Une tâche de l'application peut être ajoutée, supprimée ou replacée. Le but est de reconfigurer la plateforme de communication tout en minimisant le temps de replacement des tâches.

OST et al [58], proposent plusieurs heuristiques pour un mapping dynamique mono-tâche et multi- tâches pour une architecture 2D maillée homogène. Ils donnent aussi une taxonomie pour le mapping des tâches. Ils classent le processus du mapping selon quatre critères : (1) le moment de son exécution, (2) le nombre de tâches par IP, (3) l'entité responsable du contrôle de mapping et (4) l'architecture cible.

Considérant le moment quand les tâches sont mappées, deux approches peuvent être utilisées :

- Au moment de la conception (statique) : les tâches sont affectées aux IPs avant l'exécution de l'application.

- Au moment de l'exécution (dynamique) : les tâches sont affectées aux IPs pendant l'exécution de l'application, et peuvent migrer (re-mappées) vers d'autres IPs pour améliorer les performances du système.

Considérant le nombre de tâches mappées par IP, deux approches peuvent être utilisées :

- Mono-tâche : dans cette approche, seulement une tâche est assignée pour chaque IP.

- Multi- tâches : dans cette approche, un groupe de tâches (cluster) est assigné à un IP.

Le mapping dynamique nécessite une entité (Mapper Actor), responsable du mapping des tâches au moment de l'exécution. Le contrôle de mapping peut être centralisé ou distribué:

- Centralisé : un IP est responsable de recevoir les requêtes de mapping, et d'envoyer la tâche à l'IP choisi.

- Distribué : le MPSoC est divisé en régions (clusters), et dans chaque région, un IP est responsable de l'exécution de mapping dans sa region.

Finalement, le mapping peut être classé selon l'architecture cible :

- Homogène : tous les IPs sont identiques.
- Hétérogène : différent IPs sont utilisés dans la même architecture.

Le tableau suivant résume les caractéristiques de quelques travaux qui offrent un mapping dynamique.

Tableau 2.3 : classification de mapping dynamique

Référence	Mono/multitâches	Homogène /hétérogène	Centralisé /distribué	Avec ou sans re-mapping (migration)
[58]	Multi- tâches	Homogène	Centralisé	Sans migration
[59]	Multi- tâches	Hétérogène	Centralisé	Sans migration
[60]	Mono-tâche	Hétérogène	Distribué	Sans migration
[61]	Mono-tâche	Hétérogène	Distribué	Avec migration

Le concept de migration de tâche dans les architectures hétérogènes est exploité pour améliorer les performances du système en plaçant une tâche dans l'IP le plus approprié (par exemple : affecter une nouvelle tâche a l'IP qui l'exécute le plus rapidement ou qui consomme le moins d'énergie et migrer la tâche qu'il exécute à un autre IP).

Une comparaison entre le mapping statique et dynamique pour une architecture 2D maillée homogène est discutée par Carvalho et al [62]. Les algorithmes SA (Simulated Annealing) et TS (Taboo Search) pour le mapping statique et les

algorithmes PL (Path Load) et BN (Best Neighbor) pour le mapping dynamique sont implémentés. Les tests prouvent les avantages du mapping dynamique par rapport au mapping statique (avec une réduction de 8.5% de la latence, 15.5% consommation d'énergie et 3.5% pour le temps d'exécution total).

Certains travaux considèrent le problème de mapping seulement, d'autres le combinent avec l'ordonnancement, le routage ou même avec le floorplanning. Quelques travaux combinant le mapping avec d'autres problèmes sont cités dans la prochaine section.

2.3.3. Mapping combiné avec d'autres phases

2.3.3.1. Ordonnancement

Les applications exécutées dans le contexte d'un NoC sont divisées en tâches. La représentation d'une telle application est modélisée sous forme de graphe de tâches (Figure 2.5): les nœuds désignent les tâches et les arcs désignent les relations de précédences entre ces tâches. L'ordonnancement consiste à choisir pour chaque tâche l'IP qui l'exécute et à quelle date. Plusieurs critères peuvent être considérés dans cette phase : le temps total d'exécution, le temps moyen que passe une tâche dans le système, l'énergie consommée par la tâche dans un IP, l'affectation de plusieurs tâches à un même IP et l'architecture du système (homogène ou hétérogène).

Une fois l'application modélisée de manière simple sous forme de blocs d'IPs avec un ordonnancement des tâches, vient ensuite l'étape de Mapping. Le Mapping permet le placement de ces IPs sur l'architecture du réseau sur puce. Les auteurs en [63, 64] combinent l'ordonnancement et le mapping.

2.3.3.2. Routage

Le routage détermine l'acheminement des paquets entre les nœuds source et destination. En [57], les auteurs traitent le problème de routage afin de construire un routage qui garantit l'absence d'interblocage (*deadlock-free*) qui minimise

l'énergie totale de communication (*dans le mapping et le routage*). Et dans [55**,** 65], le but du routage est de déterminer un routage sans interblocage (*deadlock-free*) et un chemin minimal pour chaque trace de communication. L'interblocage se produit lorsque plusieurs paquets se bloquent mutuellement car ils détiennent des ressources *demandées* et demandent d'autres déjà *détenues*.

L'algorithme de routage le plus utilisé est l'algorithme XY pour une topologie 2D maillée (Figure 1.9) : les paquets circulent dans la direction X puis dans la direction Y. Cet algorithme est simple à implémenter.

2.3.3.3. Floorplanning

La phase de Floorplanning consiste à définir le placement physique (position) des différents composants du circuit sur la puce. Afin d'implémenter les composants sur une puce, ces composants sont arrangés sous forme de blocs rectangulaires [66]. Le problème de floorplanning, appelé aussi problème de placement de blocs peut être formulé comme suit [67] : donnant un ensemble de modules rectangulaires et les informations d'interconnexions entre ces modules, trouver la surface de placement minimale avec les plus courtes liaisons (*wirelength*).

A cause de sa complexité, le floorplanning est généralement divisé en deux étapes [68] : la première consiste à trouver les positions relatives des modules (la topologie) et la deuxième à trouver les positions exactes, la surface et la taille des modules. Les modules ou composants qui communiquent fréquemment sont de préférence placés tout près [67]. La première étape dans le floorplanning permet de définir la topologie du circuit (qui spécifie l'organisation physique du circuit et donc la position relative de chaque composant du circuit est fixée). Dans [69], cette étape est définie sous le nom de Mapping et son résultat est considéré comme entrée pour la phase de floorplanning avec la taille de chaque composant. De ce fait, le problème de floorplanning est réduit en une seule étape : trouver les positions exactes des composants [70].

Dans [71] le résultat de la phase de Mapping est utilisé comme entrée pour la phase de Floorplanning. Les mêmes auteurs proposent d'abord l'étape de Floorplanning afin de générer une topologie personnalisée [72]. D'autres travaux dans ce genre (*topologie personnalisée*) sont cités dans la section 2.3.4.

2.3.4. Mapping avec topologie régulière ou personnalisée

On constate que la topologie 2D maillée est la plus utilisée dans les réseaux sur puce. Ceci est dû à la facilité de son implémentation et son indépendance de la taille du réseau. En plus, le routage dans la topologie 2D maillée est simple [29]. La topologie 2D maillée est une topologie régulière qui assure qu'il existe toujours un chemin entre 2 *IPs* quelconques. Ceci n'est pas forcément un avantage si des *IPs* donnés ne se communiquent pas. Dans ce cas, ces chemins inutiles seront implémentés par des interconnexions inutiles et qui engendreraient des problèmes de surface supplémentaire, des capacités qui seraient au détriment du temps de transfert des données et de la dissipation d'énergie [31].

D'autres approches permettent la personnalisation de l'architecture d'interconnexion. Ces approches sont classées en deux catégories : La première catégorie concerne une architecture d'interconnexion régulière générique telle que 2D maillée où on y insère ou supprime des liens supplémentaires. Un exemple d'une topologie *irrégulière* est présenté dans la figure 2.8 [73] : Dans cet exemple on trouve une topologie de type maillée où les routeurs non utilisés sont simplement enlevés du réseau.

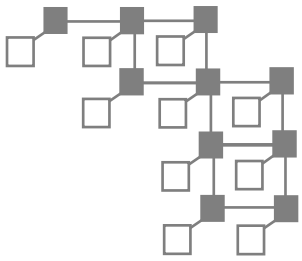


Figure 2.8 : Exemple d'une topologie irrégulière des réseaux sur puce basée sur la structure de type 2D maillée [73]

La méthode présentée dans [74] permet la détermination des routeurs présentant un goulot d'étranglement et des liens supplémentaires peuvent être insérés pour alléger leur charge et améliorer ainsi les performances.

Les approches de la deuxième catégorie personnalisent entièrement l'architecture pour la rendre plus appropriée à une application donnée. Par conséquent, l'architecture dépend de l'application, et n'est pas nécessairement conforme aux architectures régulières (par exemple, 2D maillée, ...).

A titre d'exemple, dans [75], un algorithme génétique a été proposé dont l'objectif principal est de minimiser la consommation d'énergie et la surface de silicium. L'algorithme génère plusieurs configurations (architecture totalement personnalisée) procédant en 3 étapes : La première étape permet de former des ensembles de routeurs de la bibliothèque choisie aléatoirement. Ensuite, pour chaque ensemble, affecter aléatoirement les IPs aux différents ports des routeurs. Et enfin, relier les routeurs via les ports routeurs aléatoirement, puis l'algorithme MSP (Modified Shortest Path) est appliqué afin de générer le plus court chemin (trafic) entre les nœuds source et destination.

L'approche proposée par Chatha et al [76] consiste a partir du floorplanning (placement des IPs, figure 2.9-a), allouer a chaque IP un routeur parmi les routeurs de ses coins, ensuite, minimiser le nombre de routeurs (*merging*) pour générer la topologie finale (figure 2.9-b). On trouve une approche similaire dans [77, 78].

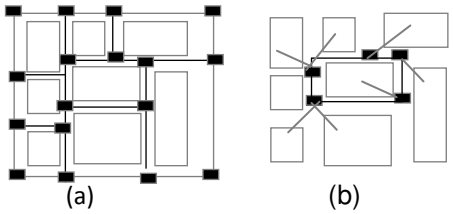


Figure 2.9: (a) Floorplanning. (b) Topologie finale [76]

Une comparaison entre les topologies régulières et personnalisées est discutée dans la thèse de Xinyu LI [79]. Le tableau suivant résume les principales différences :

Tableau 2.4 : comparaison de topologies régulière et personnalisée [79]

Topologie régulière	**Topologie personnalisée**
Maillée, torus, …	topologie Irrégulière
SoC utilité générale	SoC Application spécifique
Routeurs homogènes	Routeurs hétérogènes
Topologies réutilisées	Conception de routeurs réutilisée
Peu de temps de conception	Long temps de conception
Performances basses	Performances élevées
Energie élevée	Energie basse

2.4 Discussion

Après avoir présenté les critères d'évaluation et de classification des différentes stratégies de mapping, nous comparons dans cette section les travaux de chaque classe afin de montrer leurs avantages et limites. Le tableau 2.5 récapitule les techniques de mapping.

Tableau 2.5 : Classification des différentes stratégies de mapping

		Mapping statique	Mapping dynamique
Application	ASICs: Application spécifique	+	-
	MPSoC: Multiprocesseur	+	+
Méthode	Heuristiques	+	+
	Meta-heuristiques	+	-
Phases	Mapping seul	+	+
	Combiné	+	-
Topologie	Régulière	+	+
	Personnalisée	+	-

Le mapping statique est plus adéquat pour les applications spécifiques puisque le trafic de communication est connu à priori. Par contre, le mapping dynamique est plus approprié pour les processeurs a usage général (MPSoC) où le trafic de communication est identifié au moment de l'exécution de l'application. Dans le cas de mapping dynamique, l'architecture est prédéfinie, c'est-à-dire, les IPs sont déjà

placés sur les tuiles du NoC et les tâches peuvent être mappées au fur à mesure du besoin de l'application. Ceci permet un gain en ressources du NoC (seulement les tâches qui doivent s'exécuter sont mappées). Certains travaux [59, 60] ont exploité cette caractéristique pour mapper plusieurs applications (multi-applications) dans un même NoC. L'inconvénient majeur du mapping dynamique est le retard engendré par l'exécution de l'algorithme de mapping au moment de l'exécution de l'application (qui retarde l'exécution de l'application).

Pour le mapping dynamique, les heuristiques constructives sont plus appropriées. Ce sont des heuristiques gloutonnes permettant la génération de solutions partielles suivant les requêtes de mapping de tâches au moment de l'exécution de l'application. Les métaheuristiques transformatives à base de population de solutions sont plus utilisées pour le mapping statique multi-objective. Elles permettent de trouver un ensemble de solutions de mapping optimisant plusieurs objectifs pouvant être contradictoires. Les heuristiques constructives sont normalement plus rapides que les métaheuristiques transformatives [36]. Néanmoins, les métaheuristiques transformatives ont donné de meilleurs résultats que les heuristiques constructives [40, 47].

Combiner le mapping avec l'ordonnancement, le routage ou le floorlanning n'est possible que pour le mapping statique (au moment de la conception). Pour le mapping dynamique, les IPs sont déjà placés sur les tuiles de l'architecture du NoC, les tâches sont mappées à un IP approprié au moment de l'exécution de l'application. La combinaison de plusieurs phases de conception permet un gain en temps de développement (un même algorithme d'optimisation peut être appliqué aux différentes phases). La combinaison permet ainsi de considérer plusieurs critères afin d'optimiser un même objectif en même temps : la consommation de puissance, par exemple, est la consommation de puissance d'un IP quand il exécute une tâche (ordonnancement) ajoutée à la consommation de puissance due aux transferts de données entre IPs (qui dépend des positions des IPs et les chemins empruntés pour la communication). Les auteurs dans [63] formulent une même fonction objective afin de résoudre l'ordonnancement et le mapping en même temps.

Les topologies peuvent être personnalisées au moment de la conception (puisque les caractéristiques de l'application sont connues à priori). Pour le mapping dynamique, les topologies régulières sont les plus utilisées. Les topologies régulières requièrent peu de temps de conception et peuvent être réutilisées par d'autres applications mais les topologies personnalisées ont plus de performances. Dans [80], les routeurs ont été personnalisés afin d'optimiser l'énergie et la surface.

Nous pouvons également classer le mapping en trois types: le mapping des tâches, le mapping des IPs et le mapping des routeurs. Seul le mapping des tâches est possible en mapping dynamique puisque le réseau sur puce est déjà conçu. Pour le mapping statique, le mapping des tâches peut affecter considérablement les performances du système principalement dans une architecture homogène (à savoir les mêmes IPs sont utilisées). Pour une architecture hétérogène, l'ordonnancement des tâches et la mapping des IPs sont prises en considération. Pour la topologie personnalisée, on commence par le placement des IP (floorplanning), le mapping consiste à placer un nombre minimum de routeurs pour optimiser l'énergie et la surface.

Ces dernières années, un certain nombre d'efforts de recherche ont abordé la technologie d'intégration 3D. L'avantage des architectures tridimensionnelles est la réduction de la longueur des interconnexions, ce qui permet de réduire sensiblement la consommation d'énergie et la surface et augmente la performance des circuits [81], [82]. Il est montré qu'une architecture 3D peut réduire la longueur du fil autant que la racine carrée du nombre de couches empilées [83]. En plus des communications horizontales, l'intégration 3D permet des communications verticales à travers des interconnexions verticales courtes. Ces fils courts remplaçant les fils horizontaux longs, permettent la diminution de la résistance et la charge capacitive moyenne (la capacité et la résistance d'un fil sont proportionnelles à sa longueur) et réduire le nombre de répéteurs nécessaires à l'amplification du signal. Cette réduction va significativement réduire la consommation électrique globale au sein des circuits 3D. Une autre conséquence

majeure de la réduction de la résistance et de la capacité du fil est la diminution significative du délai de propagation du signal (proportionnel au produit résistance fois la capacité), entraînant un gain considérable des performances du système. Enfin, les technologies d'intégration 3D permettent de réduire la surface de la puce.

Plusieurs travaux [84, 85, 86, 87, 88, 89, 90, 91] basés sur des topologies en maille tridimensionnelle (figure 2.10) ont été développées ces dernières années. Des NoCs 3D nécessitent la construction de nouveaux routeurs (routeurs 3D), ainsi que le développement d'algorithmes de routage qui prennent en considération la 3ème dimension.

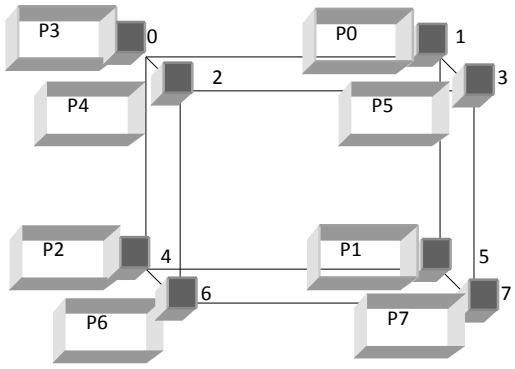


Figure 2.10: Mapping dans un NoC 3D

2.5 Conclusion

Ce chapitre a été consacré à la définition du problème de mapping et l'étude des stratégies existantes permettant sa résolution. De cette étude un framework de comparaison et d'évaluation des différentes stratégies de mapping été proposé [35]. Le framework que nous proposons permet aux concepteurs de partager une taxonomie pour qualifier chaque système selon quatre critères (la méthode développée, l'approche choisie, le flot de conception suivi et la topologie visée). Ce framework permet d'identifier les techniques les plus pertinentes et les principales contraintes dédiées à une famille d'application (à usage général ou

application spécifique) selon les objectifs visés des concepteurs de réseaux sur puce.

Les tendances actuelles visent à émerger de nouvelles technologies afin d'améliorer les performances des réseaux sur puce. L'intégration 3D, les interconnexions optiques, ainsi que les interconnexions sans fil, font partie des solutions proposées. Nous nous intéressons dans les travaux de cette thèse aux réseaux sur puce 3D.

Les circuits intégrés tridimensionnels peuvent être réalisés en utilisant des interconnexions verticales (principalement TSV- Through Silicon Vias). Il est évident que la performance maximale peut être obtenue par une connexion complète. Toutefois, à mesure que le nombre de tuiles augmente, il peut être difficile de supposer que chaque tuile sera reliée aux TSV correspondants en raison de la limitation du coût de fabrication et de la surface de la puce [89]. Ainsi, il est important de réduire la superficie totale des TSVs en minimisant leur taille (rôle du fabricant) et leur nombre (rôle du concepteur) afin de réduire le coût des circuits 3D. Par conséquent, l'idée de réseaux sur puce 3D partiellement connectés a été introduite pour bénéficier de la technologie 3D tout en conservant un haut rendement.

Notre contribution algorithmique permet un placement et une réduction efficaces d'interconnexions verticales sur une architecture 3D [92, 93] après un placement d'IPs sur ce type d'architecture [33, 34]. Ceci sera détaillé au prochain chapitre.

CHAPITRE 3
PLACEMENT EFFICACE DE LIENS VERTICAUX SUR UNE ARCHITECTURE 3D

3.1Introduction

Le réseau sur puce en 3D est une amélioration de celui en 2D. En plus des communications horizontales, l'intégration 3D permet des communications verticales à travers des liens verticaux courts. L'avantage principal de l'intégration 3D est la réduction considérable de la longueur des liens d'interconnexions et donc, la diminution du chemin parcouru pour le transfert de données.

Pour ce faire, nous considérons des communications entre des IPs disposés sur une architecture 3D. Lorsque 2 IPs se communiquant ne se trouvent pas sur le même plan de l'architecture, la communication entre 2 plans successifs doit se faire moyennant des liens verticaux. Le problème est que le nombre de liens verticaux a un impact sur les caractéristiques du réseau en matière de surface, vitesse et consommation de puissance. L'enjeu est donc de trouver un bon compromis pour cette problématique.

Nous commençons ce chapitre par une brève présentation de la technologie 3D dans les circuits intégrés. Nous abordons ensuite l'émergence de cette technologie dans la conception des réseaux sur puce. Et puis, dans les sections qui suivent, nous décrivons la solution de placement de liens verticaux proposée.

3.2 La technologie 3D

Les circuits intégrés tridimensionnels ont émergé comme une solution prometteuse pour poursuivre les progrès de la loi de Moore. La technologie 3D permet de réduire le problème des longues interconnexions dans les circuits intégrés 2D en empilant plusieurs couches verticalement.

Le concept de l'empilement vertical de dispositifs électroniques date des années 50. L'empilement de plusieurs puces (3D Die Stacking) les unes sur les autres était réalisé par des câbles extérieurs à la puce (Wire Bonding). La figure 3.1 [94] présente un empilement de plusieurs puces reliées électriquement par câblage extérieur. Cette technique était utilisée pour l'empilement de deux niveaux de mémoires (SRAM/ Flash, Flash /Flash,) [95]. Cependant, ce type de technologie occupe une grande surface (impose des connexions sur les bords des puces).

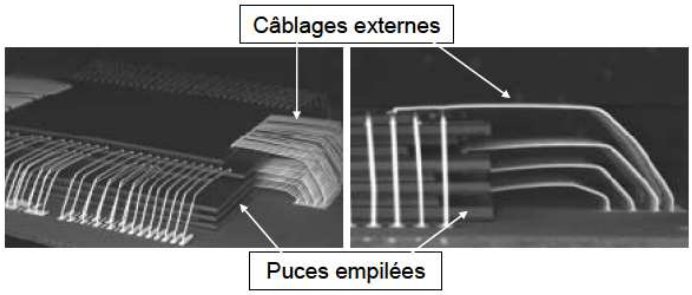


Figure 3.1: Empilement de plusieurs puces par Wire Bonding [94]

Une nouvelle alternative envisage la connexion des puces les unes aux autres par l'intermédiaire d'interconnexions inter-circuit. La technologie des TSVs (Through Silicon Via) est la plus soutenue actuellement par l'industrie. De nouvelles générations d'appareils photos numériques et de téléphones portables sont mises en production mettant à profit la technologie des TSVs. Samsung a annoncé en décembre 2011 avoir développé une barrette de 8 Go à base de puces mémoire interconnectées par des TSV qui était jusqu'à 40% plus économe en énergie [96]. Une interconnexion à base de TSVs (interconnexion verticale) permet de réduire considérablement la latence d'interconnexion par rapport aux interconnexions 2D. Néanmoins, elle présente un problème vis à vis de la surface réservée à la pose des TSVs. Il existe également une autre technologie d'interconnexions inter-circuit, nommée MIVs (Monolithic Inter-tier Via) de taille réduite par rapport aux TSVs (réduction de 8% de surface [97]). Notons que cette dernière technique est toujours au stade de recherche.

Dans une partie de nos travaux dans le cadre de cette thèse, nous nous intéressons à la réduction et au placement de liens verticaux dans une

architecture 3D. Dans ce qui suit, nous présentons quelques concepts liés aux réseaux sur puce dédiés aux architectures 3D.

3.3 Les réseaux sur puce 3D

Afin de permettre l'intégration de plusieurs IPs (des centaines d'IPs) et de répondre aux exigences des applications modernes, des réseaux de grande taille doivent être mis en place pour communiquer ces différents IPs. Cependant, le fait d'agrandir la taille de réseau a des mauvaises conséquences sur les performances de système sur puce. Cette dégradation est due principalement à l'augmentation de la longueur des interconnexions, qui augmente par conséquence la surface, les délais de transfert des données et la consommation de puissance à cause des nombreux sauts que doivent effectuer les messages avant d'atteindre leurs destinations respectives.

L'apparition de la technologie à 3 dimensions admet l'empilement des multiples couches en silicium ensemble. Ces couches communiquent à travers des liens d'interconnections verticaux plus courts (TSV Through Silicon Via) que celles utilisées en 2 dimensions (interconnexions horizontales). Cela a poussé la communauté des chercheurs à concevoir leurs réseaux en utilisant cette nouvelle technologie pour en exploiter les avantages par rapport aux réseaux sur puce à 2 dimensions.

L'amélioration des performances de réseaux 3D par rapport à celles de 2D est établie grâce à la réduction des distances parcourues entre les IPs se communiquant : réduction en longueur des interconnexions verticales ainsi qu'en nombre moyen de sauts ou nombre de routeurs traversés. En effet, le routeur dans un réseau à 3 dimensions possède des liens supplémentaires par rapport à un routeur de deux dimensions afin de connecter les différentes couches.

Certes la technologie 3D apporte des progrès considérables pour les réseaux sur puce, néanmoins de nouveaux défis se sont révélés pour la conception des réseaux sur puce 3D. Les concepteurs des réseaux sur puce 3D se sont focalisés sur deux problématiques :

-D'abord la problématique majeure est la conception des routeurs 3D ainsi que le développement de nouveaux algorithmes de routage pour prendre en considération les nouveaux chemins verticaux.
-La deuxième problématique est l'étude d'interconnexions verticales utilisées pour l'empilement de plusieurs couches (plans) à travers principalement des TSVs (Through Silicon Vias). Certes, l'insertion des TSVs permet une réduction de longues interconnexions horizontales par la diminution de nombre de sauts traversés pour le transfert des données. Cependant, les TSVs souffrent de la grande taille de la section des TSV par rapport aux transistors. Ce qui augmente par conséquence la surface totale du circuit. Ainsi, il est important de réduire la superficie totale des TSVs en minimisant leur taille (rôle du fabricant) et leur nombre (rôle du concepteur) afin de réduire le coût des circuits 3D.

L'idée de réseaux sur puce 3D partiellement connectés verticalement est parvenue afin d'exploiter l'intégration de la technologie 3D tout en conservant un haut rendement. Dans ce contexte, les chercheurs ont étudié le placement d'un nombre fixe de TSVs pour les architectures 3D partiellement connectés verticalement. Dans [89], des emplacements d'un nombre fixe (complet, demi et quart) de TSVs sont comparés avec une topologie maillée 3D (4*4). L'expérimentation a montré une réduction de 5.24% et 2.18% de latence du circuit avec 100% et 50% d'interconnexions verticales par rapport à 25%. La même idée est étudiée dans [90] pour une architecture 3D (8*8) afin de comparer un placement de 25% de liens verticaux avec un placement de 100%. Les résultats dévoilent une augmentation de 18.7% de latence pour une configuration de 25%. Dans [91] le placement des TSVs est soumis à la contrainte qu'il doit y avoir un minimum de deux sauts entre les TSVs. Le placement d'un nombre fixe de TSVs a engendré une suppression de liens verticaux, ce qui complique encore le routage des architectures partiellement connectées. Ainsi, de nouveaux algorithmes de routage sont nécessaires.

Pour conclure sur ces différentes techniques de placement de liens verticaux au sein d'un réseau sur puce, on constate que :

-d'une part, un nombre fixe de TSVs est considéré dans les différentes études;
-d'autre part, le placement de ces TSVs avec une énumération exhaustive (méthode exacte) n'est possible que pour des architectures réduites.

Nous proposons dans les travaux de cette thèse de développer une technique qui permet l'optimisation du nombre de liens verticaux (TSVs/MIVs) ainsi que leurs placements en utilisant le concept d'heuristique afin de prendre en considération des applications de grande taille.

Avant de détailler notre démarche, nous expliquons dans la section suivante la nouvelle méthodologie de conception des réseaux sur puce fournie dans le contexte de projet de cette thèse. Dans la suite, l'écriture « 3D partiellement connecté » est utilisé à la place de 3D partiellement connecté verticalement.

3.4 Nouvelle méthodologie de conception

Dans le cadre de projet mené au niveau de la division Microélectronique et Nanotechnologie du centre de développement des technologies avancées, une nouvelle méthodologie de conception de réseaux sur puce est proposée [31]. Dans cette méthodologie, seuls les IPs se communiquant sont connectés entre-eux (figure 3.2). Les IPs qui communiquent fréquemment les uns avec les autres sont placés côte à côte et une insertion de répéteurs peut être employée pour les éventuelles longues interconnexions.

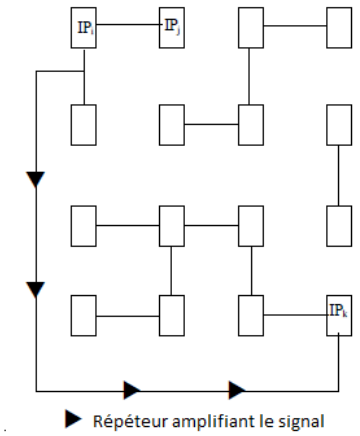


Figure 3.2 : L'architecture de la topologie proposée dans [31]

L'architecture générée n'est pas de type d'interconnexion point-à-point où pour chaque paire d'IPs communiquant il existe un fils dédié uniquement à l'échange de données entre ces deux IPs (section 1.2.1, chapitre 1). Dans cette méthodologie, les interconnexions sont partagées et permettent aussi des transferts parallèles entre des IPs. Par exemple, pour l'ensemble des transferts T_{ij} (de l'IP_i vers l'IP_j) entre 4 IPs de la figure 3.3, quatre interconnexions sont proposées (S1 à S4) : S1 pour T14 et T42, S2 pour T24 et T23, S3 pour T12 et S4 pour T32. Les transferts qui se font sur les différentes interconnexions peuvent être parallèles. Par contre, celles qui partagent la même interconnexion ne se font pas en parallèle.

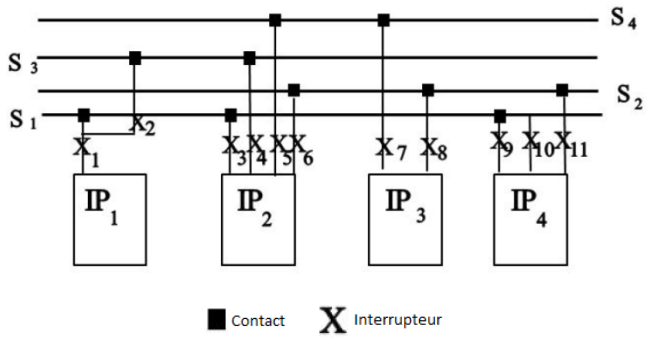


Figure 3.3 : Connexion des ports d'IPs aux interconnexions [31]

Le nombre d'interconnexions et l'affectation des transferts aux interconnexions sont définis par une méthode développée dans [31]. Les ports d'IPs sont reliés aux interconnexions utilisées pour leurs transferts par des contacts VIAs sur le même plan de l'architecture (voir figure 3.3). Les routeurs sont simplement remplacés par des transistors de passage (interrupteurs) pour l'aiguillage des données et le routage est spécifié pendant la phase de conception (une unité de commande permet le contrôle de ces transistors de passage). Ce qui permet des gains en surface, en latence et en consommation de puissance. En plus, puisque le routage est déterminé lors de la conception (en exécution, le routage ne consomme pas beaucoup de temps), le budget de temps est presque consommé par l'application elle-même, et non pas par le routage, ce qui sera utile pour la conception des systèmes temps réel.

Les travaux présentés dans [31] et [98-100] illustrent bien les apports qu'offre cette nouvelle méthodologie pour l'amélioration des performances des réseaux sur puce dédié aux applications spécifiques.

Pour le placement des IPs, l'idée de mettre les IPs qui communiquent fréquemment côte à côte en se basant sur une liste des priorités pré-établie (en commençant par ceux qui ont de forts degrés de communication) est considéré dans plusieurs travaux [49, 50, 51, 54]. De même dans [31], une disposition des IPs est établie selon une certaine méthode, ensuite les IPs sont placés sur l'architecture sous forme de serpent (pour assurer le placement côte à côte des IPs communiquant fréquemment). En passant à la 3ème dimension, le même principe de placement d'IPs sur les couches y est appliqué.

Dans le cadre de la méthodologie développée dans [31,32], et à partir d'un placement d'IPs sur une architecture 3D élaboré dans [33, 34], notre travail s'intéresse au problème d'insertion des liens verticaux. Plus précisément, il s'agit d'optimiser le nombre et le placement des liens verticaux afin de satisfaire les contraintes de surface, vitesse et consommation de puissance.

3.5 Méthodes de placement de liens verticaux proposées

Le nombre et le placement des liens verticaux ont un rôle clé dans les réseaux sur puce 3D partiellement connectés car ils offrent un nombre de sauts moyen pour le transfert des données. Par conséquent, ils affectent le délai, la surface et la consommation de puissance. Dans la réalité les objectifs qu'on cherche à atteindre sont antagonistes, car, par exemple, en minimisant la surface (en supprimant les liens verticaux) on ralentit la vitesse, ce qui affecte les performances du système. Donc on doit trouver un compromis entre ces objectifs, ce qui va nous mener à faire un traitement d'optimisation multi-objectif.

Par conséquent, l'exploration des topologies de NoCs 3D partiellement connectés est importante et non négligeable.

Pour comprendre l'importance de la réduction du nombre des liens verticaux au sein d'un réseau sur puce 3D partiellement connectés, nous avons mené une étude comparative afin de montrer l'intérêt de la démarche poursuivie.

Nous prenons comme exemple l'application VOPD (16 IPs) de la figure 3.4, et deux architectures : 2D (4*4) et 3D avec 2 couches (4*2) de la figure 3.5. On compare ces architectures par rapport au coût de communication total de l'ensemble des transferts effectués entre les IPs de l'application. Ce coût sera calculé comme suit :

Coût de communication=∑ (volume de communication* nombre de sauts)

Le volume de communication entre les IPs est indiqué sur les arêtes du graphe de communications. Le nombre de sauts dépend du chemin parcouru dans l'architecture (nous supposons le plus court chemin).

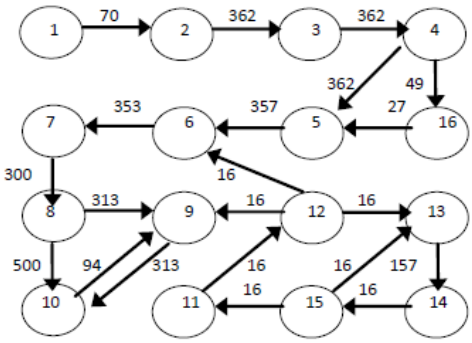
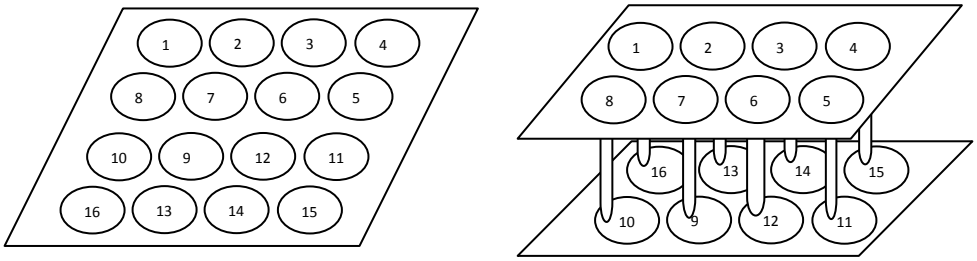


Figure 3.4 : graphe de communication de l'application VOPD



Architecture 2D 4*4 Architecture 3D 4*2

Figure 3.5 : Les architectures de l'application VOPD

Le coût de communication total de chaque architecture est résumé dans le tableau suivant:

Tableau 3.1 : Coût de communication de VOPD dans les différentes architectures

Architecture	2D 4*4	3D 4*2
Coût de communication	4981	4363

Il est à noter une réduction de 12,4% du coût de communication de l'architecture 3D par rapport à celui de l'architecture 2D. Cette réduction est due à l'insertion des liens verticaux qui permet de réduire le nombre de sauts entre les IPs. Par exemple, si on suppose que l'IP4 veut faire un transfert vers l'IP15, il est clair que le nombre de sauts dans l'architecture 3D est inférieur à celui de l'architecture 2D puisqu'un seul saut (le lien vertical) sera nécessaire pour ce transfert.

Pour le nombre de liens verticaux nécessaires aux transferts entre les couches de l'architecture 3D, nous avons constaté que le nombre de liens verticaux dans une architecture complète (8 pour l'architecture 4*2) dépasse le nombre de transferts entre les couches. Pour 20 transferts entre les IPs de l'application VOPD, seulement 5 transferts se font entre les 2 couches de l'architecture 4*2 (IP4 vers IP16, IP5 vers IP16, IP6 vers IP12, IP8 vers IP9 et IP8 vers IP10). A partir de cette simple constatation, il apparait clairement que le nombre de liens verticaux utilisés est au maximum égal au nombre de transferts entre les couches (5 au lieu de 8). Ceci, en conservant le même coût de communication (avec 8, 7, 6 ou 5 liens verticaux, le coût de communication sera toujours le même pour l'architecture 3D 4*2). Non seulement il y'a des liens non utilisés (qui consomment donc de la surface sans réduire le délai des transferts), des transferts peuvent partager les liens verticaux afin de réduire la surface mais ce serait au détriment de le la vitesse de transferts.

Donc, il est évident qu'un gain considérable en performance peut être atteint avec les plans de l'architecture 3D partiellement connectés. Dans notre démarche, nous proposons d'optimiser le nombre de liens verticaux (gain en surface) ainsi que leurs placements (gain en vitesse). De ce fait, le problème sera formulé ainsi :

À partir du placement des IPs sur l'architecture 3D (graphe d'architecture) [33, 34] et l'ensemble des transferts entre ces IPs (graphe de communications), nous déterminons le nombre et l'emplacement des liens verticaux (TSVs/MIVs) qui permet d'optimiser les contraintes (délai, consommation de puissance et surface).

Graphe d'architecture : Le graphe d'architecture du NoC A (V, L) est un graphe orienté où chaque sommet désigne un IP mappé sur l'architecture du NoC, et chaque lien définit un lien de communication physique entre deux IPs voisins. Chaque lien peut être pondéré par un coût d'envoi d'un paquet sur lui. Dans le cas où le coût est le nombre de sauts effectué pour le transfert entre 2 IPs, nous considérons le coût de chaque saut égal à 1 (la pondération d'un saut ou un pas égal à 1).

En plus des liens (horizontaux) reliant les IPs d'un même plan (même couche), des liens verticaux permettent de connecter deux couches constituant une architecture 3D. Nous supposons initialement une architecture avec une configuration complète (il existe des liens horizontaux et verticaux entre tous les IPs voisins) afin de définir les positions possibles des liens verticaux (liens verticaux candidats).

Graphe de communication (ou graphe d'application) : Le graphe de communication G (V, E) est un graphe orienté, où chaque sommet désigne un IP de l'application. Un lien orienté eij $\in$ E désignent la communication (le transfert) entre IPi et IPj. Chaque lien est affecté d'un poids w(eij) qui représente le nombre de paquets envoyés par IPi à IPj. Nous nous intéressant seulement aux transferts entre deux IPs appartenant à des plans (couches) différentes (nécessitant des liens verticaux).

En utilisant les deux graphes définis ci-dessus, le problème peut être formulé de la manière suivante :

La fonction **aff ( )** qui consiste à affecter à chaque Transfert eij $\in$ E de l'application un lien vertical LVk de l'architecture est définie comme suit :

$$\text{aff}() \begin{cases} \text{E} \rightarrow \text{L} \\ \text{eij} \rightarrow \text{LVk} \end{cases}$$

$\forall\, \text{eij} \in \text{E}, \exists\, \text{LVk} \in \text{L}: \text{aff}(\text{eij}) = \text{LVk}$ …….. (1)

Dans le graphe d'architecture, il existe toujours un chemin constitué de plusieurs liens (horizontaux et verticaux) permettant la communication entre deux IPs non voisins. Soit d_{ij} la distance entre IPi et IPj. Pour calculer dij, nous considérons le plus court chemin de chaque transfert entre 2 IPs passant par un LV_k.

dij=distance (IPi, LVk)+ distance (LVk, IPj) k=1,2, …, nbLV ……(2)

Avec nbLV le nombre maximum de liens verticaux dans l'architecture 3D.

A chaque affectation un coût est associé :

coût = w(eij)*dij …….. (3)

On veut déterminer le nombre et positions des liens verticaux afin d'assurer tous les transferts entre les IPs appartenant à des plans différent dont l'objective d'optimiser le délai, la consommation de puissance et la surface totale du circuit. Comme expliquer précédemment, ces objectives sont antagonistes, car, en minimisant la surface (en supprimant les liens verticaux) on augmente le délai et la consommation de puissance. Ces objectives sont formalisés par les équations (4) et (5) ainsi :

min coût= $\sum \text{w(eij)} * \text{dij}$ …… (4)

min $\sum_{\text{k}\in\{1..\text{nbLV}\}} \text{LV}_\text{k}$ …… (5)

Afin de trouver la solution optimale, nous pouvons essayer toutes les combinaisons possibles de nombre et placement de lien vertical, et puis, prendre la meilleure solution. Une énumération exhaustive n'est possible que pour des petits espaces de recherche, tandis que la recherche guidée par les heuristiques est requise pour les grands espaces de recherche.

Plusieurs techniques d'optimisation multi-objective existent. Notre démarche consiste à minimiser le nombre de liens verticaux (gain en surface), puis, optimiser leurs placements (gain en délai et consommation de puissance). Donc, nous avons choisi l'approche non Pareto (en transformant le problème multiobjectif en

monoobjectif) en développant des méthodes scalaires de type e-contraintes (qui consiste à choisir une seule fonction objective à optimiser et transforme les autres fonctions en contraintes**)** afin de minimiser la surface sous les contraintes de délai et consommation de puissance.

La fonction objective devient l'équation 5 et la contrainte à satisfaire l'équation 6 :

$\min \sum_{k\in\{1..nbLV\}} LV_k$ …… (5)

Coût= $\sum w(eij) * dij \leq$ Contrainte Coût …… (6)

Les deux heuristiques spécifiques [92, 93] seront détaillées dans les sous sections de 3.5.1à 5.3.4. Nous présentant ensuite la métaheuristique développée dans les sous sections 3.5.5 et 3.5.6. L'analyse des résultats des différentes méthodes sera discutée dans la sous section 3.6.

3.5.1La première Heuristique

Nous procédons en trois étapes :

(1) dans la première étape, nous déterminons le nombre maximum de liens verticaux candidats (LVs) ainsi que leurs placements optimaux,

(2) la deuxième étape consiste à calculer le coût des transferts pour chaque LV candidat (supposons que chaque LV peut être utilisé pour n'importe quel transfert),

(3) finalement, trouver la solution pré-optimale voire optimale qui minimise le coût total.

Afin d'obtenir le placement efficace (pré-optimal voire optimal) des liens verticaux, nous considérons le plus court chemin de chaque transfert entre 2 IPs. Le nombre de placements possibles de LVs détermine le nombre de LVs candidats. Nous avons initialement pris pour chaque transfert entre 2 plans successifs deux positions (LVx et LVy). Donc on aura le nombre de LVs candidats maximum qui est égal à 2* nombre de transferts. Les positions LVx et LVy sont calculées à partir des positions des IPs d'un transfert donné. Soit un transfert entre IP1 (position-i1, position-j1) du plan 1 et IP2 (position-i2, position-j2) du plan 2 : la position de LVx= (position-i1, position-j2) et la position de LVy= (position-i2 , position-j1). Notons

que les LVs candidats déterminent les chemins possibles pour effectuer un quelconque transfert entre deux IPs (Les LVs candidats déterminent les chemins possibles afin d'effectuer un transfert entre 2 IPs quelconques).

L'étape 2 consiste à estimer le coût de chaque transfert par rapport a chaque LV candidat. Donc, chaque chemin (un transfert passant par un LV) est caractérisé par un coût. Le coût représente le nombre de sauts parcourus dans un chemin. Pour chaque LV candidat, nous calculons le coût total (coût LV) estimé pour effectuer tous les transferts entre les IPs. Ce coût sera utilisé dans l'étape 3 pour donner une certaine priorité aux candidats à faible coût (un candidat qui fait plus de transferts en un minimum de sauts sera plus favorisé).

L'étape 3 permet de chercher la solution voulue. Cette solution donne le nombre et le placement des liens verticaux qui minimisent le coût total des transferts (coût T). Ainsi, pour chaque combinaison de chemin de l'étape 2, nous calculons le coût total des transferts. La solution optimale est celle qui donne le coût total minimal. Pour N transferts, on aura $(2*N)^N$ combinaisons de chemins possibles. Soit l'exemple de 2 transferts qui peuvent passer chacun par 4 chemins (chaque LV définit un chemin différent). Chaque transfert peut emprunter l'un des 4 chemins et chaque chemin a son coût (tableau 3.2). Pour chaque combinaison de chemins un Coût T est calculé (tableau 3.3).

Tableau 3.2: Tableau des chemins (chaque chemin est caractérisé par un coût)

	Chemin C1	Chemin C2	Chemin C3	Chemin C4
Transfert T1	Coût1	Coût3	Coût5	Coût7
Transfert T2	Coût 2	Coût4	Coût6	Coût8
Coût LV	Coût1+ Coût 2	Coût3+ Coût4	Coût5+ Coût6	Coût7+ Coût8

Tableau 3.3 : Les combinaisons des chemins possibles

	Chemins des transferts		**Coût T**
	T1	**T2**	
Combinaison 1	C1	C1	Coût1+ Coût 2
Combinaison 2	C1	C2	Coût1+ Coût4
Combinaison 3	C1	C3	Coût1+ Coût5

Combinaison 4	C1	C4	Coût1+ Coût7
Combinaison 5	C2	C1	Coût 3+ Coût2
….	….	….	…..
Combinaison 16	C4	C4	Coût7+ Coût8

Pour un nombre important de combinaisons possibles, une heuristique gloutonne est proposée :

L'idée de l'heuristique est de démarrer avec un seul LV et d'insérer les LVs candidats un par un jusqu'à trouver une solution qui satisfait la contrainte (coût inférieur à un coût max défini par l'utilisateur comme une donnée d'entrée à l'application). L'heuristique gloutonne est décrite dans l'Algorithme 1.

Les LVs insérés sont triés selon l'ordre croissant de leurs coûts du transfert calculés à partir de l'étape précédente. Le coût total d'un LV est la somme des coûts de chaque transfert, nommé coût LV dans le tableau 3.2.

A chaque insertion d'un nouveau LV, le candidat est affecté à un ensemble de transferts (le choix des transferts ainsi que le nombre de transfert sont aléatoires). L'affectation est effectuée k fois pour modifier l'affectation du LV candidat aux différents transferts (le paramètre k est aussi aléatoire). On prend la solution (l'affectation) qui minimise le coût total.

Algorithme 1: Heuristique gloutonne: Placement d'un minimum de liens verticaux

Entrées : l'ensemble des LVs candidats et leurs coûts pour chaque transfert entre 2 IPs quelconques et la contrainte de coût.
Sorties: le nombre minimum de LVs et leurs positions

1: Insérer un nouveau candidat LV (de coût LV minimum);
2: for (i=1 to K) //le paramètre k est aléatoirement généré
    Affecter aléatoirement le candidat sélectionné à un ensemble de transferts ;
    Calculer le coût de cette affectation ;
4: Prendre le coût minimum (parmi les k coûts trouves dans l'étape 2);
5: Calculer le coût total (coût T) ;
6: Si (coût total > contrainte de coût) et (nombrer LVs < max LVs) aller à l'étape 1;
7: Si (coût total > contrainte de coût) et (nombre de LVs = max LVs) retourner contrainte non satisfaite ;
8: Si (coût total ≤ contrainte de coût) retourner le nombre et les positions des LVs.

3.5.2Expérimentations

Dans cette section, nous présentons l'évaluation expérimentale d'un nombre différent de transferts (9 benchmarks aléatoires) entre 32 IPs pour une architecture 3D 2*8 (2 couches de 2 lignes et 8 colonnes). Les résultats sont résumés dans le tableau 3.4.

La colonne #transfert spécifie le nombre de transferts effectués entre les IPs, la colonne #LV donne le nombre de LVs proposé. Pour les colonnes coût, nous calculons le coût minimum utilisant seulement un seul LV, ensuite avec une méthode exacte (une énumération exhaustive), le coût minimum de toutes les combinaisons possibles est proposé (pour les 4 premier benchmarks). Enfin, pour notre heuristique, une contrainte de coût est introduite par l'utilisateur afin d'améliorer le coût (diminuer le délai en ajoutant des liens verticaux, ce qui serait au détriment de la surface).

Tableau 3.4: Résultats expérimentaux de la première heuristique

Benchmark	#transfert	Coût 1 VL	Méthode exacte		Notre heuristique		
			Coût	#LV	Contrainte coût	Coût	#LV
1	2	17	9	2	15	9	2
2	3	25	11	2	24	21	2
					20	11	3
					15	11	3
3	4	35	11	3	30	25	2
					20	19	3
					12	11	3
4	5	42	10	5	35	34	2
					25	20	4
					11	10	5
5	6	46	-	-	45	44	2
					35	32	4
					30	30	4
6	8	68	-	-	60	58	3
					50	46	4
					40	36	4
7	10	92	-	-	80	76	2
					70	68	4
					60	66	4
8	20	168	-	-	150	146	5
					145	138	5
					140	136	6
9	40	330	-	-	320	320	2
					310	306	3
					300	286	4

3.5.3 La deuxième Heuristique

Pour la deuxième heuristique, nous considérons seulement un seul lien vertical pour chaque transfert (plus court chemin). Donc, le nombre de liens verticaux candidats est égal au nombre de transferts. Après la définition du nombre maximum de candidats avec leurs placements optimaux, l'heuristique cherche la solution qui minimise la surface tout en satisfaisant les contraintes de délai et de consommation de puissance.

Cette fois-ci, nous démarrons avec une surface maximale, c'est-à-dire, tous les LVs candidats sont placés (donc, le délai est le minimum). Afin de diminuer la surface, nous procédons par la suppression graduelle de LVs tout en nous assurant à toujours respecter les contraintes de délai et de consommation de puissance. La suppression ou la diminution de nombre de LVs est réalisée comme suit : nous groupons deux LVs voisins dans un même cluster (auquel nous définissons un LV commun). Ensuite, pour chaque cluster, la nouvelle position est calculée en nous basant sur le centre de gravité des LVs regroupés. L'opération est itérée tant que les contraintes restent satisfaites (figure 3.6).

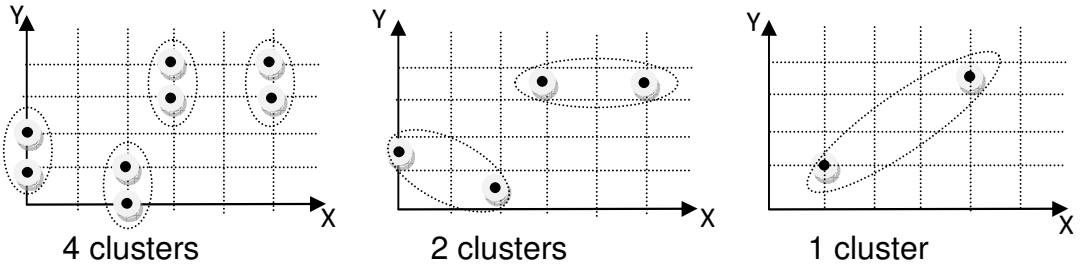


Figure 3.6: Groupement de deux LVs voisins dans un même cluster

L'algorithme 2 résume les étapes de l'heuristique :

Algorithme 2 : Heuristique à base de Clusters

Entrées : l'ensemble des LVs candidats et leurs positions initiales ainsi que les contraintes de délai et de puissance
Sorties: le nombre minimum de LVs et leurs positions

1: Faire
  2: Former clusters de deux voisins LVs;
  3: Remplacer les LVs de chaque cluster par le nouveau LV dont la position est leur centre de gravité;
  4: Affecter les nouveaux LVs aux transferts (remplacer les LVs supprimés par les nouveaux);
  5: Recalculer le délai et la consommation de puissance;
  6: Tanque (Nb_cluster>1 et contraintes non violées);
7: retourner le nombre et les positions des LVs

3.5.4 Expérimentations

Les résultats expérimentaux de 4 benchmarks (les applications PIP, VOPD, MWD et B1 benchmark aléatoire) sont résumés dans le tableau 3.5. La colonne #IPs dénote le nombre des IPs de l'application, la colonne architecture présente le nombre de lignes et colonnes de chaque couche. Les 3 dernières colonnes indiquent le nombre de liens verticaux (ici nous avons choisi les liens verticaux de type miv - Monolithic tier via- pour utiliser les outils de calcul du délai et consommation de puissance fournis par le CDTA). La colonne Complète #mivs indique le nombre de LVs maximum (nombre de lignes *nombre de colonnes *nombre de couches-1 pour chaque architecture). Dans la colonne Initiale #mivs, est donné le nombre de LVs initial qui représente le nombre de LVs candidats. Enfin, la dernière colonne montre le nombre de LVs déduit par notre heuristique sous 4 contraintes. Les contraintes sont exprimées par un pourcentage de diminution maximale permise de délai et de consommation de puissance par rapport à la solution initiale (nombre de LVs initial).

Tableau 3.5 : Résultats expérimentaux de la deuxième heuristique

Benchmark	#IPs	Architecture (3 couches)	Complète #mivs	Initiale #mivs	Optimal #mivs (% diminution maximal permise)			
					18%	20%	50%	70%
PIP	8	1*3	6	4	-	-	-	2
VOPD	16	3*2	12	3	-	-	2	2
MWD	12	2*2	8	5	-	-	2	2
B1	90	20*2	80	30	6	4	2	2

Pour l'application PIP (figure 3.7), l'heuristique permet de réduire le nombre de liens verticaux initialement de 6 à 4, ensuite de 4 à 2 (figure 3.8), soit une réduction totale de 66.67% avec une augmentation de 28.7% et de 66.43% de délai et de consommation de puissance, respectivement.

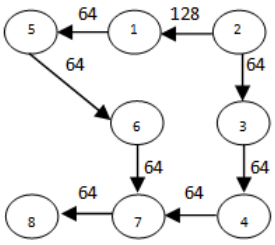
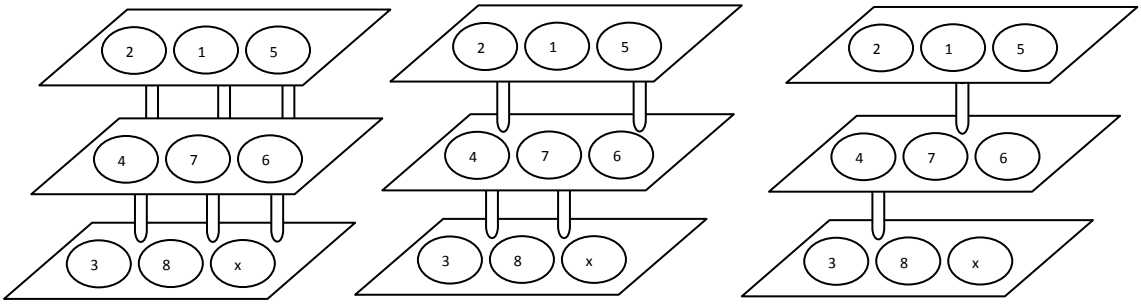


Figure 3.7 : Graphe de communication de PIP



Architecture compète Architecture initiale Architecture optimisée

Figure 3.8 : L'optimisation du nombre et du placement des liens verticaux

Notre heuristique permet pour le Benchmark B1 de 90 IPs une réduction initiale de 62,5% (80 à 30) de liens verticaux. A partir de la solution initiale une réduction de 80%, 86,66% et 93% de liens verticaux est obtenue avec juste 17.94%, 18.36% et 22.62% d'augmentation en consommation de puissance, respectivement.

Nous avons aussi développé la métaheurstique du recuit simulé. Une méthode plus générique, indépendante de la structure du problème traité et peut être appliquée à un large ensemble de problèmes grâce à son adaptabilité. Appelé aussi métaheuristiques de trajectoire, car elle fait évoluer une seule solution sur l'espace de recherche à chaque itération. Le paragraphe suivant décrit son détail.

3.5.5 Métaheurstique du Recuit simulé

Le recuit simulé s'inspire du phénomène de réchauffage et refroidissement lent des métaux. En partant d'une température élevée où le métal serait liquide, on refroidit le métal progressivement jusqu'à avoir des métaux solides sans défaut (par la recherche d'un état d'énergie minimum correspondant à une structure stable de ce métal). Par analogie en optimisation, le recuit simulé permet de parcourir de manière itérative l'espace des solutions. La fonction objective, assimilée à l'énergie d'un matériau, est alors minimisée, moyennant l'introduction d'une température fictive qui est contrôlée par une fonction décroissante définissant un schéma de refroidissement. Passer d'un état du métal à un autre correspond à passer d'une solution à une solution voisine. À chaque nouvelle itération, une nouvelle solution est générée de manière aléatoire dans le voisinage de la solution courante. Cette nouvelle solution peut améliorer la fonction objective en diminuant l'énergie du système. Dans le cas où la nouvelle solution est moins bonne alors elle sera acceptée avec une certaine probabilité P qui dépend de la

valeur de la température initialement choisie très grande (une température élevée correspond à une probabilité plus grande d'accepter des dégradations, la probabilité d'accepter des dégradations est diminuée progressivement).

L'algorithme 3 décrit l'heuristique du recuit simulé adaptée a notre problème. La solution initiale S_0 est générée aléatoirement en affectant un LV candidat aléatoire à tous les transferts. Ensuite dans une boucle tant que la contrainte n'est pas satisfaite et la température permet d'accepter de nouvelles solutions, nous générons une nouvelle solution voisine en modifiant la solution courante. La nouvelle solution admet l'affectation d'un LV candidat à un ensemble de transferts aléatoirement. Une deuxième version est utilisée pour le choix du LV candidat (pour la solution initiale) en priorisant le LV du coût minimum (comme dans la première heuristique, le candidat qui fait plus de transferts en un minimum de sauts sera plus favorisé).

Algorithme 3 : Recuit simulé

Entrées : l'ensemble des LVs candidats et leurs coûts pour chaque transfert entre 2 IPs quelconques et la contrainte de coût.
Sorties : le nombre minimum de LVs et leurs positions

$T=T_{max}$;
S_0=GénérerSolutionInitiale();/* un seul LV pour tous les transferts*/
E_0=CalculerCout(S_0);
$S_{best}=S_0$; $E_{best}=E_0$;
K=1 ;
Tantque ((E_{best}>ContrainteCoût) et ($T>T_{min}$))
Faire
S_{new}=GenererSolutionVoisine(); /*Affectation aléatoire d'un LV a un ensemble de transferts*/
E_{new}=CalculerCout(S_{new});
dE=(E_{new}-E_{best})/E_{best} ;
p=exp(-dE/T) ;
Si (dE≤0) ou(dE>0 et p>random(0,1))
$S_{best}=S_{new}$;
$E_{best}=E_{new}$;
Finsi ;
$T=T^*$ α;
k=k+1;
Fait
Si ($T=T_{min}$ et E_{best}>ContrainteCoût) retourner contrainte non satisfaite ;
Sinon retourner S_{best} (Le nombre et position des LVs) ;

3.5.6 Expérimentations

Les résultats expérimentaux des applications PIP et VOPD sont résumés dans les tableaux 3.6 et 3.7. La température initiale est fixé à Tmax=10^5, la température minimale à Tmin=10^{-5} et α=0.9.

Tableau 3.6: Résultats expérimentaux du recuit simulé version 1

Benchmark	**Recuit simulé V1**						
	Coût Solution Initiale	Contrainte Coût	Solution finale				
			Itération k	Coût	#LV	Position	Transfert
PIP	5	4	17	3	2	1, 2 2,1	T1 T2
	7	3	20	3	2	1,1 2,1	T1 T2
VOPD	23	17	28	17	4	2,3 1,1 2,2 2,1	T1 T2 T3 T4 et T5
	21	15	131	15	2	2,3 2,1	T1 et T3 T2, T4 et T5
	19	13	117	13	3	1,4 2,1 2,3	T1 T2, T4 et T5 T3

Tableau 3.7: Résultats expérimentaux du recuit simulé version 2

Benchmark	**Recuit simulé V2**						
	Coût Solution Initiale	Contrainte Coût	Solution finale				
			Itération k	Coût	#LV	Position	Transfert
PIP	5	4	34	3	2	1, 2 2,1	T1 T2
		3	12	3	2	1,2 2,1	T1 T2
VOPD	19	17	28	17	2	1,4 2,1	T1 T2,T3, T4 et T5
		15	52	13	3	1,4 2,3 2,1	T1 T2 et T3 T4 et T5
		13	161	13	3	1,1 2,3 2,1	T1 T2 et T3 T4 et T5

3.6 Discussion

Ce chapitre a porté sur la détermination du nombre et de l'emplacement de liens verticaux (TSVs ou MIVs) pour une architecture 3D. Ceci, à partir d'un graphe de communications et d'un emplacement d'IPs sur une architecture 3D [33, 34]. Pour ce faire, nous avons développé des heuristiques.

Nous avons d'abord procédé par fixer le nombre maximum de liens verticaux nécessaires pour effectuer tous les transferts de l'application ainsi que leurs placements adéquats. Ensuite, nous minimisons la surface (en optimisant le nombre de liens verticaux) tout en satisfaisant les contraintes de délai et de consommation de puissance (placement efficace des liens verticaux).

Dans cette étude, il a été mis en évidence que la réduction du nombre de liens verticaux et la détermination de l'emplacement de chaque lien vertical retenu ont un impact favorable sur les caractéristiques de l'architecture 3D. De ce fait, nous avons commencé par proposer un placement initial d'un maximum de liens verticaux (appelés candidats dans nos algorithmes). Les positions de ces liens sont choisies d'une façon à réduire le nombre de sauts utilisés pour les transferts. Par conséquent, à converger vers de meilleures valeurs de délai et de consommation de puissance.

Dans une deuxième étape, les heuristiques offrent la possibilité d'optimiser la surface en réduisant le nombre de liens verticaux sous certaines contraintes (dont la longueur des liens parcourus pour effectuer les transferts).

Les objectifs des heuristiques sont les mêmes (optimiser les performances : surface, délai et consommation de puissance). Mais, la façon de procéder pour atteindre ces objectifs est différente. Sachant que dans la métaheuristique du recuit simulé nous avons procédé par l'ajout de liens verticaux (comme dans la première heuristique spécifique). Le tableau 3.8 résume les points qui diffèrent :

Tableau 3.8 : Les points de différence entre les heuristiques

Heuristique 1 et Recuit simulé	**Heuristique 2**
Insertion de liens verticaux	Suppression de liens verticaux
Les positions des liens verticaux sont fixées	Les positions des liens verticaux sont calculées
Un transfert peut utiliser n'importe quel lien vertical	Chaque transfert utilise le lien vertical de son cluster
Coût = nombre de sauts	Coût = délai et consommation de puissance
Le type des liens verticaux est quelconque	Le type des liens verticaux est particulier

La première différence est le nombre de liens verticaux initial : dans l'heuristique 1 (ainsi que le recuit simulé), nous commençons par insérer un seul lien vertical entre deux couches successives. Ensuite, nous ajoutons les liens verticaux un par un jusqu'à atteindre le coût souhaitable. Par contre, dans l'heuristique spécifique 2, nous considérons, initialement, une surface maximale, c'est-à-dire que tous les liens verticaux sont placés sur l'architecture (maximum de liens verticaux). Ensuite, nous supprimons les liens verticaux afin de minimiser la surface, mais de manière à ne pas enfreindre les contraintes imposées sur le délai et la consommation de puissance.

Le deuxième point est l'emplacement des liens verticaux : les positions des liens verticaux sont fixés dans la première méthode et sont définies par les plus courts chemins des transferts entre IPs. Ces positions ne changent pas au cours de l'exécution de l'heuristique. De la même manière, les positions initiales sont définies dans la deuxième méthode, mais cette fois ci, les positions changent pendant l'exécution de l'heuristique. A chaque construction de nouveaux clusters, le centre de gravité des liens verticaux d'un même cluster représente la nouvelle position du lien vertical les remplaçant.

Dans la première heuristique, un lien vertical peut être utilisé pour n'importe quel transfert. En revanche, dans la deuxième, à chaque cluster est associé un lien vertical d'un ensemble de transferts d'une région précise. Donc, chaque transfert utilise le lien de son cluster.

Un autre point important est la fonction de coût qui permet de vérifier l'amélioration des performances du système. Dans la première méthode, nous avons utilisé une métrique qui est indépendante du type des liens verticaux, c'est-à-dire, la technologie choisie (TSVs/ MIVs). Cette métrique est le nombre de sauts effectués pour réaliser un transfert entre deux IPs. Le nombre de sauts dépend de la distance parcourue (qui dépend à son tour des chemins et liens verticaux disponibles) par chaque transfert et influence le délai et la consommation de puissance (un long trajet consomme plus de délai et de puissance). Pour la

deuxième heuristique, des outils d'estimation 3D de performances (délai et consommation de puissance) par rapport à la technologie utilisée sont nécessaires.

Le fait d'utiliser des estimations de coût différentes dans les heuristiques ne nous permet pas de comparer et dire quelle est la meilleure heuristique. De ce fait, nous avons décidé de mener une nouvelle expérimentation. Nous avons introduit la métrique de la première heuristique (nombre de sauts) pour l'estimation du coût dans la deuxième heuristique. Les heuristiques sont appliquées sur les mêmes benchmarks PIP et VOPD avec une architecture en deux couches. Les résultats obtenus sont résumés dans les tableaux 3.9, 3.10 et 3.11.

Tableau 3.9: Résultats expérimentaux de l'heuristique 1 pour PIP et VOPD

Benchmark	#transfert	Coût 1 VL	Méthode exacte		Notre heuristique		
			Coût	#LV	Contrainte coût	Coût	#LV position->transfert
PIP	2	5	3	2	4	3	2 : (1,1)-> T1 (2,1)-> T2
					3	3	2 : (1,1)->T1 (2,2)->T2
VOPD	5	19	13	3	17	16	2: (2,1)-> T1, T4 et T5 (2,2)-> T2 et T3
					15	15	3: (2,1)-> T2, T4 et T5 (2,3)-> T2 et T3 (2,2)-> T1
					13	13	3: (1,1)-> T1 et T2 (2, 1)->T4 et T5 (2, 3)->T3

Tableau 3.10 : Résultats expérimentaux l'heuristique 2 pour PIP et VOPD

Benchmark	#transferts	Initiale		Notre heuristique			
		#LVs	Coût	#LV	Coût	Position	Transfert
PIP	2	2	3	1	5	1,1	T1 et T2
VOPD	5	4	13	3	13	1,1 2,1 2,3	T1 T2, T4 et T5 T3
				2	22	1,1 2,3	T1, T2, T3, T4 ,T5 T3
				1	23	1,2	Tous les transferts

Tableau 3.11: Comparaison des résultats des heuristiques

Benchmark	#transferts	Heuristique spécifique 1		Heuristique spécifique 2		Recuit simulé			
						V1		V2	
		#LV	Coût	#LV	Coût	#LV	Coût	#LV	Coût
PIP	2	2 1	3 5	2 1	3 5	2 1	3 5	2 1	3 5
VOPD	5	3 2 1	13 16 19	3 2 1	13 22 23	3 2 1	13 15 19	3 2 1	13 17 19

Pour le benchmark PIP, les résultats sont les mêmes. Par contre pour le benchmark VOPD, la première technique (ajout progressive de liens verticaux) propose des coûts inférieurs avec le même nombre de liens verticaux. Ceci est dû au fait que la deuxième heuristique propose de nouvelles positions pour les liens verticaux (centres de gravités), ce qui peut augmenter le coût total des transferts.

Notons que la première technique exploite plus de pistes puisque chaque lien peut être utilisé pour n'importe quel transfert. La deuxième heuristique est plus rapide car à chaque itération le nombre de liens est réduit de 50%.

Afin d'enrichir notre expérimentation, nous avons ensuite comparé toute les méthodes basées sur l'ajout progressive de liens verticaux (l'heuristique 1, recuit simulé version 1 et recuit simulé version 2) utilisant les benchmarks du tableau 3.4. Le tableau 3.12 résume les résultats.

Les résultats obtenus sur les 25 expériences ont révélés que l'heuristique 1 propose des solutions meilleures que le recuit simulé. L'heuristique 1 est meilleure que la première version du recuit simulé sur 13 cas (52%) et 10 cas (40%) par rapport au coût et nombre de liens verticaux respectivement. Et 32% et 20% par rapport à la deuxième version.

Tableau 3.12 : Comparaison de l'heuristique 1 avec le recuit simulé

Benchmark	Contrainte coût	L'heuristique 1		Recuit Simulé						
				Version 1				Version 2		
		Coût	#LV	Coût Init	K	Coût	#LV	k	Coût	#LV
1	15	9	2	19	43	9	2	15	9	2
2	24	21	2	41	14	23	2	7	23	2
	20	11	3	25	30	11	3	19	11	2
	15	11	3	29	23	15	3	66	15	3
3	30	25	2	55	23	29	3	8	29	4
	20	19	3	47	38	15	3	57	19	2
	12	11	3	35	174	11	4	136	11	3
4	35	34	2	60	15	34	3	1	30	2
	25	20	4	46	92	24	3	15	14	3
	11	10	5	54	-	-	-	-	-	-
5	45	44	2	88	25	44	3	25	38	3
	35	32	4	62	78	34	3	44	30	3
	30	30	4	52	124	30	3	36	30	4
6	60	58	3	72	94	60	3	34	48	3
	50	46	4	104	156	48	3	110	50	4
	40	36	4	68	176	38	5	213	38	5
7	80	76	2	140	69	78	5	16	78	3
	70	68	4	130	98	68	5	160	58	4
	60	66	4	126	-	-	-	143	56	4
8	150	146	5	206	91	148	5	97	144	5
	145	138	5	176	117	144	5	124	140	3
	140	136	6	226	130	134	4	88	136	6
9	320	320	2	608	74	318	4	196	320	3
	310	306	3	388	108	294	5	207	306	5
	300	286	4	428	95	300	7	104	298	4

Les résultats de l'heuristique 1 sont les mêmes que celles du recuit simulé pour 28% (coût) et 32% (#LV) pour la version 1 et 32% et 60% pour la version 2. Néanmoins, après plusieurs itérations du recuit simulé (ce qui a nécessité plus de temps d'exécution).

Le recuit simulé a donné également de meilleurs résultats; pour 12% (coût) et 20% (#LV) pour la première version et 32% et 16% pour la deuxième. Ces résultats sont obtenus grâce à l'exploration plus large de l'espace de solution (le faite d'affecter un lien candidat aléatoire a un ensemble de transferts aléatoirement nous permet d'exploiter encore de nouvelles pistes).

Cependant, pour plusieurs cas, le recuit simulé n'a pas donné des résultats (n'a pas pu satisfaire la contrainte du coût) car la boucle du recuit simulé a atteint la température minimale avant de pouvoir trouver une solution respectant la contrainte (le nombre d'itération est limité).

3.7 Conclusion

De nombreux travaux de recherche cités dans la littérature montrent clairement l'avantage des architectures 3D par rapport aux architectures 2D. En effet, de longues interconnexions (présentant donc de grandes valeurs de résistance et de capacité) peuvent être remplacées par un lien plus court en disposant les IPs concernées sur deux plans successifs. Ceci a pour effet de diminuer les dites valeurs, permettant ainsi d'améliorer les caractéristiques de l'architecture en termes de vitesse et de consommation de puissance.

Toutefois, l'ajout de liens verticaux nécessaires aux communications entre des IPs disposées sur des plans différents de l'architecture n'est pas sans impact. En effet, chacun de ces liens verticaux (notamment celui de type TSV) occupera une surface et présentera une résistance et une capacité dont il faudra tenir compte pour évaluer les valeurs totales de surface, vitesse et consommation de puissance. Pour ce faire, nous avons considéré des liens verticaux de type MIV plutôt que TSV car les premiers se caractérisent par des valeurs nettement plus faibles de surface, résistance et capacité. A côté de cet avantage, nous avons aussi opté pour une autre amélioration. Celle-ci consiste à ne pas utiliser autant de liens verticaux que de transferts de données entre des IPs se trouvant sur des couches différentes de l'architecture. Les travaux de littérature ont étudié l'impact des architectures 3D partiellement connecté en considérant un nombre fixe de liens verticaux. Les résultats ont souligné l'importance de maintenir l'équilibre entre les performances et les coûts de fabrication des liens verticaux pour la conception d'un NoC 3D. Nous nous somme intéresser dans les travaux de cette thèse à l'optimisation de nombre de liens verticaux ainsi qu'a leur placement. Le but est donc de réduire autant que possible la surface occupée par ces liens verticaux (et les parasites électriques qu'ils engendrent) de manière à ne pas

enfreindre les contraintes portées sur la vitesse et la consommation de puissance. A cet effet, nous avons développé plusieurs heuristiques que nous pourrons avantageusement utiliser pour une application donnée (les exécuter toutes, puis retenir celle donnant la meilleure solution à ce problème d'optimisation). Les résultats illustrent que notre méthode permet d'importantes économies en coûts et d'améliorer le rendement avec une faible pénalité en performance

Le chapitre suivant sera consacré à l'étude de la deuxième partie de notre travail de thèse qui consiste à l'optimisation du nombre de dispositifs utilisés pour générer les fréquences d'horloge (appelé PLLs) requises pour le fonctionnement des différents composants du système sur puce. Ceci permettant une réduction de surface et consommation de puissance.

CHAPITRE 4
OPTIMISATION DU NOMBRE DE PLLs (Phase Locked Loop) POUR LA GENERATION DES HORLOGES DANS UN RESEAU SUR PUCE

4.1 Introduction

Les réseaux sur puce permettent aux concepteurs d'intégrer de plus en plus d'unités de traitement dans un système sur puce [6]. Un système sur puce se compose aujourd'hui de plusieurs composants fonctionnant à différentes fréquences d'horloges [101]. Cela pose donc des problèmes supplémentaires pour la génération d'horloges dans la conception de ces systèmes [102]. Le dispositif permettant la génération d'horloge est appelé PLL (Phase Locked Loop pour boucle à verrouillage de phase) qui multiplie la fréquence d'oscillateur de cristal (horloge externe) afin de créer la fréquence nécessaire au fonctionnement de chaque composant (horloge interne). Par exemple, la PLL du processeur permet la génération d'une fréquence multiple à partir du signal d'horloge de la carte mère. En raison de l'augmentation du nombre de composants dans les systèmes sur puce, l'utilisation de PLL dédiés pour la génération d'horloge pour chaque composant devient inefficace en termes de puissance et de coût [102]. Afin de surmonter ces problèmes, la génération des différentes fréquences nécessaires pour les composants du système avec un nombre minimum de PLLs est envisagée.

Ce chapitre traitera le problème d'optimisation du nombre de PLLs pour la génération des horloges des composants d'un réseau sur puce. Dans un premier temps, nous discuterons, à travers une technique relativement récente [102], des détails de la problématique de réduction du nombre de PLLs. Par la suite, nous présenterons notre propre méthodologie qui sera comparée à celle citée dans [102].

4.2 Problématique de la réduction de PLLs pour la génération d'horloges

Les systèmes sur puce actuels incluent un nombre important de composants intégrés. Dans la pratique, ces composants ne fonctionnent pas forcément à la même fréquence. Concevoir autant de PLLs que de fréquences différentes devient un vrai dilemme. En effet, chacune de ces PLLs occupera une surface non négligeable dont il faudra tenir compte dans l'évaluation de la surface totale de l'architecture implémentant l'application donnée. Plus problématique encore, une PLL est un circuit consommant une puissance importante. A titre d'exemple, une PLL générant une fréquence de 6 GHz consomme pas moins de 11 mW. Ceci ne va pas sans dire que c'est un vrai problème pour concevoir des systèmes où la consommation de puissance est un paramètre critique (systèmes portables, embarqués, etc.). Il devient alors primordial de se limiter à un nombre réduit de PLLs permettant, toutefois, de générer toutes les fréquences d'horloge requises par les composants du système. Ceci nous amène, comme nous le verrons ultérieurement, à faire face à un problème d'optimisation combinatoire. Celui-ci, pour le résoudre en un temps CPU raisonnable, doit être abordé moyennant le développement d'un algorithme (à base d'heuristique ou de méta-heuristique) approprié pour espérer converger vers une solution de qualité (pré-optimale, voire optimale dans certains cas). Avant de présenter notre propre méthodologie, nous décrivons une technique relativement récente [102] qui a servi pour la comparaison de nos résultats. Pour diminuer le coût et la consommation de puissance de ces horloges (fabriquées à base de PLL), Reuben J et al [102] offrent une nouvelle méthodologie.

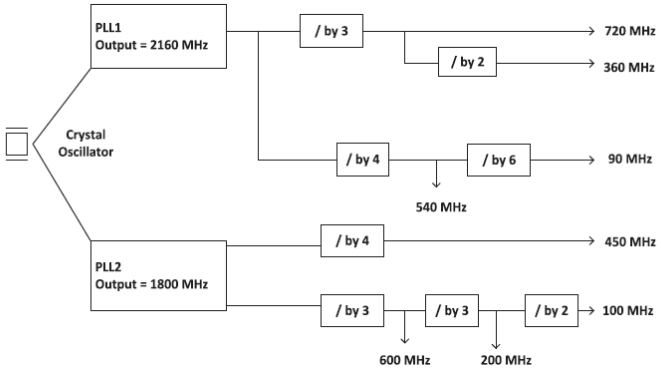


Figure 4.1 : Circuit générateur d'horloges [102]

L'idée générale est de faire partager une PLL par un groupe de composants au lieu de concevoir une PLL pour chacun de ces composants. Un groupe de composants partagent une même PLL si la fréquence de sortie fournie par cette PLL est multiple de leurs fréquences. Le système de génération d'horloge produit les différentes fréquences des composants à partir des PLLs comme multiplicateurs de fréquences suivies par les diviseurs d'horloges [103]. La figure 4.1 [102] illustre un circuit générateur d'horloges : il montre comment les 8 fréquences désirées sont générées à partir de 2 PLLs seulement (et non pas 8) et de diviseurs. Notons qu'un diviseur consomme nettement moins de puissance qu'une PLL.

La fréquence de la PLL est le plus petit multiple commun PPMC (*Least Common Multiple LCM*) des fréquences des composants. Les fréquences individuelles des composants seront ensuite dérivées à partir de la fréquence PPMC par des diviseurs de fréquences d'horloge.

L'exemple qui vient d'être cité illustre bien la réduction de PLLs pour la génération de fréquences d'horloge souhaitées. Dans le prochain paragraphe, une description du problème posé sera donnée de manière plus formelle.

4.3 Formulation de la problématique

L'approche conventionnelle nécessite n PLLs pour la génération de n fréquences uniques requises pour les composants d'un système sur puce [104, 105]. Comme dit auparavant, l'utilisation d'un grand nombre de PLLs conduit à une plus grande consommation de puissance et de surface [102]. Partager une PLL par un ensemble de composants permet de minimiser la surface (minimum matériel ou PLL) ainsi que la consommation de puissance. Reuben J et al [102] proposent le partage d'une PLL par un ensemble de composants qui ont un plus petit multiple commun en cherchant la fréquence de fonctionnement optimale FFO (*Optimum Operating Frequency OOF*) en-dessous de la fréquence de fonctionnement maximale du composant.

Le PPMC est basé sur les fréquences de fonctionnement des composants. Notons que chaque composant fonctionne dans un intervalle de fréquences [f_{min}, f_{max}]. Pour pouvoir générer une PLL commune à un ensemble de composants, il faut trouver une fréquence de fonctionnement optimale de chaque composant incluse dans l'intervalle des fréquences de fonctionnement de ce composant tout en pouvant être générée à partir de la même fréquence de PLL (PPMC des FFOs), et ce, en utilisant des diviseurs.

En outre, les PLLs jusqu'à 2,5 GHz seulement sont généralement disponibles en tant que composants industriels [102, 106]. Par conséquent, le PPMC est limité à 2,5 GHz. Ainsi, nous pouvons présenter la formulation mathématique du problème de la manière suivante:

Soit C l'ensemble des composants de l'application dont la cardinalité est nbIP et L l'ensemble des PLLs dont la cardinalité est nbPLL. La fonction associate**()** qui consiste à associer à chaque Composant IPi une PLLj dont la fréquence égale à FQ_j est définie comme suit :

$$\forall \text{IPi} \in \text{C}, \exists \text{PLLj} \in \text{L tel que FQi} = \text{FQj/DIVk} : \text{associate(IPi)} = \text{PLLj} \quad \text{..... (1)}$$

Tel que FQi est la fréquence nécessaire pour le fonctionnement du composant, DIV_k est le diviseur d'horloge permettant la génération de la fréquence du composant à partir de la fréquence de la PLL. Notre fonction objective peut être formulée ainsi :

min (nbPLL) tel que nbPLL ϵ [1, nbIP] ..... (2)

Sous contrainte : $\forall \text{PLLj} \in \text{L}$: FQj ≤ 2,5 GHz … ..... (3)

Trouver la solution optimale nécessite de vérifier toutes les combinaisons de solutions possibles. L'espace de recherche de solution augmente considérablement avec la taille du SoC (nombre de composants), nécessitant des années de calcul. Considérons, par exemple, un système avec 40 composants dont chacun peut fonctionner dans une plage de 10 fréquences [f1, f2, f3, ..., f10]: le nombre de combinaisons possibles serait de 10^{40}. En supposant qu'une nanoseconde est nécessaire pour évaluer une solution, plus d'un siècle (10^{40}

nanoseconde = 3,17 × 10^{23} ans [107]) serait nécessaire pour déterminer la solution optimale. Il est clair que ce problème n'est pas de complexité algorithmique polynomiale. Pour espérer converger vers une solution de qualité (pré-optimale, voire optimale dans certains cas) en un temps CPU raisonnable, un algorithme à base d'heuristique ou de méta-heuristique est requis.

À notre connaissance, Reuben J et al [102] proposent le premier et l'unique travail considérant la minimisation du nombre de PLLs pour le problème de génération de fréquences d'horloge. Ils ont développé un algorithme basé sur la technique du recuit simulé (métaheuristique) pour obtenir le plus petit multiple commun (PPMC) qui minimise le nombre de PLLs en trouvant une fréquence de fonctionnement optimale (FFO) de chaque composant. Celle-ci peut être jusqu'à 10% inférieure à la fréquence de fonctionnement maximale de ce composant. Les FFOs sont choisis de telle sorte que le PPMC des FFOs de tous les composants soit minimisé. Leur méthode a permis de réduire environ 50% de la consommation de puissance du circuit générateur d'horloges.

Les méthodes approchées (heuristiques ou métaheuristiques) ne permettent pas forcement de trouver la solution optimale. Elles guident plutôt le processus de recherche de solution afin d'essayer de trouver des solutions de qualités dans un temps CPU raisonnable. Nous proposons une nouvelle approche basée sur des heuristiques spécifiques à cette problématique. L'expérimentation (section 4.5) montre l'efficacité de l'approche proposée.

Afin d'évaluer et comparer la consommation de puissance du générateur d'horloges, nous utilisons les mêmes modèles utilisées dans [102, 108]. La consommation de puissance d'une PLL de fréquence f (en MHZ) est décrite ainsi :
Consommation Puissance= 2+ (1.5*f)/1000 ……. (4)
Pour le diviseur d'horloge, la consommation de puissance du diviseur [102, 103] est : Consommation Puissance=f*1.8*nombre de diviseurs/45000 ……. (5)

Après avoir décrit la problématique à travers le travail de [102], nous présentons ci-après notre méthode de réduction de PLLs.

4.4 Notre contribution [109]

Les données d'entrée sont l'ensemble des intervalles de fréquences des composants du système. En sortie, doivent être générés le nombre minimal de PLLs, l'ensemble des composants reliés à chaque PLL ainsi que leurs diviseurs d'horloges. On peut distinguer 3 cas (figure 4.2) :

Cas 1 : Tous les intervalles de fréquences se chevauchent. On peut prendre la plus petite fréquence commune de tous les intervalles comme fréquence de la PLL (seulement une seule PLL est nécessaire pour tout le circuit).

Cas 2 : Quelques intervalles de fréquences se chevauchent. Dans ce cas, nous groupons les intervalles qui se chevauchent pour former un nouvel intervalle de fréquences communes. Cependant, on peut avoir plusieurs regroupements possibles : par exemple, l'intervalle 1 dans la figure 4.2 (b) peut être regroupé avec l'intervalle 2 ou l'intervalle 3 (l'intervalle 2 ne se chevauche pas avec l'intervalle 3, donc ils ne peuvent pas être regroupés). Pour minimiser le nombre de PLLs, nous devons minimiser le nombre d'intervalles. Ce problème d'optimisation est NP-difficile. Nous utilisons l'heuristique des graphes colorées [110] afin de déterminer ces groupes. Chaque nœud du graphe est attribué à un intervalle de fréquences d'un composant. Ensuite, 2 nœuds sont connectés si et seulement si leurs intervalles correspondants ne se chevauchent pas. La coloration du graphe vise à trouver le nombre minimal de couleurs, réduisant ainsi le nombre d'intervalles de fréquence: les nœuds qui ont reçu la même couleur partagent une fréquence commune. Ainsi, les composants correspondant aux nœuds ayant la même couleur peuvent partager la même PLL générant la fréquence commune. Dans le cas où la technique de coloration de graphe produit de nombreux intervalles de fréquence sans chevauchement, les algorithmes du cas 3 peuvent être davantage utilisés pour minimiser le nombre de PLLs.

Cas 3 : Tous les intervalles de fréquences ne se chevauchent pas. Quand les fréquences de deux ou plusieurs composants sont multiples les uns des autres, la plus élevée des fréquences peut être générée a partir de la PLL et les autres peuvent être dérivées par des diviseurs d'horloges. Une heuristique à base de clusters est proposée pour ce troisième cas.

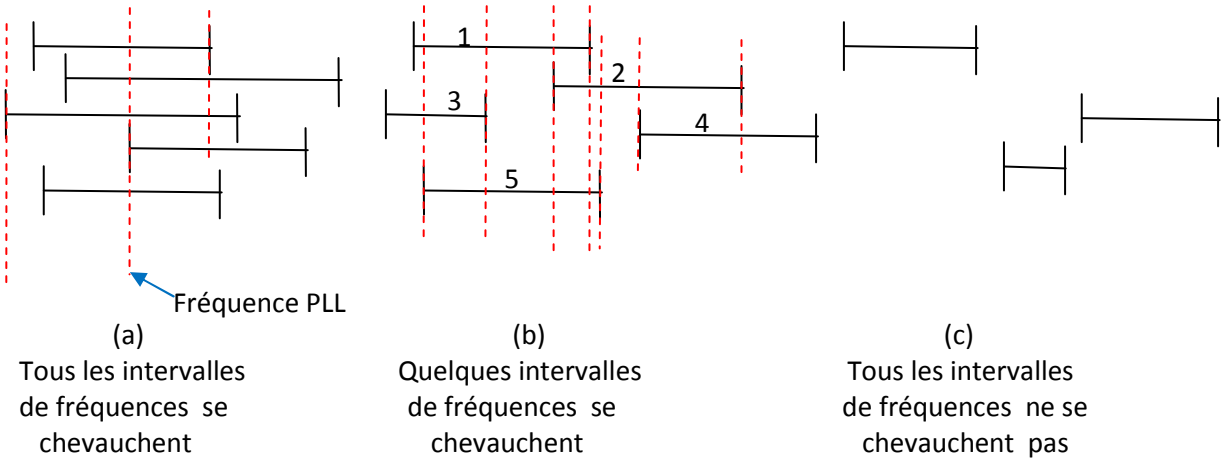


Figure 4.2: Intervalles de fréquences

La figure 4.3 illustre notre framework pour l'optimisation du nombre de PLLs.

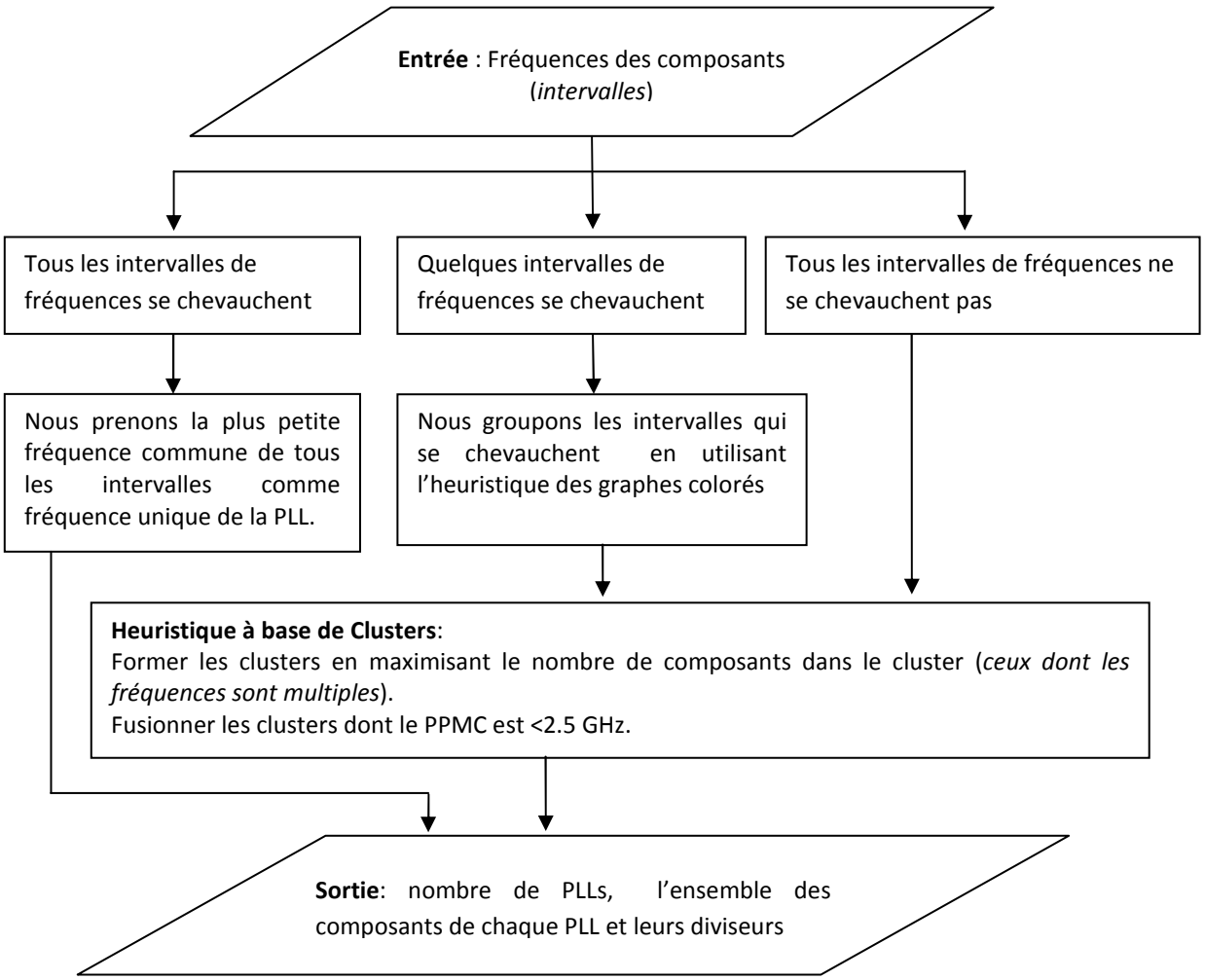


Figure 4.3**:** Notre framework d'optimisation du nombre de PLLs

L'Heuristique à base de clusters :

Nous procédons en deux étapes : dans la première, nous formons les clusters en maximisant le nombre de composants dans le cluster : composants dont les fréquences sont multiples (en cherchant la fréquence de fonctionnement optimale FFO de chaque composant). Ensuite, dans la deuxième étape, nous fusionnons les clusters obtenus en utilisant le PPMC : en intégrant dans un même cluster ceux dont le PPMC est inférieur à 2.5 GHz (à l'échelle industrielle, la limite est à 2.5 GHz). Les deux algorithmes suivants détaillent ces deux étapes.

Algorithme 1 :

1) Sélectionner l'intervalle qui inclut les fréquences les plus élevées (à partir de l'ensemble des intervalles de fréquences des composants)
2) Former une liste dont les éléments sont les fréquences de l'intervalle trouvé dans (1)
3) Pour chaque élément de la liste formée en (2), créer une sous-liste de fréquences de tous les autres composants dont les fréquences sont divisibles par la fréquence de l'élément associé dans la liste (Par exemple 7 et 14 dans la figure 4.4)
4) Pour chaque élément de la liste, calculer le nombre de composants dans la sous-liste (Notons que chaque fréquence de la liste est un multiple des fréquences qui se trouvent dans les sous-listes associées)
5) Pour former un cluster, choisir, parmi toutes les sous-listes, celui qui contient le plus grand nombre de fréquences (le plus grand nombre de composants).
6) Si tous les composants appartiennent à un cluster, arrêter
Sinon, pour tous les composants qui n'appartiennent à aucun cluster, refaire les étapes 1 à 6.

Exemple 1: Considérons l'exemple donné dans [102]: 3 composants qui peuvent fonctionner dans la plage de fréquences de [6, 8], [14, 16] et [3, 5] respectivement. Notre liste sera:

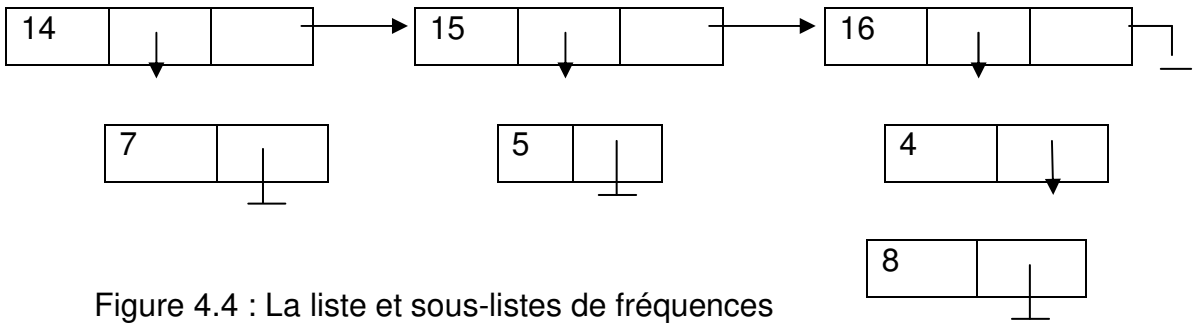


Figure 4.4 : La liste et sous-listes de fréquences

Chaque cluster contient uniquement les composants dont la fréquence la plus élevée est multiple des autres.

Dans cet exemple, la 3ème sous-liste (3ème cluster) contient tous les composants. Donc, une seule PLL de fréquence 16 MHz et 2 diviseurs sont retenus pour obtenir les fréquences de 4 MHz et 8 MHz.

Exemple 2 : Considérons le cas décrit par le tableau 4.1

Tableau 4.1: Exemple avec 6 composants

Composant #	Intervalle de fréquence	Fréquence	PLL	Diviser par
1	[5, 17]	11	#1	6
2	[18, 22]	22	#1	3
3	[27, 36]	33	#1	2
4	[40, 51]	40	#3	-
5	[55, 58]	55	#2	-
6	[63, 66]	66	#1	-

Le composant 6 a la fréquence la plus élevée, 66 MHz (qui est un multiple de 11, 22 et 33 qui sont les fréquences des composants 1, 2 et 3, respectivement). Ainsi, pour cet exemple, nous obtenons 3 PLLs: la première PLL partagée par les composants 1, 2, 3 et 6, la deuxième PLL pour le 4ème composant et la troisième PLL pour le cinquième.

Avec cet algorithme, les fréquences des clusters (PLLs) seront égales aux fréquences maximales des composants de chaque cluster (pour les clusters avec un seul composant, nous prenons la plus petite fréquence). Ces fréquences correspondent aux FFOs (fréquences de fonctionnements optimales) qui permettent une première minimisation du nombre de PLLs tout en minimisant leurs consommations de puissance.

Sachant que la fréquence maximale d'une PLL est limitée à 2,5 GHz (comme expliqué dans la section 3), et qui peut être plus élevée que les fréquences des composants (qui peuvent être en MHz), nous pouvons alors minimiser le nombre de PLLs en fusionnant les clusters (PLLs) en utilisant le PPMC à condition qu'il soit inférieur à 2,5 GHz. La technique est décrite dans l'algorithme 2. D'abord nous regroupons les clusters dans un seul ensemble dont la fréquence est égale au

PPMC. Si le PPMC est réalisable (≤ 2,5 GHz) alors une seul PLL dont la fréquence est égale au PPMC est nécessaire pour tout le circuit. Sinon, l'algorithme 2 permettra de définir le nombre minimum de PLLs satisfaisant la contrainte.

Algorithme 2

1) Subdiviser l'ensemble des clusters en deux sous-ensembles de manière aléatoire (nous créons diverses combinaisons de sous-ensembles afin de rechercher toutes les partitions possibles, c'est-à-dire que l'ensemble est réorganisé de manière aléatoire).
2) Calculer le PPMC de chaque sous-ensemble
3) Si la solution n'est pas réalisable et que le nombre d'éléments des sous-ensembles est supérieur à 2, effectuer un appel récursif à l'algorithme pour subdiviser l'ensemble.
Sinon, Retourner le nombre de PLLs avec les composants associés et leurs diviseurs.

4.5 Expérimentations

Dans ce paragraphe, nous présentons les résultats expérimentaux obtenus par l'exécution de nos heuristiques. Afin d'évaluer les résultats, nous avons effectué des expériences sur différents benchmarks (pseudo-aléatoires) avec des composants dont la fréquence de fonctionnement maximale varie de 40 MHz à 840 MHz et la fréquence minimale de chaque composant est jusqu'à 10% inférieure à la fréquence de fonctionnement maximale selon le benchmark proposé dans [102]. Ceci, afin de faire la comparaison des résultats obtenus avec celle citée dans la littérature [102].

Nous rappelons que dans le cas où tous les intervalles de fréquences se chevauchent (cas 1 de la figure 4.2 (a)), seulement la recherche de la plus petite fréquence commune est nécessaire. Dans le cas où il existe deux ou plusieurs intervalles de fréquences qui se chevauchent (figure 4.2 (b)), l'heuristique des graphes colorés est d'abord utilisée pour regrouper les composants qui partagent les mêmes fréquences. Et puis, dans le dernier cas (cas 3 : figure 4.2 (c)), nous traitons les intervalles de fréquences qui ne se chevauchent pas avec l'heuristique à base de clusters.

Seulement les résultats du troisième cas seront comparés avec ceux de [102] puisque le cas 1 et le cas 2 n'ont pas été traités par eux (notre travail est le premier qui considère tous les cas possibles).

Cas 1: Tous les intervalles de fréquence se chevauchent. Un seul benchmark est présenté ici. Considérons les 4 intervalles de fréquence suivants: [180, 200], [189, 210], [194, 215], [198, 220]. Dans ce cas, tous les intervalles de fréquence se chevauchent. Alors, une seule PLL est nécessaire pour ce circuit. La fréquence de la PLL est de 198 MHz (la fréquence commune minimale) avec une consommation d'énergie de 2.30 mW.

Cas 2 : Quelques intervalles de fréquence se chevauchent. Quatre benchmarks de 27 composants chacun sont simulés. La 2ème colonne du tableau 4.2 présente les résultats du graphe coloré : le nombre de couleurs présente les intervalles de fréquences qui ne se chevauchent pas (les intervalles qui se chevauchement sont regroupés et forment la même couleur). Ensuite, une PLL unique peut être utilisée pour chaque groupe (couleur). Si le nombre de couleurs est supérieur à 1, l'heuristique à base de clusters traite le cas des intervalles de fréquences qui ne se chevauchent pas (cas 3).

Tableau 4.2 : Résultats de l'heuristique du graphe coloré

Benchmarks	Graphe coloré (nombre de couleurs)
1	11
2	10
3	10
4	7

Cas 3 : Tous les intervalles de fréquences ne se chevauchent pas. 12 benchmarks (benchmarks 9 à 12 du cas 2) sont utilisés et les résultats de la première étape de l'heuristique à base de clusters sont résumés dans le tableau 4.3 suivant :

Tableau 4.3 : Résultats expérimentaux de l'algorithme 1 (étape 1 de l'heuristique à base de clusters)

	Benchmark		**Heuristique à base de clusters**	
	#composants	#couleurs	Nombre de PLLs	Comsommation de puissance (mW)
1	6		3	6.24
2	8		4	11.26
3	8		4	11.47
4	20		11	32.75
5	20		10	29.38
6	20		11	32.04
7	20		11	32.12
8	40		21	61.64
9	27	11	6	16.82
10	27	10	5	13.32
11	27	10	5	13.32
12	27	7	5	12.26

Le nombre de PLLs est réduit en moyenne à 50%. En général, cet algorithme (étape 1) permet de générer des faibles fréquences pour les PLLs (fréquence PLL = fréquence de composant). Ainsi, toutes les fréquences de cluster (PLLs) sont au maximum égales à celles des composants (<840 MHz). Comme expliqué au paragraphe 4.4, la fréquence maximale d'une PLL ne doit pas dépasser 2,5 GHz. Par conséquent, nous pouvons encore réduire le nombre de PLLs en nous basant sur le calcul du PPMC des clusters (PLLs) et en intégrant celles dont le PPMC est inférieur à 2.5 GHz. Le tableau 4.4 illustre cette amélioration par la comparaison des résultats avant et après l'exécution de l'algorithme 2 (l'étape 2 permettant l'utilisation des PPMCs).

Tableau 4.4 : Comparaison des résultats avant et après l'utilisation de l'algorithme 2 sur le benchmark 3

Composant	Intervalle de fréquences (MHz)	Fréquence du composant (MHz)	Fréquence de la PLL (MHz)		Numéro du cluster (PLL)		Division par	
			Avant	Après	Avant	Après	Avant	Après
1	[90, 98]	90	720	2160	1	1	8	24
2	[100, 190]	180	720	2160	1	1	4	12
3	[200, 300]	240	720	2160	1	1	3	9
4	[360, 369]	360	720	2160	1	1	2	6

5	[450, 500]	450	450	1800	4	2		4
6	[540, 580]	540	540	2160	3	1		4
7	[600, 641]	600	600	1800	2	2		3
8	[720, 729]	720	720	2160	1	1		3

PPMC (composant #5, composant #7) = PPMC (450, 600) = 1800
PPMC (composant #6, composant #8) = PPMC (540, 720) = 2160

Ces PPMC sont inférieurs à 2560 MHz (2,5 GHz). Par conséquent, les composants seront fusionnés dans un même cluster. Ce qui permet la réduction du nombre de PLLs nécessaires pour le circuit.

En raison du manque de benchmarks dans la littérature (un seul benchmark de 27 composants a été utilisé dans [102]) et afin de pouvoir comparer les résultats des différentes heuristiques nous avons ré-implémenté la méthode proposée dans [102]. L'algorithme 3 et l'algorithme 4 décrivent la métaheuristique du recuit simulé et la technique de partitionnement adopté par [102] respectivement.

Algorithme 3 : Recuit simulé [102]

Entrées : les intervalles de fréquences des composants
Sorties : S_{best} (FFOs - fréquences de fonctionnements optimaux des composants)

$S = \{f_{max}\}$, $S_m = \{f_{min}\}$, $T = T_{max}$, $S_{best} = S$, $E_{best} = LCM(S)$,
Tantque($T > T_{min}$)
Faire
  S_{new}=GenererSolutionVoisine();
  $E_{new} = LCM(S_{new})$;
  $dE = (E_{new} - E_{best})/E_{best}$;
  p=exp(-dE/T) ;
  Si (dE≤0) ou(dE>0 et p>random(0,1))
    $S_{best} = S_{new}$;
    $E_{best} = E_{new}$;
  Finsi ;
 T=T* α;
Fait

Algorithme 4 : Partitionnement [102]

Entrées : S_{OOF} (l'ensemble des fréquences de fonctionnements optimaux des composants)
Sorties : Nombre de PLLs

S_1=Premier (n/2) éléments de S_{OOF} ;
S_2=Reste des éléments de S_{OOF} ;
$lcm_1 = LCM(S_1)$; $lcm_2 = LCM(S_2)$; $lcm_{sum} = lcm_1 + lcm_2$;
k=1;

Pour i=0 à 2
Faire
Tantque (k≤10000)
Faire
$NewS_{OOF}$=Réarrangement_aléatoire(S_{OOF}) ;
$NewS_1$=Premier {(n/2)-i} éléments de $NewS_{OOF}$;
$NewS_2$=Reste des éléments de $NewS_{OOF}$;
lcm'_1=LCM($NewS_1$) ; lcm'_2=LCM($NewS_2$) ; lcm'_{sum}=lcm'_1+lcm'_2;
Si($lcm'_{sum<}$ lcm_{sum})
Sauvgarder($NewS_1$, $NewS_2$, lcm'_1, lcm'_2) ;
lcm_{sum}= lcm'_{sum} ;
K=k+1 ;
Fait
Si (lcm'_1 ou lcm'_2) est non réalisable alors appeler récursivement l'algorithme de partitionnement;
Fait

Pour l'expérimentation nous avons considéré le paramétrage suivant pour le recuit simulé $Tmin=10^{-13}$, $Tmax=10^{8}$ et $\alpha=0.09$. Les résultats sont exposés dans le tableau 4.5. Nous représentons le nombre et la consommation des PLLs avant et après réduction en appliquant les différentes heuristiques.

Tableau 4.5 : Comparaison des résultats des heuristiques

	Sans réduction		**Travaux de [102]**			**Nos travaux**		
Benchmark	#PLL	Puissance	#PLL	Puissance (mW)	Temps CPU (seconde)	#PLL	Puissance (mW)	Temps CPU (seconde)
1	8	21.11	2	9.94	52.67	2	9.94	0
2	20	54.29	18	49.74	342.75	11	32.12	0
3	20	53.39	18	51.14	355.69	8	30.00	0
4	27	69.22	10	38.04	310.7	8	28.35	0
5	40	106.75	36	101.27	564.3	18	59.85	0
Réduction	**-**	**-**	**31%**	**23.15%**	**-**	**61%**	**48.13%**	**-**

Nous avons obtenus une réduction jusqu'à 61% en moyenne par rapport à la méthode conventionnelle (sans réduction) pour le nombre de PLLs (gain en surface) avec 48.13% de la réduction en consommation de puissance. Soit une réduction de 30% et 25% en nombre et consommation de puissance des PLLs par rapport à la littérature avec beaucoup moins de temps CPU.

Nous rappelons que les auteurs dans [102] ne considèrent pas les intervalles de fréquences qui se chevauchent. Donc, leur PPMC obtenu pourrait être très élevé. D'autre part, en utilisant l'heuristique du graphe coloré, il nous a été possible d'obtenir les fréquences communes des intervalles. Les benchmarks de 9 à 12 du tableau 4.3 (de 27 composants chacun) sont des exemples de ce cas. En

considérant tous les cas possibles (intervalles sans ou avec chevauchement), les résultats expérimentaux montrent une réduction de 80% du nombre de PLLs (5 PLLs dans le benchmark 12) et une réduction de 80% de la consommation de puissance (de 64,48 à 12,27 mW dans le benchmark 12).

Grâce à la rapidité de nos algorithmes, un espace de solutions plus grand pourrait être efficacement exploré tandis que d'autres méthodes (par exemple [102]) pourraient échouer à produire une solution intéressante en un temps CPU raisonnable.

4.6 Conclusion

Ce chapitre nous a permis de présenter une nouvelle manière pour confronter le problème de génération d'horloges pour des composants fonctionnant à différentes fréquences d'horloges au sein du réseau sur puce. L'approche utilise une heuristique des graphes colorés et une heuristique à base de clusters afin d'explorer les intervalles de fréquences des composants et de trouver les fréquences individuelles des composants qui optimisent le coût: nombre de PLLs (donc surface) et consommation de puissance.

Comparant au travail de [102] qui est, à notre connaissance, l'unique travail considérant l'optimisation du nombre de PLLs pour la génération de fréquences d'horloge, nos techniques prennent aussi les cas des intervalles qui se chevauchent (cas 1 et cas 2). Les expériences menées avec des benchmarks générés de manière aléatoire montrent que nos algorithmes permettent des réductions considérables en nombre et consommation de puissance des PLLs par rapport à la littérature et avec beaucoup moins de temps d'exécution.

CONCLUSION

Les travaux réalisés dans le cadre de cette thèse ont permis de montrer l'intérêt que peuvent avoir les réseaux sur puce afin de résoudre le problème de communication dans les systèmes sur puce complexes. Ainsi, cette nouvelle structure d'interconnexion autorise l'échange concurrent des informations entre un grand nombre de composants permettant de répondre aux besoins de nouvelles applications.

Avec la complexité de conception de grands circuits, les concepteurs de circuits électroniques sont amenés à utiliser des outils informatiques adéquats. Cependant, il n'existe pas d'outils d'automatisation de l'ensemble des phases de conception d'un réseau sur puce. C'est pourquoi les outils d'aide à la conception de ces systèmes sont des avenues de recherche très actives. Ces outils de CAO doivent optimiser les circuits au niveau de la performance, de la taille et de la consommation de puissance.

Une activité de recherche intense nous a permis de cerner les problèmes et les stratégies de mapping dans les réseaux sur puce. Ceci nous a permis d'élaborer un framework de comparaison et d'évaluation de ces stratégies. Le framework permet d'abord le partage d'une taxonomie pour qualifier chaque système selon quatre critères (la méthode développée, l'approche choisie, le flot de conception suivi et la topologie visée). Et puis, d'identifier les techniques les plus pertinentes et les principales contraintes dédiées à une famille d'application (à usage général ou application spécifique) selon les objectives visés des concepteurs de réseaux sur puce.

Nous avons par la suite entamé l'aspect optimisation du nombre de liens verticaux et leur placement dans une architecture 3D, celle-ci offrant de meilleures caractéristiques qu'une architecture 2D.

Nos contributions algorithmiques se sont alors basées sur des techniques approchées, des heuristiques spécifiques et métaheuristiques, afin de résoudre des problèmes d'optimisations combinatoires qui prennent en considération des systèmes de grande taille.

Pour ce faire, nous avons développé plusieurs algorithmes d'optimisation différents, afin de mettre en évidence certains aspects de la recherche. Notre démarche consiste à minimiser le nombre de liens verticaux (gain en surface), puis, optimiser leurs placements (gain en délai et consommation de puissance). Donc, nous avons choisi l'approche non Pareto en développant des méthodes scalaires de type e-contraintes afin de minimiser la surface sous les contraintes de délai et consommation de puissance. Les résultats expérimentaux ont prouvé l'intérêt des architectures 3D partiellement connectés verticalement.

Le deuxième volet de ce travail de thèse nous a permis d'appréhender une nouvelle problématique, qui n'a jamais été prise en considération dans le domaine des réseaux sur puce jusqu'à présent. Le problème de génération d'horloges, qui a un impact sur les performances globales du système en termes de surface et de consommation de puissance. La PLL est le dispositif permettant la génération d'horloge par la production de la fréquence nécessaire au fonctionnement de chaque composant. Ces composants ne fonctionnent pas forcément à la même fréquence. Concevoir autant de PLLs que de fréquences différentes devient un vrai dilemme. Nous avons donc abordé le problème d'optimisation du nombre de PLLs. Dans les expérimentations réalisées, nous avons pu observer l'efficacité de notre méthode en la comparant à celle de la littérature.

Néanmoins, plusieurs points restent encore à approfondir. Les premières perspectives de recherche que nous pouvons donner concernent les améliorations de nos algorithmes. Citons :

- Prendre en considération le volume de communication entre les composants dans la fonction de coût de la première heuristique gloutonne permettant le placement d'un nombre minimum de liens verticaux afin de favoriser le placement des liens à côté des composants fortement communicants. Ainsi, dans le calcul du centre de gravité des clusters dans la deuxième heuristique, en mettant un centre de gravité pondéré par le volume de communication.

- Traiter les transferts selon une priorité : commencer l'ajout de liens verticaux (dans la première heuristique) pour les transferts qui ont le plus de volume de communication pendant le placement d'un nombre minimum de liens verticaux.

- Spécifier une autre technique pour varier le nombre de clusters dans la deuxième heuristique.

Nous proposons également d'étudier l'hybridation des deux problématiques, le placement des IPs avec le placement des liens verticaux.

Une autre perspective pour nos travaux est d'explorer d'autres méthodes d'optimisation combinatoire, d'autres heuristiques ou métaheuristiques. Ainsi que considérer les solutions Pareto pour l'optimisation multiobjective.

Comme perspectives pour nos travaux, nous souhaitons aussi valider les outils développés sur une application réelle et estimer les performances de son réseau sur puce par rapport aux deux problématiques traitées dans les travaux de cette thèse (réduction et placement efficace de liens verticaux sur une architecture 3D et optimisation du nombre de PLLs).

APPENDICE A
LISTE DES ABREVIATIONS

ASIC : Application Specific Integrated Circuit

CAO : Conception Assistée par Ordinateur

CDTA : Centre de Développement des Technologies Avancées

CPU : Central Processing Unit

DSP : Digital Signal Processor

FFO : Fréquence de Fonctionnement Optimal

IP : Intellectual Property

LV : Lien Vertical

MIV : Monolithic Inter-tier Via

MPSOC : Multiprocessor System-On-Chip

MWD : Multi-Window Display

NI : Network Interface

NOC : Network On Chip

PIP : Picture In Picture

PLL : Phase Locked Loop

PPMC : Plus Petit Multiple Commun

RAM : Random Access Memory

SOC : System On Chip

TSV : Through Silicon Vias

VLSI : Very-Larg-Scale Integration

VOPD : Video Object Plan Decoder

REFERENCES

1. Guerrier P. et Greiner A., "A generic Architecture for On-chip Packet-Switch Interconnections". Design, Automation and Test, France, (March 2000)
2. Dally W.J. et Towles B., "Route Packets, Not Wires: On-chip Interconnection Networks". Design Automation, USA, (May 2005)
3. Hu J., Deng S.Y. et Marculescu R., "System-Level Point-to-Point Communication Synthesis Using Floorplanning Information". VLSI Design and Design Automation, India, (January 2002)
4. Slavisa J., " Architecture reconfigurable de système embarqué auto-organisé ". Thése de Doctorat 2009.
5. Grecu C., Pande P.P., Ivanov A. et Saleh R., "Structured Interconnect Architecture: A Solution for the Non-Scalability of Bus-based SoCs". ACM Great Lakes symposium on VLSI, USA, (April 2004)
6. Xuan-Tu T., "Méthode de test et conception en vue de test pour les réseaux sur puce asynchrones : application au réseau ANOC ". Thèse Doctorat 2008.
7. Koch-Hofer C, "Modélisation, validation et présynthèse de circuits asynchrones en systemc". Thèse Doctorat 2009.
8. Duato J., Yalalanchili S. et Ni L., "Interconnection Networks". Morgan Kaufmann, 2002.
9. Marescaux T., Bartic A., Verkest D., Vernalde S., et Lauwereins R., "Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs". Field-Programmable Logic and Applications, France, (September 2002)
10. Karim F., Nguyen A., Dey S., "An interconnect architecture for networking systems on chip". IEEE Micro, Vol 22, n° 12, (December 2002), 36-45
11. Karim F., Nguyen A., Dey S. et. Rao R., "On chip communication architecture for OC-768 network processors". Design Automation, USA, (2001)
12. Pande P.P., Grecu C., Ivanov A., et Saleh R., "Design of a switch for network on chip applications". Circuits and Systems, Thailand, (June 2003)
13. Andriahantenaina A., et Greiner A., "Micro-network for SoC Implementation of a 32-port SPIN network". Design, Automation and Test, Germany, (December 2003)
14. Dall'Osso M., Biccari G., Giovannini L., D. Bertozzi et Benini L., "Xpipe: a latency insensitive parameterized network on chip architecture for multiprocessor SoCs". Computer Design, Canada, (December 2012)
15. Rantala V., Lehtonen T., Plosila J., "Network on Chip Routing Algorithms". 2006. TUCS Technical Report No 779, (August 2006)
16. Hu J., Marculescu R., "DyAD : smart routing for networks on chip". Design Automation Conference, USA, (July 2004)
17. Kim J., D. Park, Theocharides T., Vuaykrishnan N. et Das C.R., "A low latency router supporting adaptivity for on chip interconnects". Design Automation Conference, USA, (June 2005)
18. Jie Wu: "A fault-tolerant and deadlock-free routing protocol in 2d meshes based on odd-even turn model". IEEE Transactions on Computers, Vol 52, n° 9 (September 2003), 1154-1169
19. Boppana R.V., Chalasani S.,"Fault-tolerant wormhole routing algorithms for mesh networks". IEEE Transactions on Computers, Vol 44, n° 7 (July

1995), 848-864
20. Chen K.H., et Chiu G.M., " Fault-tolerant routing algorithms for meshs without using virtual channels". Journal of Information Science and Engineering, Vol 14, n°12, (December 1998), 765-783
21. Moussa H., "Architectures de réseaux sur puce pour de codeurs canal multiprocesseurs ". Thèse Doctorat 2009.
22. Kouadri-Mostéfaoui A. M., "Architectures Flexibles pour la Validation et l'Exploration de Réseaux Sur Puce". Thèse Doctorat 2009.
23. Quartana J., "Conception de réseaux de communication sur puce asynchrones : Application aux architectures GALS".Thèse Doctorat 2004.
24. Bjerregaard T., ″The MANGO Clockless Network-on-Chip: Concepts and Implementation″. Thèse Doctorat 2005.
25. Goossens K., Dielissen J., Radulescu A., "AEthereal network on chip: concepts architectures, and implementations", Design & Test of Computers, IEEE , Vol. 22, n°9, (September 2005), 414-421
26. Liang J., Swaminathan S., and Tessier R., "aSOC : A Scalable, Single-Chip Communications Architecture". Parallel Architectures and Compilation Technique, USA, (October 2000)
27. Moraes F., Calazans N., Mello A., Moller L., and Ost L., "HERMES : an Infrastructure for Low Area Overhead Packet-Switching Networks on Chip". Integration, the VLSI Journal, Vol. 38, n° 10 (October 2004), 69–93
28. Bolotin E., Cidon I., Ginosar R., and Kolodny A., "QNOC : QoS Architecture and Design Process for Network on Chip", Journal of Systems Architecture, Vol. 50 n° 2, (February 2004), 105-128
29. Delorme J. and Houzet D., " Une technique de mapping pour les réseaux sur puce 2D", In JNRDM2006 Journée Nationnale du Reséau Doctorale en Microélectronique, (Mai 2006)
30. Zarkesh-Ha P., Bezerra G. B. P., Forrest S., et Moses M., " Hybrid Network on Chip (HNoC): Local Buses with a Global Mesh Architecture ″, System level interconnect prediction, USA, (June 2010)
31. Mahdoum A, "A New Design Methodology of Networks On Chip" IEEE/ASQED (Asian Symposium on Quality Electronic Design), Malaysia, (July 2012)
32. Mahdoum A.,"Combined Heuristics for Synthesis of SoCs with Time and Energy Constraints" Computers & Electrical Engineering, Vol. 38, (2012),1687-1702
33. Souna B.,"Placement optimal ou pré-optimal des composants d'un système sur puce implémenté par une architecture 3D " Thèse de Master 2015 (encadrée par A. Mahdoum)
34. Mahdoum A., Souna B., "An efficient IP placement on 3D architectures", Design of Circuits and Integrated Systems, Spain, (November 2016)
35. Toubaline N., Bennouar D., Mahdoum A., "A Classification and Evaluation Framework for NoC Mapping Strategies", Journal of Circuits, Systems and Computers, Vol. 26, n°2 (February 2017),1-44
36. Sahu P. K., Chattopadhyay S., "A survey on application mapping strategies for network-on-chip design", Journal Systems Architecture Vol. 59, n°1 (January 2013), 60–76
37. Hu J., Marculescu R., "Energy-aware mapping for tile-based NoC architectures under performance constraints", Asia and South Pacific Design Automation Conference, Japan (January 2003), 233–239

38. Lu Z., Xia L., Jantsch A., “Cluster-based simulated annealing for mapping cores onto 2D mesh networks on chip“, IEEE Workshop Design and Diagnostics of Electronic Circuits and System, USA (April 2008), 1–6.
39. Lei T., Kumar S., “A two-step genetic algorithm for mapping task graphs to a network on chip architecture“, Euromicro Symposium on Digital Systems Design, Turkey, (September 2003), 180–187
40. Janidarmian M., Fekr A. R., “A survey of meta-heuristic solution methods for mapping problem in network-on-chips“, Advanced Materials Research, Vol. 403-408, (2012), 3994-4008
41. Jena R. K., Sharma G. K., “A multi-objective evolutionary algorithm based optimization model for network-on-chip synthesis“, International Conference on Information Technology, USA, (April 2007), 977–982.
42. Ascia G., Catania V., Palesi M., “Multi-objective mapping for mesh-based NoC architectures“, International conference on Hardware/software codesign and system synthesis, Sweden, (September 2004), 182-187
43. Ascia G., Catania V., Palesi M., “A multi-objective genetic approach to mapping problem on network-on-chip“, Journal of Universal Computer Science, vol. 12, n°. 4 (April 2006), 370-394
44. Ascia G., Catania V., Palesi M., “Mapping cores on network-on-chip“, International Journal of Computational Intelligence Research, Vol.1, No.2 (2005), 109–126
45. Jena R. K., Mahanti P. K., “Design space exploration of network-on-chip: A system level approach“, International Journal of Computing and ICT Research, Vol. 2, n°. 1, (2008), 17 - 25
46. Benyamina H., Boulet P., Aroui A., Eltar S., Dellal K., “Mapping real time applications on NoC architecture with hybrid multi-objective algorithm“, Intenational Conference on Metaheuristics and Nature Inspired Computing, Tunisia, (October 2010)
47. Jena R. K., “System level approach to NoC design space exploration“, International Journal of Information and Electronics Engineering, vol. 2, n° 3, (Mars 2012), 151-155
48. Chen Y., Xie L., Li J., “An energy-aware heuristic constructive mapping algorithm for network on chip“, International Conference on ASIC, China, (October 2009), 101–104.
49. Tavanpour M., Khademzadeh A. , Janidarmian M., “Chain mapping for mesh based network on chip architecture“, IEICE Electronics Express, Vol. 6, n°11 , (November 2009) 1535–1541
50. Mehran A., Saridi S., Khademzadeh A., Afzali-Kusha A., “SPIRAL: A heuristic mapping algorithm for network on chip“, IEICE Electronics Express, Vol. 4, n°8, (August 2007) 478–484.
51. Janidarmian M., Khademzadeh A., Tavanpour M., “Onyx: A new heuristic bandwidth- constrained mapping of cores onto tile-based network on chip“, IEICE Electronics Express, Vol. 6, n°1, (January 2009) 1–7
52. Murali S., Micheli G. D., “Bandwidth-constrained mapping of cores onto noc architectures“, Design, Automation and Test in Europe Conference, France, (February 2004)
53. Srinivasan K., Chatha K. S., “A technique for low energy mapping and routing in network-on-chip architectures“, International symposium on Low power electronics and design, USA, (August 2005), 387-392
54. Saeidi S., Khademzadeh A., Vardi F., “Crinkle: A heuristic mapping

algorithm for network on chip“, IEICE Electronics Express, Vol. 6, n°12, (December 2009) 1737–1744
55. Moein-Darbari F., Khademzadeh A., Gharooni-Fard G., “CGMAP: A new approach to network-on-chip mapping problem“, IEICE Electronics Express, Vol. 6, n°1, (January 2009) 27–34
56. Fen G., Ning W., “Genetic algorithm based mapping and routing approach for network on chip architectures“, Chinese Journal of Electronics Vol.19, n°1, (January 2010), 91-96
57. Hu J., Marculescu R., “Energy- and performance-aware mapping for regular NoC architectures“, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 24, n°4, (April 2005), 551-562
58. Ost L., Mandelli M., Almeida G.M., Moller L., Indrusiak L. S., Sassatelli G., Benoit P., Glesner M., Robert M., Moraes F., “Power-aware dynamic mapping heuristics for NoC-based MPSoCs using a unified model-based approach“, ACM Transactions on Embedded Computing Systems, Vol. 12, n° 3, (March 2013), 1-22
59. Singh A. K., Srikanthan T., Kumar A., Jigang W., “Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms“, Journal of Systems Architecture**,** Vol. 56, n° 7, (July 2010), 242–255
60. Carvalho E., Calazans N., Moraes F., “Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs“, International Workshop on Rapid System Prototyping, USA, (May 2007), 34-40
61. Al Faruque M. A., Krist R., Henkel J., “ADAM: Run-time agent-based distributed application mapping for on-chip communication“, IEEE Design Automation Conference, USA, (June 2008), 760–765.
62. Carvalho E., Marcon C., Calazans N. and Moraes F., “Evaluation of static and dynamic task mapping algorithms in NoC-based MPSoCs“, International Symposium on System-on-Chip, Finland, (October 2009), 87–90
63. Carvalho da Silva M. V., Nedjah N., de Macedo Mourelle L., “Optimal application mapping on NoC infrastructure using NSGA-II and MicroGA“, International Conference on Intelligent Engineering Systems, Barbados, (April 2009) 83–88.
64. Guangyu C., Feihul L., Son S. W., Kandermir M., “Application mapping for chip multiprocessor“, IEEE/ACM Design Automation Conference, USA, (June 2008), 620–625
65. Ghosal P., Karmakar S., “Diametrical mesh of tree (D2D-MoT) architecture: A novel routing solution for NoC“, ACEEE International journal on Communications, Vol. 03,n°3, (March 2012), 30-34
66. Cung T., Tarzia S., Wade K., “Supervised Floorplanning“,Research report, 2008, https://stevetarzia.com/pubs.php
67. Hu J., Deng Y., Marculescu R., “System-level point-to-point communication synthesis using foorplanning information“, Proc. 7th ASP-DAC and 15th VLSI Design Conference, (2002), 573–579
68. Kim J. G., Kim Y. D., “A linear programming-based algorithm for foorplanning in VLSI design“,IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 22, n°4, (April 2003), 584-592
69. Ogras U. Y., Hu J., Marculescu R., “Key research problems in NoC design: A holistic perspective“, International conference on Hardware/software

codesign and system synthesis, USA, (September 2005), 69-74
70. Murali S., De Micheli G., “SUNMAP: A tool for automatic topology selection and generation for NoCs“, Design Automation Conference, UAS, (July 2004), 914–919.
71. Srinivasan K., Chatha K. S., Konjevod G., “Linear programming based techniques for synthesis of network-on-chip architectures“, IEEE International Conference on Computer Design: VLSI in Computers and Processors, USA, (2004), 422–429
72. Srinivasan K., Chatha K. S., Konjevod G., “Linear programming based techniques for synthesis of network-on-chip architectures“, IEEE Transactions on Very Large Scale Integration (VLSI) Systems. Vol. 14, n°4, (April 2006) 407–420.
73. Jovanovic S., “Architecture reconfigurable de système embarqué auto-organisé“, Thèse de doctorat (2009)
74. Chariete A., Bakhouya M., Gaber J., Wack M., “Une approche de personnalisation des architectures pour réseaux d'interconnexion sur puce“,
Les Journées Doctorales d'Informatique et Réseaux (JDIR) 2011, Belfort, France
75. Srinivasan K., Chatha K., “ISIS: A genetic algorithm based technique for custom on chip interconnection network synthesis“, International Conference on VLSI Design, India, (January 2005), 623–628
76. Chath K. S., Srinivasan K., and Konjevod G., ″Automated Techniques for Synthesis of Application-Specific Network-on-Chip Architectures″. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 27, N°. 8, (August 2008), 1425-1438
77. Leary G., and Chatha K. S., “Automated technique for design of NoC with minimal communication latency”, in *Proc.* 7th IEEE/ACM International Conference on Hardware/Softwere Codesigne System Synthesis, 2009, 471–480
78. Fen Ge, Ning Wu, Xiaolin Qin and Ying Zhang. “Clustering-Based Topology Generation Approach for Application-Specific Network on Chip“, Proceedings of the World Congress on Engineering and Computer Science, (October 2011), San Francisco, USA, 19-21
79. Li X., “Méthodologie de Conception Automatique pour Multiprocesseur sur puce Hétérogène“, Thèse de doctorat, 2009
80. Hur J. Y., Stefanov T., Wong S., Goossens K., “Customisation of on-chip network interconnects and experiments in field-programmable gate arrays“, IET Computers & Digital Techniques, Vol. 6, n°1, (January 2012), 59-68
81. Joyner J. W., Venkatesan R., Zarkesh-Ha P., Davis J. A., Meindl J. D., “Impact of three dimensional architectures on interconnects in gigascale integration“, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 9, n°12, (December 2001), 922-928
82. Pavlidis V. F., Friedman E. G., “Interconnect delay minimization through interlayer via placement in 3-D ICs“, ACM Great Lakes symposium on VLSI, USA, (April 2005), 20-25
83. Joyner J., Zarkesh-Ha P., Meindl J., “A stochastic global net-length distribution for a threedimensional system-on-a-chip (3d-soc) “, IEEE International ASIC/SOC Conference, USA, (September 2001), 147–151

84. Chen Y., Hu J., Ling X., “Topology and mapping co-design for complex communication systems on wireless NoC platforms“, IEEE Conference on Industrial Electronics and Applications, Australia, (June 2013), 1442–1447
85. Elmiligi H., El-Kharashi M. W., Gebali F., “Power consumption of 3D networks-onchips: Modeling and Optimization“, Microprocessors and Microsystems, Vol. 37, n° 10, (October 2013), 530-543
86. Wang J., Li L., Pan H., He S., Zhang R., “Latency-aware mapping for 3D NoC using rank-based multi-objective genetic algorithm“,International Conference on ASIC, China, (October 2011), 413–416
87. Ying H., Heid K., Hollsteiny T., Hofmann K., “A genetic algorithm based optimization method for low vertical link density 3-dimensional networks-on-chip many core systems“, NORCHIP, Denmark, (November 2012), 1–4
88. Sotiriou-Xanthopoulos E., Diamantopoulos D., Siozios K., Economakos G. Soudris D., “A framework for rapid evaluation of heterogeneous 3-D NoC architectures“, Microprocessors and Microsystems, Vol 38, n°6, (June 2014), 292-303
89. Xu T. C., Liljeberg P., Tenhunen H., “A study of Through Silicon Via impact to 3D Network-on- Chip design“,International Conference On Electronics and Information Engineering, Japan, (August 2010), 333-337
90. Xu T. C., Schley G., Liljeberg P., Radetzki M., Plosila J., Tenhunen H., “Optimal placement of vertical connections in 3D Network-on-Chip“, Journal of Systems Architecture, Vol. 59, n°8, (August 2013), 441-454
91. Manna K., Shanmukha V., Teja S., Chattopadhyay S., Sengupta I., “TSV Placement and Core Mapping for 3D Mesh Based Network-on-Chip Design Using Extended Kernighan-Lin Partitioning“, IEEE Computer Society Annual Symposium on VLSI, France, (July 2015)
92. Toubaline N., Bennouar D., Mahdoum A., “Optimal Number and Placement of Vertical Links in 3D Network-On-Chip”, 19th International Conference on Parallel Processing, Italy, (March 2017)
93. Toubaline N., Bennouar D., Mahdoum A., “Cluster-based approach to optimize vertical links for 3D Networks-on-Chip architecture”, manuscript soumis pour publication.
94. Toshiba system catalog, “System-in-Package“, 2004, disponible en ligne: (Avril 2017) http://bbs.hwrf.com.cn/downebd/33213d1215758969-system_in_package_toshiba_1606.pdf
95. Garrou P., Bower C., Ramm P., “Technology and Applications of 3D Integrated Circuits“,Wiley-VCH Verlag GmbH & Co. KGaA, 2012
96. Al Attar S., “Conception et mise au point d'un procédé 3D d'assemblage de puces silicium amincies, empilées et interconnectées par des via électriques traversant latéralement les résines polyméres d'enrobage“, Thèse de doctorat, 2012
97. Chang L., Lim S. K., “A design tradeoff study with monolithic 3D integration“, International Symposium on Quality Electronic Design, USA, (March 2012)
98. Mahdoum A., “Architectural synthesis of networks on chip“, IEEE International Conference on Industrial Electonics and Applications, Australia, (June 2013), 1889–1894.
99. Mahdoum A., “Networks On Chip Design for Real-Time Systems“, System On Chip Conference, USA, (September 2014), 165-170

100. Mahdoum A., “An Efficient Network on Chip Design Flow“, International Conference on Communication Technology, China, (October 2015)
101. Saleh R., Wilton S., Mirabbasi S., Hu A., Greenstreet M., Lemieux G., Pratim Pande P., Grecu C., Ivanov A., “System-on-chip : reuse and integration“,Proceedings of the IEEE, Vol. 94, n°7, (July 2006), 1050-1069
102. Reuben J., Kittur H. M., Shoaib M., ″A novel clock generation algorithm for system-on-chip based on least common multiple″, Computers and Electrical Engineering, Vol. 40, n° 10, (October 2014), 2113–2125
103. Reuben J, Zackriya M., Kittur H. M., ″Low Power, High Speed Hybrid Clock Divider Circuit″, International Conference on Circuits, Power and Computing Technologies, India, (March 2013)
104. Kurd N., Barkarullah J., Dizon R., Fletcher T., Madland P., ″A multigigahertz clocking scheme for the pentium 4 microprocessor″,IEEE Journal of Solid-State Circuits, Vol. 36, n°11, (November 2001), 1647-1653
105. Nose K., Shibayama A., Kodama H., Mizuno M., Edahiro M., Nishi N., ″Deterministic inter-composant synchronization with periodically all-in-phase clocking for low-power multi-composant SoCs″, Solid-state circuits conference, Vol. 1, (2005) 296–599
106. IP datasheets; 2013. <http://www.design-reuse.com>.
107. http://www.convertworld.com
108. Lee J-Y, Park M-J, Min B-H, Kim S, Park M-Y, Yu H-K. ″A 4-GHz all digital PLL with low-power TDC and phase-error compensation″, IEEE Transactions on Circuits and Systems I: Regular Papers, Vol. 59, n° 7, (July 2012), 1706-1719
109. Toubaline N., Bennouar D., Mahdoum A., “An efficient clock generation algorithm for system-on-chip based on least common multiple”, manuscript soumis pour publication.
110. Mahdoum A., Louiz F., Belkouche H., ″FEWERCOLORS: A New Technique for Solving the Graph Coloring Problem″, Design Automation and Test in Europe Conference, France, (March 2002)