République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la Recherche scientifique
Université SAAD DAHLEB BLIDA-1

Filière : Informatique

Spécialité : Informatique

Option : Systèmes d'informatiques et Réseaux

Thème :

Mapping dans les réseaux sur puce 3D dédiés aux applications spécifiques

Réalisé par :

Aissaoui Khaled Reda Ismail
Abdelhak Badreddine

Soutenu devant un jury constitué de :

❖ Mr Ould-Aissa Ahmed	Président
❖ Mme Boutoumi	Examinatrice
❖ Mme Toubaline N.	Promotrice

Année universitaire 2021/2022

Résumé

Le réseau sur puce est un nouveau concept d'interconnexions dans les systèmes sur puce. Cette architecture facilité l'intégration de composants complexes et semble s'adapter à l'évolution des applications. Cependant, comme toute nouvelle technologie, elle requiert des efforts en recherche, en particulier pour l'accélération et la simplification des phases de conception.

La phase de mapping représente une phase centrale lors de la mise en œuvre d'un réseau sur puce. Elle permet de placer les **IPs** sur l'architecture de réseau sur puce.

C'est dans ce cadre que s'insère notre travail. Il consiste à proposer une technique de placement des **IPs** d'une application sur architecture de réseau sur puce **3D** afin de minimiser le coût de communications et la surface (nombre des **TSVs**). Ces nouvelles solutions sont basées sur l'optimisation multi-objective par l'algorithme des colonies de fourmis (**MOACO**).

Mots clés : Réseau sur puce, mapping, application spécifique, topologie, technologie 3D et optimisation par colonies des fourmis.

Abstract

Network-on-Chip is a new concept of interconnection in System-on-Chip. This architecture facilitates the integration of complex components and seems to adapt to the evolution of the application. However, like any new technology, it requires research work, especially to speed up and simplify the design phase.

The mapping phase represents a central phase during the implementation of a network on chip. It allows placing **IPs** on the network-on-chip architecture.

It is within this framework that our work takes place. It consists of proposing a technique for placing the **IPs** of an application on a **3D** network-on-chip architecture in order to minimize the cost of communications and the surface (number of **TSVs**). These new solutions are based on multi-objective optimization by the ant colony algorithm (**MOACO**).

Keywords: Network on chip, mapping, specific application, topology, 3D technology and ant colony optimization.

Remerciements

Au terme de ce mémoire, nous tenons à exprimer tous nos remerciements et notre profonde gratitude avant tout au DIEU de nous avoir donné la force et le courage pour mener à bien ce modeste travail, nous saisissons de cette occasion pour exprimer notre profonde reconnaissance à nos Promotrice Mme. TOUBALINE Nesrine pour leur soutien et leur effort durant toute la durée de ce travail.

Nos remerciements y vont aussi à l'endroit de tous les professeurs du département informatique de l'université Saad Dahleb de Blida 1.

Pour terminer, nous remercions tous ceux qui nous ont aidés à élaborer ce travail de près ou de loin.

Dédicaces

Je dédie ce modeste travail accompagné d'un immense amour :

À celle qui m'a donné la vie, qui m'a arrosé de tendresse et d'espoir à la source d'amour infini
Ma très chère mère

À mon support dans la vie celui qui a été toujours à mes côtés et qui m'a soutenu dès mes débuts
Mon très cher père

À ce qui m'a couvert par son affection et bienveillance et qui m'a toujours aimée et protégée
À la source d'énergie incessible mon frère Marouen.

À ceux avec qui j'ai passé les meilleurs moments de mon cursus :
Amine, Wassim, Ghanou, Kamel, Adem.

À mon binôme Badreddine.

Khaled Reda Ismail

Dédicaces

Je dédie ce modeste travail accompagné d'un immense amour :

À celle qui m'a donné la vie, qui m'a arrosé de tendresse et d'espoir à la source d'amour infini
Ma très chère ***mère***

À mon support dans la vie celui qui a été toujours à mes côtés et qui m'a soutenu dès mes débuts
Mon très cher ***père***

À ceux qui m'a couvert par son affection et bienveillance et qui m'a toujours aimée et protégée
À la source d'énergie incessible mes ***frère***
alaa,abdelhadi mohamed.

À mon binôme Khaled Reda Ismail.

Badreddine

TABLE DES MATIÈRES

Chapitre 2 : L'optimisation combinatoire

Chapitre 3 : Le mapping dans les réseaux sur puce (état de l'art)

Chapitre 4 : Conception de la technique proposée

TABLE DES FIGURES

LISTE DES TABLEAUX

Chapitre 1 : Les réseaux sur puce (Noc)

Chapitre 2 : L'optimisation combinatoire

Chapitre 3 : Le mapping dans les réseaux sur puce (état de l'art)

Chapitre 4 : Conception de la technique proposée

Chapitre 5 : Tests et résultats

LISTE DES ABRÉVIATIONS

A :

ACO : Ant Colony Optimization

ACP : Application Communication Pattern

ASIC : Application Specific Integrated Circuit

B :

B&B : Branch-and-bound

C :

CAD : Computer Aided Design

CMAP : Constructive Mapping Algorithm

D :

DSP : Digital Signal Processing

DOL : Distributed Operation Layer

E :

EP : Evolutionary Programming

ES : Evolution Strategies

F :

FIFO : First In First Out

FLIT : FLow control unIT

FPGA : Field Programmable Gate Arrays

G :

GPP : General Purpose Processor

GP : Genetic Programming

GD : Generational Distance

GI : Greedy Incremental

GP : Goal Programming

GA : Genetic Algorithm

H :

HGLA : Hajela and Lin's Genetic Algorithm

I :

IC : Integrated Circuit

ILP : Integer Linear Programming

IP : Intellectual Property

J :

jMetal : java Metaheuristics algorithms

M :

Micro-GA : Micro Genetic Algorithm

MMT3D : 3D Modied Mesh Topology

MOGA : Multi-Objective Genetic Algorithm

MOACO : Multi-Objective Ant Colony Algorithm

MPSOC : Multi-Processor System On Chip

N :

NBI : Normal Boundary Interaction

NI : Network Interface

NoC : Network On Chip

NSGA : Non-dominated Sorting Genetic Algorithm

NSGAII : Non-dominated Sorting Genetic Algorithm II

O :

OAC : Optimization by Ant Colony

P :

PSO : Particle Swarm Optimization

PMO : Multiobjective Optimization Problem

P-metaheuristics : Population-solution based metaheuristics

PBBB : Pareto-based Branch and Bound

PBNMAP : Pareto-based NMAP

R :

ReNoC : Recongurable NoC

S :

SOC : System On Chip

SNN : The Smart Nearest Neighbor

T :

TSV : Through Silicon Via

TG : Task Graph

V :

VCT : Virtual-Cut-Through

VDD : Supply Voltage Values

VEGA : Vector Evaluator Genetic Algorithm

W :

WS : Weighted Sum

INTRODUCTION GÉNÉRALE

L´évolution de la technologie des circuits intégrés et les performances toujours croissantes des systèmes électroniques rendent les systèmes intégrés sur puce (**SoC**) de plus en plus complexes. Pour répondre aux besoins des nouvelles applications, qui nécessite une augmentation des communications dans les circuits. Il est nécessaire de trouver une solution efficace pour la communication sur puce afin d'améliorer les performances globales des systèmes intégrés.

Dans ce contexte ; les solutions d'interconnexion classiques généralement présentent des limites. Elles ne supportent pas les débits élevés, manquent de flexibilité et elles ne seront pas adaptées pour les systèmes futurs implémentant plusieurs centaines d'unités de traitement.

Afin de surmonter ces problèmes, un nouveau paradigme de communication a été proposé par différents groupes de recherche dans les années **2000**. Ce nouveau paradigme est appelé réseau sur puce. Ce type d'architecture présente de réelles réussites aux niveaux des performances et implémentation. Cette architecture facilite l'intégration de composants complexes et semble s'adapter à L'évolution des applications.

La technologie d'intégration **3D** est une nouvelle façon d'augmenter les performances du système sans mise à l'échelle en raison de plusieurs caractéristiques de l'intégration **3D** (la longueur de fil réduite, un délai d'interconnexion réduit, un nombre accru d'interconnexions entre couches)**[79].**

Les **Noc 3D** peuvent être utilisés pour construire des systèmes hétérogènes qui offrent également un meilleur rendement à faible coût. Ils sont des puces en silicium empilées et connectés par des interconnexions verticales (**TSVs**). La longueur, le retard, la puissance et la consommation d'énergie des interconnexions verticales et horizontales est généralement asymétrique. Les circuits intégrés **3D** réduisent la longueur et le nombre d'interconnexions, ce qui réduit à son tour le retard de fil et augmente la localité spatiale avec des interconnexions verticales (augmentation au niveau de surface) **[81].**

L'utilisation des liens verticaux (**TSV**) entre les couches offrent une bande passante plus élevée et une faible latence **[82]**.

La phase de mapping est une phase centrale dans la conception des **Nocs**, le problème c'est comment trouver un bon placement des IPs sur une architecture Noc pour maximiser les performances.

Le Projet :

Notre travail consiste à implémenter une application qui résoudre le problème de mapping sur un réseau sur puce avec une topologie de maillage **3D** pour réduit le coût de la communication et le cout de surface en utilisant une méta-heuristique appelé **ACO**.

Le plan du document :

Afin de permettre au lecteur de suivre et de comprendre notre démarche, le reste de manuscrit est organisé comme suit

Le premier chapitre

Va traiter les concepts de base des réseaux sur puce et quelques notions et concepts qui leurs sont liés (composants, caractéristiques…etc) ainsi que les différentes structures d'interconnexions.

Le second chapitre introduit les notions liées à l'optimisation. Nous résumons les différentes approches et méthodes multi-objectif. Puis, nous terminons par une description générale de l'algorithme de colonie de fourmis (**ACO**).

Le troisième chapitre présente le problème de mapping lié au réseau sur puce, et aussi toutes les techniques de mapping proposées dans la littérature sont introduites avec leur classification. Le chapitre s'achève par une synthèse.

Le quatrième chapitre détaille la conception et la mise en œuvre de notre solution basée sur l'algorithme **MOACO** ainsi que l'architecture globale proposé.

Le cinquième chapitre présente les résultats des tests effectués avec les méthodes implémentées et les comparaisons réalisés.

Ce mémoire s'achève par une **conclusion générale** et quelques perspectives de notre travail.

Chapitre 1

Les réseaux sur puce (Noc)

1. Introduction

Le domaine des Noc est un sujet de recherche récent. Afin de bien présenter le Noc, dans un premier temps, nous expliquons en quoi les moyens de communication classiques ne répondent plus aux contraintes futures des systèmes sur puce et ce que le réseau sur puce apporte comme solution.

Les interconnexions classiques généralement basées sur des bus partagés présentent des limites. Elles ne supportent pas les débits élevés, manquent de flexibilité et ne passeront pas à l'échelle pour les systèmes futures implémentant plusieurs centaines d'unités de traitement.

Les réseaux sur puce ont été proposés comme une solution prometteuse pour résoudre les problèmes rencontrés au niveau des interconnexions classiques.

2. Les différents types d'interconnexion dans les systèmes sur puce

Le réseau sur puce est une nouvelle solution pour l'interconnexion des systèmes sur puce **(voir l'Introduction)**, Parmi les interconnexions classiques on trouve :

2.1 Point à point

Ce type d'interconnexion offre une bande passante très élevé car chaque deux IP ***(Intellectual Property)*** ont un lien dédié **[1],** mais limite toute flexibilité et extensibilité lors de la conception de systèmes sur puce **[2]**. Le taux d'utilisation des liens est très bas (environ 10% selon une recherche de Dally et al **[3]**) qui

implique un impact négatif sur le coût de la surface. Les systèmes qui comportent peu de composants peuvent utiliser cette topologie pour la communication.

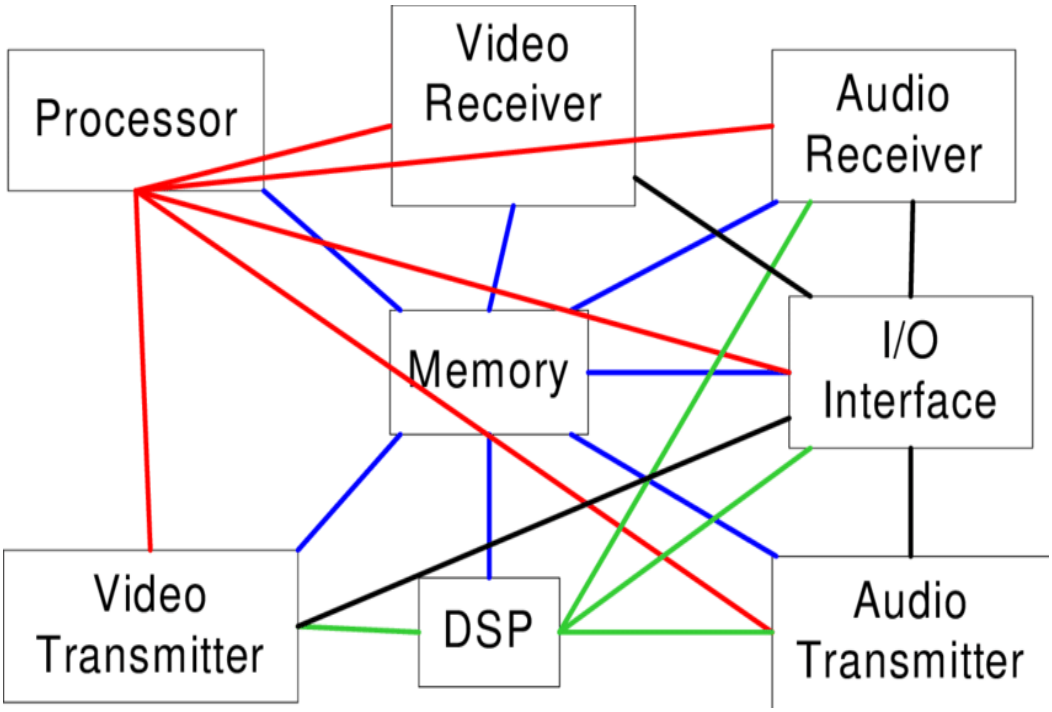

Figure 1 : Soc basée sur la topologie point à point pour la communication

2.2 Bus standard

Un bus standard ou partagé est un ensemble de connexions autour duquel communiquent plusieurs IPs**,** il est composé de lignes transportant les signaux (données et contrôles) entre une source et une cible et d'un arbitre pour désigner les composants des services du bus à un instant donné. Le bus est beaucoup plus flexible et mieux réutilisable comparé aux liaisons point à point. En effet, un IP peut être ajouté, déplacé ou supprimé facilement. Cependant, il ne permet qu'une seule communication à la fois entre les IPs raccordés. La bande passante allouée à chaque IP diminue donc avec le nombre d'IPs communicants. Enfin, du fait de la longueur des fils et de nombre de IPs connectés, la consommation d'énergie devient plus importante et les délais de transmission peuvent alors dépasser la période d'horloge du système **[4]**.

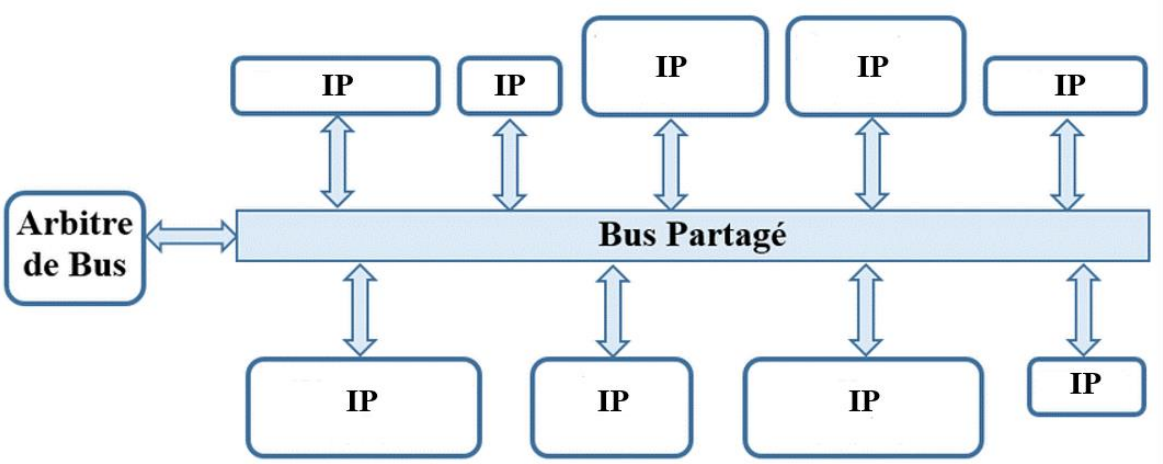

Figure 2 : Soc basée sur la topologie bus partagé pour la communication

2.3 Bus hiérarchique

Afin de surmonter les inconvénients du bus standard énoncés précédemment, la structure de bus hiérarchique a été introduite. Le concept de cette interconnexion consiste à connecter plusieurs bus standards à l'aide de ponts (passerelles).

Les différents segments du bus hiérarchique possèdent des liaisons courtes avec peu d'IPs connectées, offrant une faible capacité sur chacun de ces segments et donc une faible consommation. Cependant, l'accès d'un bus à un autre au travers d'un pont, implique un coût en latence d'où l'importance de bien répartir les différents IPs communicantes pour limiter l'utilisation du pont et ainsi favoriser le fonctionnement parallèle des bus.

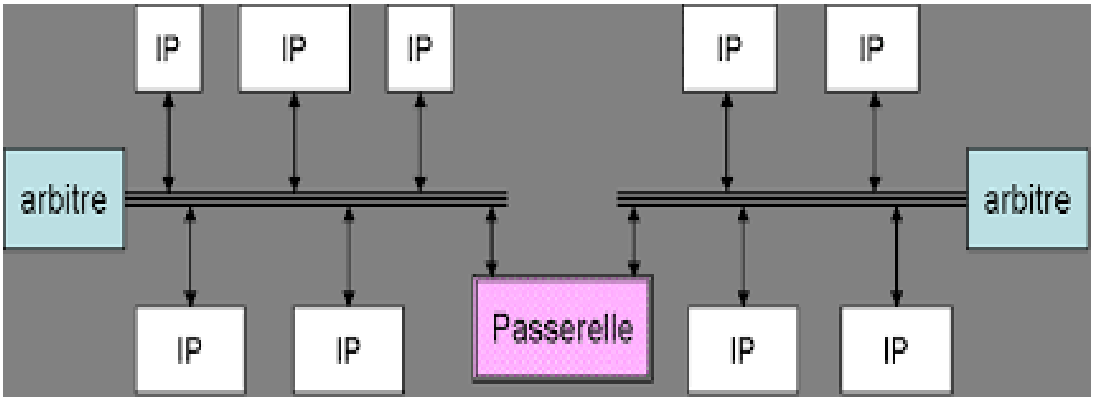

Figure 3 : Bus hiérarchique

2.4 Réseaux sur Puce (Noc)

Les limites des bus (débit maximal, scalabilité) ont conduit, depuis quelques années, à étudier de nouvelles architectures de réseaux intégrés, donnant ainsi naissance vers le début des années 2000 au concept de réseau sur puce (Noc). Ce nouveau type d'interconnexion a été développé pour pouvoir connecter efficacement les différents IPs d'un système mono puce (Soc). L'idée consiste à adapter les solutions des réseaux locaux et des réseaux d'interconnexions de machines parallèles au contexte des Socs, tout en respectant, lors de la phase de conception, les contraintes spécifiques aux circuits intégrés comme la latence, la surface du silicium et la consommation d'énergie.

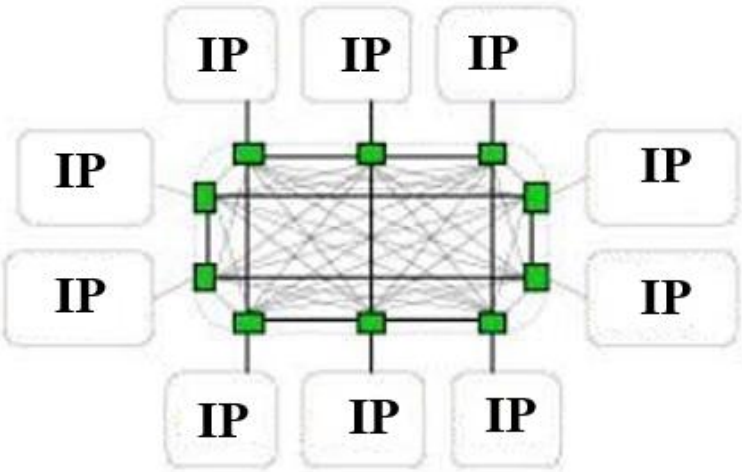

Figure 4 : Exemple d'un réseau sur puce

Une comparaison entre les différentes caractéristiques des types de communication dans les systèmes sur puce **(tableau 1.1 [2, 5])**.

Tableau 1.1 : Comparaison des interconnexions [2, 5]

	Point à point	Bus partagé	Bus hiérarchique	Noc	Noc personnalisé
Parallélisme	Très bon	Très mauvais	Bon	Très bon	Très bon
Réutilisation	Mauvais	Très bon	Très bon	Très bon	Bon
Consommation	Bon	Mauvais	Mauvais	Très bon	Très bon
Scalabilité	Très mauvais	Mauvais	Mauvais	Très bon	Très bon

Les performances globales d'un Soc dépendent fortement du type d'interconnexion utilisé **[6],** parmi les différents moyens de communication, le réseau sur puce est le meilleur. Il est capable d'être étendu contrairement de bus qui partage un même media de transmission pour les IPs connecté ce qui implique la limitation de la bande passante et l'extensibilité du système. Pour la consommation d'énergie le noc est le plus performant car les liens utilisés dans les réseaux sont plus courts. Ainsi le réseau sur puce et les connexions point à point permettent de prendre en charge plusieurs communications simultanément (parallélisme), cependant les liens point à point sont dédiés et ne peuvent être réutilisés. Le concept de routage dans les réseaux sur puce personnalisés aide la réutilisation de ces derniers dans autres applications.

3. Concepts des réseaux sur puce

3.1 Les composants d'un réseau sur puce

Un réseau sur puce est constitué de trois types de composants :

- Les nœuds de routage sont chargés de diriger les données qu'ils reçoivent vers leurs destinataires.
- Les interfaces réseau qui permettent d'accéder au réseau sur puce.
- Les liens de communication assurent le transfert des données entre deux composants du réseau sur puce (interfaces réseau ou nœuds de routage).

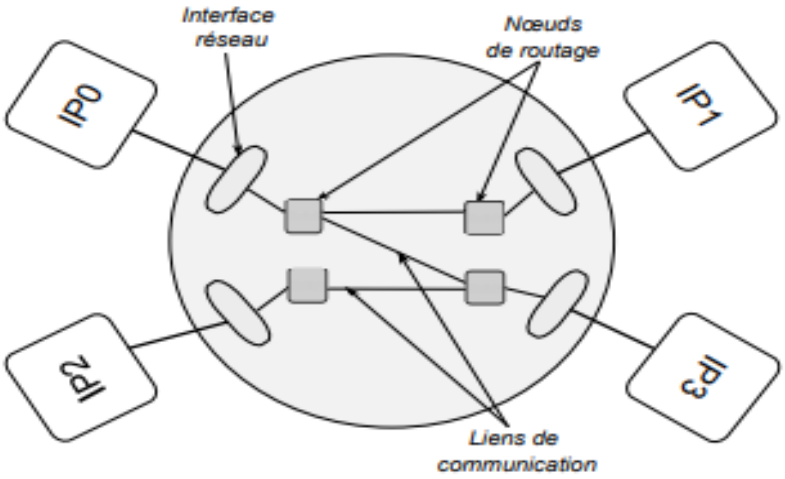

Figure 5 : Structure d'un réseau sur puce

3.1.1 Les liens

Son rôle principal 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 de plus en plus fins alors que les distances qu'ils doivent parcourir ne diminuent pas, ces fils sont donc de plus en plus sensibles aux phénomènes de bruits **[7]** et il est donc difficile de garantir l'intégrité des valeurs qu'ils communiquent. Une solution permettant de résoudre ce problème est d'utiliser des répéteurs, le plus souvent des buffers, pour segmenter les fils ayant de longues distances à parcourir. Une solution complémentaire est d'utiliser des codes correcteurs et/ou détecteurs d'erreurs.

3.1.2 Les nœuds de routage

Son rôle est d'aiguiller les données qu'il reçoit vers leurs destinataires **[8]**. La **(voir Figure 6)** illustre l'architecture générique d'un nœud de routage qui est constituée des éléments suivants :

Les files d'attente qui sont utilisées pour stocker les données en transit. Elles permettent aussi de faciliter le trafic sans bloquer les émetteurs (tant que les files ne sont pas pleines).

Le commutateur ou matrice de commutation qui connecte les files d'attente d'entrée aux files d'attente de sortie.

L'unité de routage et d'arbitrage qui définit comment doit être configuré le commutateur pour que les données contenues dans les tampons d'entrée soient correctement aiguillées. De plus, il doit gérer les conflits d'accès lorsque plusieurs files d'attente d'entrée doivent être connectées à la même file de sortie.

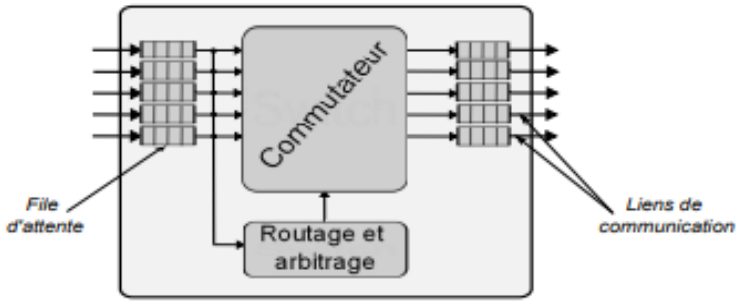

Figure 6 : Architecture générique d'un nœud de routage [7]

3.1.3 Les interfaces réseau [7]

Les interfaces réseau sont les points d'entrée-sortie d'un réseau sur puce. Une interface réseau fournit un ensemble de services de communication de haut niveau (e.g. 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 **(voir Figure 7)** 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.

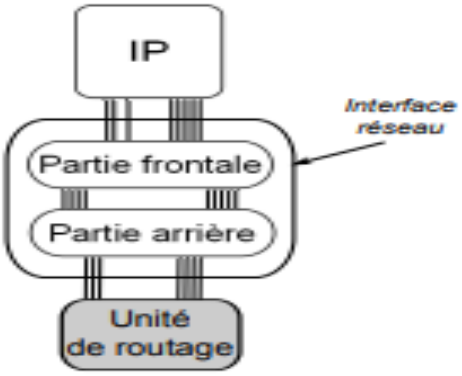

Figure 7 : Architecture d'une interface-réseau [7]

4. Topologies standards

Dans la littérature, les topologies sont généralement réparties en trois classes :

Les réseaux directes chaque nœud de routage est connecté à au moins un IP.

Les réseaux indirects les nœuds de routage peuvent être connectés seulement à d'autres nœuds de routage.

Les réseaux irréguliers (Hybrides) l'interconnexion des nœuds de routage est une combinaison des réseaux directes et indirects.

Il existe plusieurs types de topologie dans la littérature pour la représentation du réseau sur puce. Nous avons choisi de présenter les plus couramment utilisées.

4.1 Topologie 2D maillé (Mesh)

La topologie 2D maillée est la plus utilisée **(voir Figure 8).** Elle est simple de mise en œuvre car tous les liens ont la même longueur. Dans cette topologie, chaque sommet (un nœud de routage) est connecté au moins à deux autres sommets. Elle offre une grande scalabilité et permet d'utiliser des stratégies de routage simple. Parmi les réseaux développés qui utilisent cette topologie, on trouve : HERMES **[9],**NOSTRUM **[10]**.

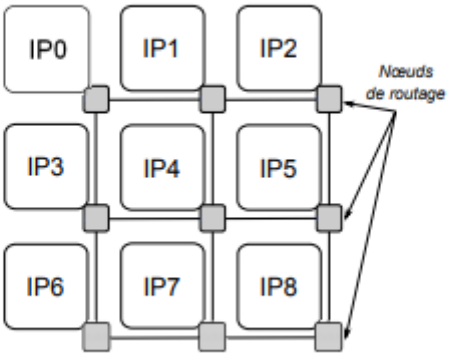

Figure 8 : La topologie de maillage à deux dimensions (2D mesh) [11]

4.2 Topologie tore (Torus)

C'est une topologie dérivée de la structure 2D, elle est très semblable à celle de la topologie 2D maillé. Elle offre une bande passante légèrement supérieure à celle de la 2D maillée mais elle est plus complexe à implémenter et très difficile à tester [11].

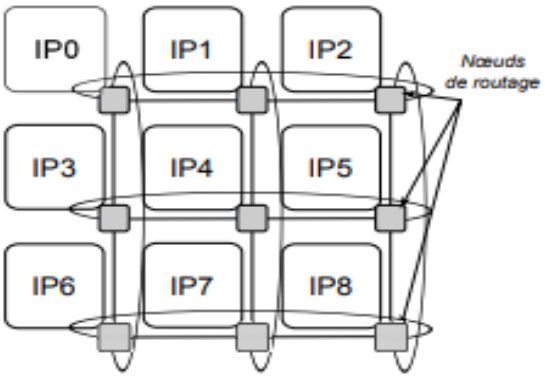

Figure 9 : La topologie tore (torus) [11]

4.3 La topologie anneau

Cette topologie ressemble à une topologie en bus, sauf qu'elle n'a pas de fin ni de début, elle forme une boucle. La topologie en anneau est une structure facilement intégrable sur silicium **(voir Figure 10),** Cependant, elle n'est pas extensible car ses performances se dégradent (la bande passante devient limitée) au fur et à mesure que le nombre d'IPs connectées augmente.

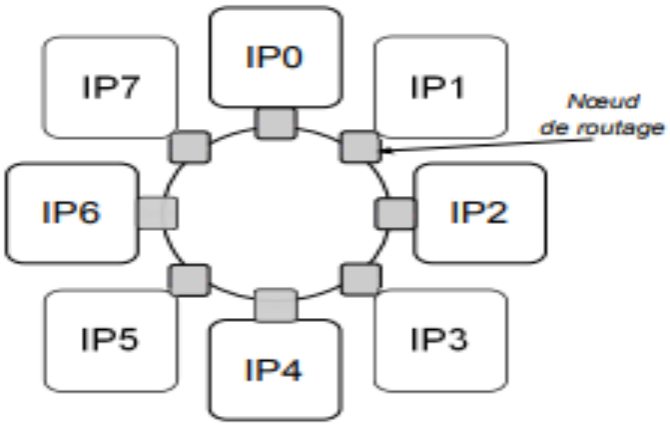

Figure 10 : La topologie anneau (ring)

4.4 La topologie anneau à cordes

Cette topologie a été introduite pour remédier aux problèmes de performance de la connexion anneau. Elle permet de réduire la latence en offrant la garantie de ne traverser au maximum que deux nœuds de routage pour n'importe quelle communication au sein du réseau.

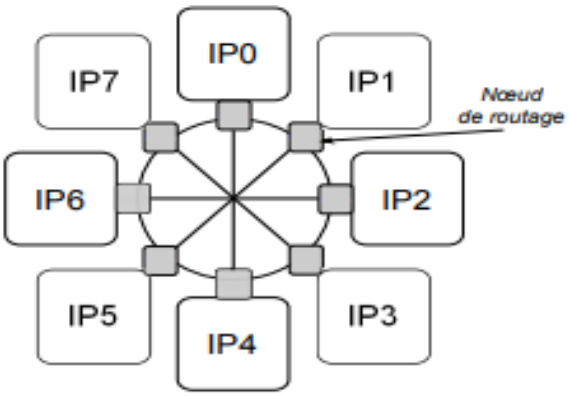

Figure 11 : La topologie anneau à cordes (chordal ring)

4.5 La topologie papillon

La **(voir Figure 12)** montre un réseau de papillon 8x8 sommets dans lequel les données transitent, à partir des IPs d'entrée sur la gauche, à travers trois étages de nœuds de routage pour atteindre les IPs de sortie sur la droite.

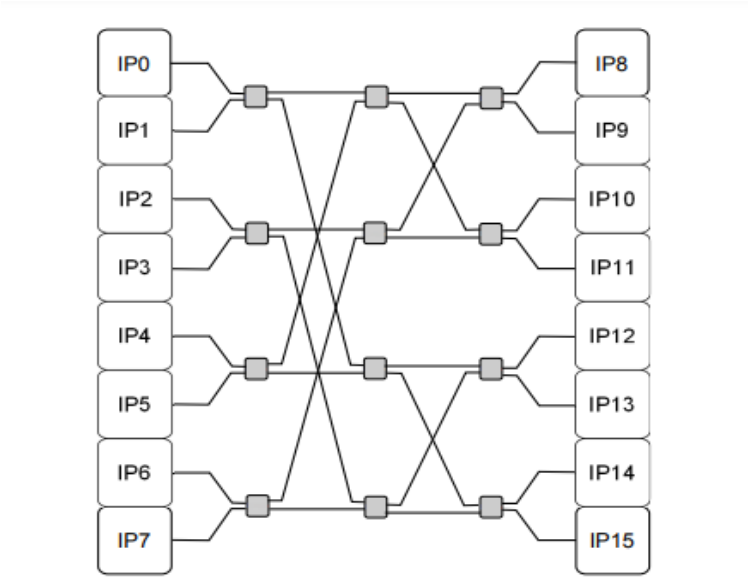

Figure 12 : La topologie papillon (butterfly)

4.6 La topologie 3D maillé (Mesh)

Elle offre une plus grande bande passante que la topologie 2D-maillé et la topologie 2D-torus, elle souffre d'un inconvénient majeur qui est la difficulté de son implantation sur silicium (routage complexe) **[12]**.

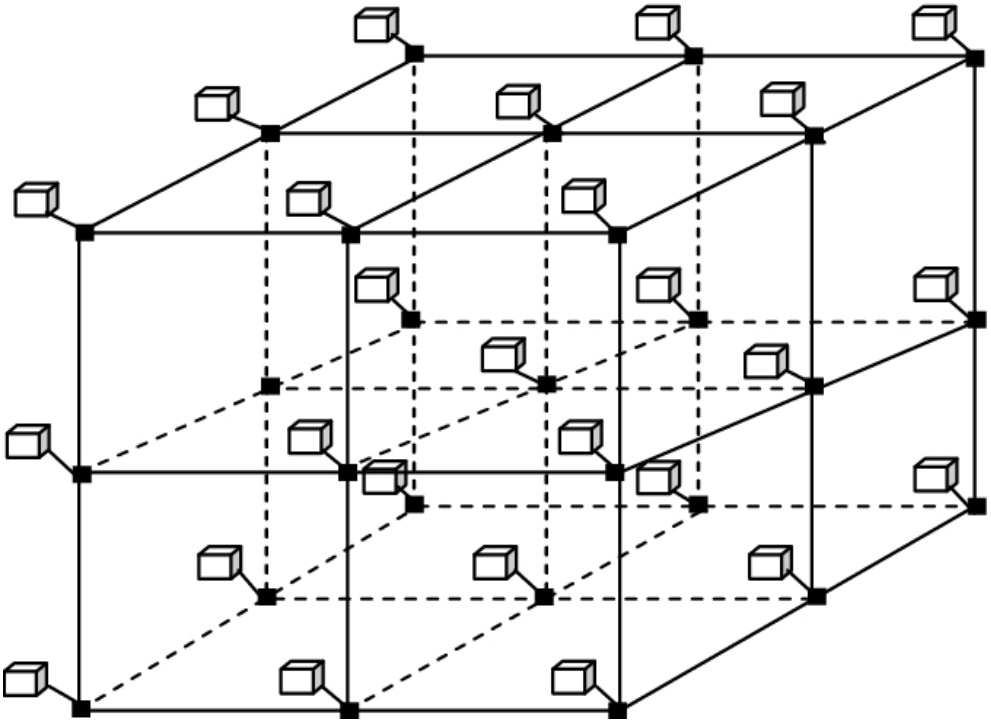

Figure 13 : La topologie 3D maillé (MESH)

5. Topologies personnalisées

Une topologie personnalisée est un ensemble d'une ou plusieurs topologies régulières avec des liens ajoutés ou supprimés **[13].** La **figure 14** présente topologie NoC personnalisé.

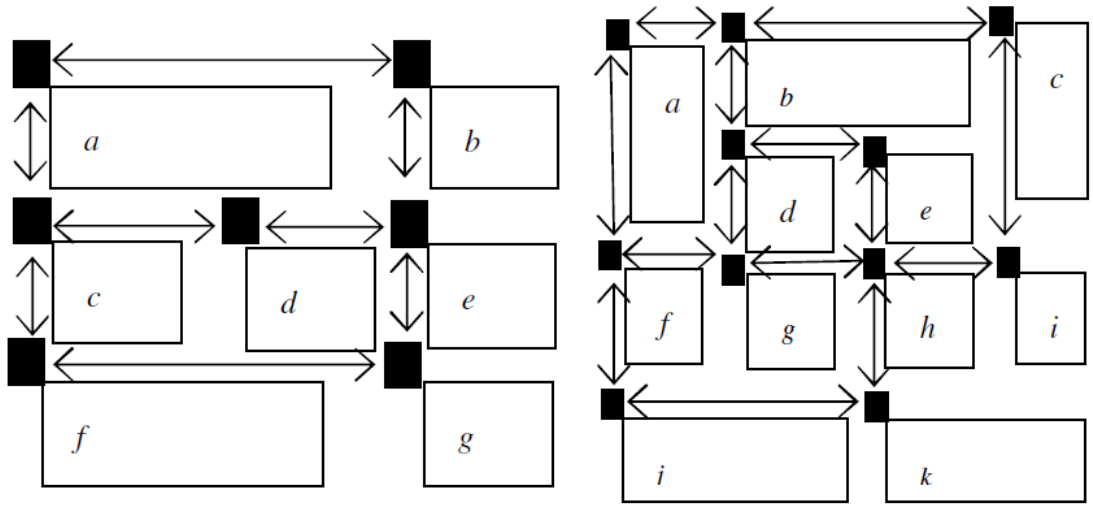

Figure 14 : NoC personnalisé [13]

6. Algorithme de routage

C'est un point très important lors de la conception des réseaux sur puce, son rôle est de définir le chemin (la suite des liens de communication à utiliser) que doit suivre un paquet pour atteindre sa destination **[15]**. Un bon algorithme de routage doit offrir les trois caractéristiques suivantes :

- Une faible latence.
- Un débit élevé.
- Une facilité de mise en œuvre.

Les techniques de routage les plus utilisées dans les structures de réseau sur puce sont X-Y, West-First et Negative-First.

6.1 L'algorithme de routage XY

L'algorithme XY est un algorithme déterministe (définit des chemins fixes) et minimal (le chemin le plus court) qui garantit toute situation de blocage **[14]**. Le principe de cet algorithme est de router les paquets horizontalement (dans la dimension X), puis verticalement (dans la dimension Y) vers sa destination.

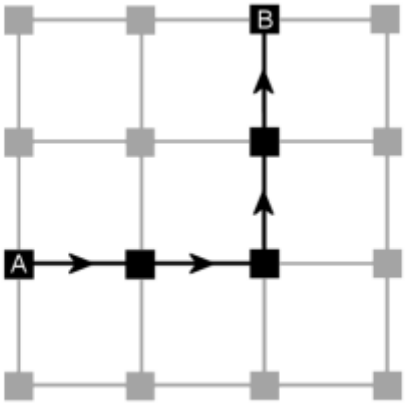

Figure 15 : L'algorithme de routage XY [15]

6.2 Algorithm Ouest-d 'abord (West-first)

Cet algorithme de routage empêche tous les virages vers l'ouest. Donc les paquets allant vers l'ouest doivent d'abord transmis loin de l'ouest que nécessaire. Le routage des paquets vers l'ouest n'est pas possible plus tard **[15]**.

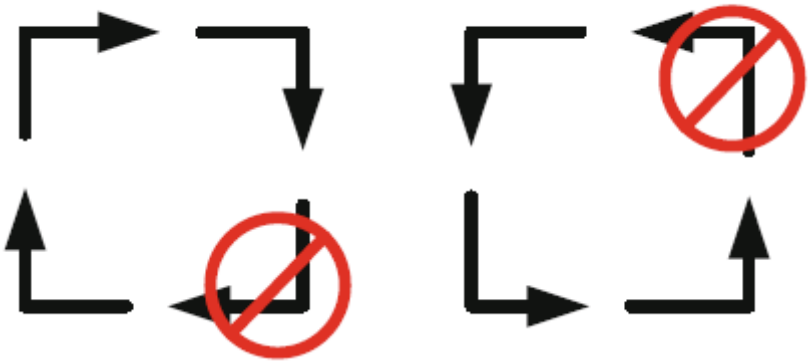

Figure 16 : Tours autorisés en Ouest-d 'abord [15]

6.3 L'algorithme Négatif-d 'abord (Négative-first)

Cet algorithme de routage autorise tous les autres virages, à l'exception des virages de la direction positive à la direction négative. Les routages de paquets vers des directions négatives doivent être effectués avant toute autre chose **[15]**.

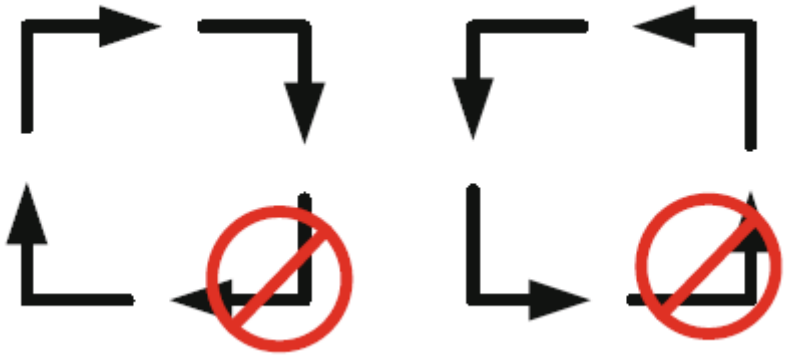

Figure 17 : Tours autorisés en Négatif-d'abord [15]

7. Les paramètres des réseaux sur puce

Plusieurs critères peuvent être pris en compte dans l'évaluation des caractéristiques d'une architecture de réseau sur puce. 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 **[4]**.

7.1 Débit de données

Il définit la quantité de données qui transitent d'une ressource à l'autre par unité de temps **[4]**.

7.2 La latence

C'est le temps entre l'émission d'une donnée sur le réseau et sa réception par la ressource destinatrice **[4]**.

7.3 La surface

C'est l'aire occupée par les composants du réseau (les nœuds de routage, les liens de communication, les blocs IPs insérés dans le réseau).

7.4 La consommation d'énergie

C'est la somme d'énergie consommer par l'ensemble des composants du réseau (les nœuds de routage, les liaisons, les IPs).

8. La conception des réseaux sur puce

Il existe actuellement un nombre important d'implémentations de réseaux-sur-puce aussi bien académiques, destinés à la recherche, qu'industriels à vocation commerciale. Dans ce qui suit nous présenterons les plus connus :

8.1 Exemples de réseaux sur puce

8.1.1 SPIN [12]

Le réseau SPIN a été développé au sein du laboratoire LIP6 (laboratoire d'informatique de paris) en 2000. Le réseau SPIN et son nœud de routage RSPIN est l'un des premiers micro-réseaux. Il a une topologie Fat-Tree **[16],** et un algorithme de routage adaptatif (les décisions d'acheminement des paquets sont prises au cours du temps en fonction des conditions du réseau) et distribué (les décisions de routage sont prises par chaque nœud de routage).

8.1.2 MANGO

C'est un réseau sur puce développé par l'Université Technique du Danemark qui adopte une topologie en forme de grille **[7] (Exemple d'une topologie de forme grille : voir Figure 8)**, et qui utilise un algorithme de routage déterministe (définit des chemins fixes).

8.1.3 ÆTHEREAL

C'est un réseau sur puce développé par Philips ayant une topologie non-régulière et utilisant un algorithme de routage par la source (les informations de routage sont additionnées dans le paquet à transmettre et les autres nœuds de routage consultent ces informations pour acheminer le paquet) **[7]**.

8.1.4 FAUST [12]

FAUST est un réseau développé par le CEA LETI de Grenoble qui a été proposé dans le cadre du projet Européen 4MORE avec une topologie de grille.
Les réseaux sur puce permis d'interconnecter plusieurs modules dans la puce de silicium et représenter une solution pour les problèmes actuelles, mais la conception reste complexe et délicate car il est difficile de concevoir un Noc efficace pour exploiter pleinement le nombre et la puissance de traitement des IPs dans un même circuit. Donc il est nécessaire d'automatisé les tâches de conception par des méthodes et des outils. Dans ce qui suit on va décrire les phases de conception des nocs et quelques outils d'aide à la conception des réseaux sur puce.

8.2 Les phases de conception d'un réseau sur puce

Conception d'un réseau sur puce efficace satisfaisante les critères de l'application est un processus complexe et portant plusieurs niveaux abstracts, de niveau modélisation jusqu'à l'implémentation physique. Elle passe généralement par les phases suivantes :

- **Modélisation du trafic généré par l'application.**
- **Choix de la topologie adaptée.**
- **L'ordonnancement et l'assignation des tâches aux IPs :** affecter les tâches aux IPs et déterminer la séquence d'exécutions des tâches. 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).

- **Mapping des IPs sur la topologie :** affecter les IPs aux tuiles de l'architecture tout en minimisant les coûts de communication, la consommation d'énergie ou la latence du système.
- **Allocation des chemins de routage et réservation des ressources :** allocation des chemins utilisés par les IPs pour la communication. Cette phase de routage prend en considération aussi le coût de communication : par le choix de l'algorithme de routage, la conception de nœud de routage spécifique (par exemple définir la taille des buffers), etc.

- **Test de performance.**
- **Implémentation.**

8.3 Outils d'Aide à la Conception des Réseaux sur Puce [17]

Voici quelques outils qui automatisent différentes phases de la conception des Nocs :

8.3.1 NS2 et NS3

NS2, NS3 sont des simulateurs de réseau utiles pour tout réseau afin de déterminer les performances du réseau. NS2 et NS3 peuvent être utilisés pour les réseaux sur puce (Noc) afin de déterminer ses performances telles que la vitesse et la consommation d'énergie.

8.3.2 Noxim

Ce simulateur est utilisé pour concevoir et analyser divers composants du réseau sur puce et de son routage. C'est un simulateur de Noc pour déterminer ses paramètres. Avec l'utilisation de Noxim, on peut personnaliser la structure du Noc, l'algorithme de routage et la transmission de données etc.

8.3.3 Visualsim

Ce simulateur utilise le schéma d'allocation de canal virtuel (VC) pour le routage à l'intérieur du Noc. Visualsim est un nouveau simulateur pour la recherche Noc. Il s'agit d'un simulateur convivial utile pour obtenir le délai et la consommation d'énergie du Noc.

8.3.4 X-pipes Compiler

C'est un simulateur pour simuler le Noc du système hétérogène sur puce (Soc). Ce simulateur optimise automatiquement les blocs de construction de Noc. Il est utile pour obtenir des résultats optimisés en termes de latence et de consommation d'énergie.

Les outils d'aide aident les chercheurs à sélectionner l'outil approprié pour leur travail de recherche.

9 Conclusion

Dans ce premier chapitre, nous avons présenté les réseaux sur puce (Nocs) qui sont venus pour remédier et résoudre les problèmes rencontrés dans les structures d'interconnexions des Socs. On a vu les composants d'un Noc, ses caractéristiques, quelques réseaux sur puce, ainsi les phases de conception des nocs et quelques outils d'aide à la conception des réseaux sur puce afin de permettre d'automatiser ces phases.

De nombreux paramètres sont à prendre en compte lors de la conception des réseaux sur puce, car chacun d'entre eux peut influencer directement ou indirectement les performances globales de système. Trouver une combinaison optimale de ces paramètres afin de maximiser les performances d'un Noc est un problème d'optimisation combinatoire NP-difficile.

Dans le chapitre suivant nous étudions les techniques d'optimisation et les méthodes dédiées aux résolutions du problème d'optimisation afin de proposer aux concepteurs des réseaux sur puce un outil d'aide de conception qui maximise plusieurs performances des Nocs.

Chapitre 2

L'optimisation combinatoire

1. Introduction

L'optimisation occupe actuellement une place importante dans tous les domaines (le mapping, l'assignation, … etc.). Elle consiste à minimiser (ou maximiser) une ou plusieurs fonctions objectives.

Une méthode d'optimisation c'est cherché le point ou un ensemble de points dans l'espace des états possibles qui satisfait au mieux un ou plusieurs critère(s), le résultat est appelé valeur optimal ou optimum **[18]**.

L'optimisation mono-objectif permet de définir facilement la solution optimale. Ce n'est pas le cas quand on a plusieurs objectifs. L'optimisation multi-objectifs est une direction de recherche très importante à cause de la nature multi objectif de la plupart des problèmes réels.

Pour les Nocs, objet de notre étude, les critères les plus connus sont la bande passante, le coût de communication, la latence, le débit, la puissance, la consommation d'énergie et les dimensions de la puce etc. Satisfaire ces critères est notre objectif. De ce fait, ce chapitre sera consacré à l'étude des méthodes d'optimisation multi objective.

2. L'optimisation

Parmi toutes les solutions possibles pour un problème donné, le processus de trouver un ensemble de solutions optimales est appelé optimisation. Les problèmes d'optimisation **(PO)** sont classés selon différents critères. Un problème d'optimisation est dit linéaire si toutes les fonctions objectif et contraintes sont linéaires sinon il est dit non-linéaire. Les **(PO)** sont classés selon le type de variables : si les variables sont continues, elle est continue, et si les variables sont du type entier elle est discrète. Les problèmes d'optimisation ayant un seul objectif sont dits mono-objectif, alors que s'il y a plusieurs objectifs

à optimiser, le problème est dit multi-objectif. Et selon le nombre de variables, il y a le **(PO)** à petite échelle et le **(PO)** à grande échelle **[19]**.

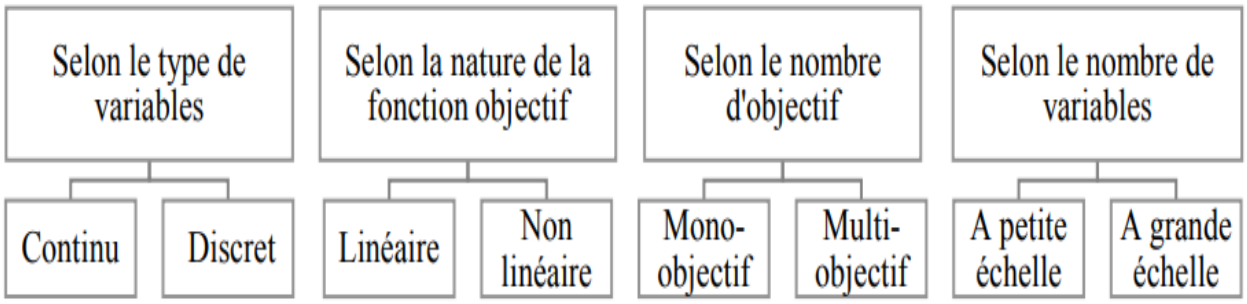

Figure 18 : Classification des problèmes d'optimisation [19]

2.1 Le problème d'optimisation :

Un problème d'optimisation se définit comme la recherche du minimum ou du maximum (de l'optimum donc) d'une fonction donnée. On peut aussi trouver des problèmes d'optimisation pour lesquels les variables de la fonction à optimiser sont contraintes d'évoluer dans une certaine partie de l'espace de recherche. Dans ce cas, on a une forme particulière de ce que l'on appelle un problème d'optimisation sous contraintes **[20].** Mathématiquement parlant, un problème d'optimisation se présentera sous la forme suivante :

$$\begin{array}{lll} \text{Minimiser} & f(\vec{x}) & (\text{fonction à optimiser}) \\ \text{Avec} & \vec{g}(\vec{x}) \leq 0 & (m \text{ contraintes d'inégalité}) \\ \text{Et} & \vec{h}(\vec{x}) = 0 & (p \text{ contraintes d'égalité}) \end{array}$$

$$\text{On a } \vec{x} \in \mathrm{R}^n, \vec{g}(\vec{x}) \in \mathrm{R}^m \text{ et } \vec{h}(\vec{x}) \in \mathrm{R}^p$$

Ici, les vecteurs $\vec{g}(\vec{x})$ et $\vec{h}(\vec{x})$ représentent respectivement m contraintes d'inégalité et p contraintes d'égalité. Cet ensemble de contraintes délimite un espace restreint de recherche de la solution optimale.

2.2 Vocabulaire et définitions [21]

- **Fonction objective** : Elle est également appelée critère d'optimisation, fonction de coût, fonction d'adaptation, ou encore performance (fonction fitness). Il s'agit d'une équation mathématique qui exprime ce qu'on désire améliorer dans un dispositif (c'est la fonction f qu'on cherche à optimiser).
- **Paramètres :** On les appelle également variables d'optimisation, variables de conception ou variables de projet. Correspondent aux variables de la fonction objective. Ils sont ajustés pendant le processus d'optimisation, pour obtenir les solutions optimales.
- **Espace de recherche :** Également connu sous le nom espace des paramètres. Il est défini par un ensemble de combinaisons de valeurs de paramètres. Correspond à l'espace des solutions. La dimension de l'espace de recherche est définie par le nombre de paramètres impliqués dans les solutions.
- **Espace des objectifs :** Ensemble de l'espace de recherche, déterminé par toutes les valeurs possibles des fonctions objectives.
- **Les Contraintes :** Sont définit comme des conditions sur l'espace d'état que les variables doivent les satisfaire. Ces contraintes sont souvent des contraintes d'inégalité ou d'égalité et permettent en général de limiter notre espace de recherche.
  - ✓ $g(x) \leq 0$ Contraintes d'inégalité
  - ✓ $h(x) = 0$ Contraintes d'égalité
- **Domaine réalisable :** Aussi appelé espace admissible. C'est la région de l'espace (des paramètres et/ou des objectifs) dans laquelle les contraintes sont respectées.
- **Domaine non-réalisable :** Région de l'espace où les contraintes sont violées. Le mécanisme d'exploration de l'espace de recherche propre à chaque méthodologie d'optimisation est contrôlé par des paramètres de contrôle (nombre d'itérations, direction de recherche, vérification de convergence, etc.) et des conditions initiales (valeurs initiales des paramètres, limites des domaines, etc.). La **Figure 19** illustre le principe de base d'une méthodologie d'optimisation.

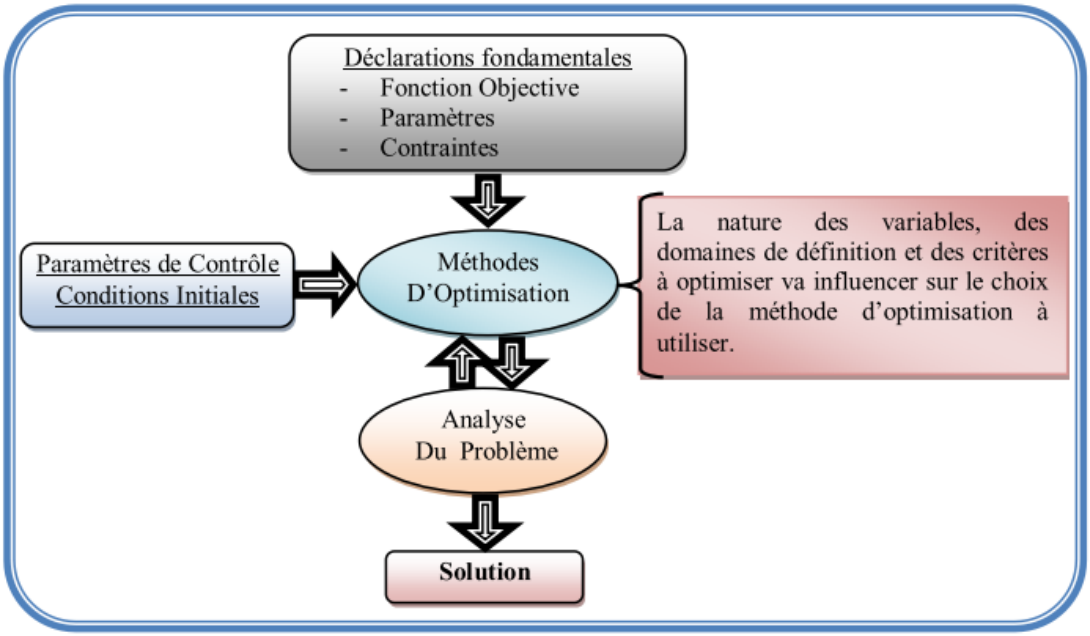

Figure 19 : Principe de base d'une méthodologie d'optimisation [21]

- **Variables de décision** : elles sont regroupées dans le vecteur $\overrightarrow{x}$ qui correspond à l'ensemble des variables du problème.
- **Problème mono-objectif :** Un problème d'optimisation mono-objectif est présenté sous la forme suivante :

  Minimiser $f(x)$ (fonction à optimiser)
- **Problème multi-objective :** Un problème d'optimisation multi-objectif est représenté sous la forme suivante :

  Minimiser $f_i(x), i = 1, \ldots, N_{obj}$ (N_{obj} nombre de fonction à optimiser)

2.3 La dominance

Dans le domaine d'optimisation multi objectif, le décideur évalue généralement une solution par rapport à chacun des critères, et se positionne donc naturellement dans l'espace objectif. Néanmoins, contrairement au cas mono-objectif où il existe un ordre total sur **R** parmi les solutions réalisables, il n'existe pas généralement de solution qui serait à la fois optimale pour chaque objectif, étant donnée la nature conflictuelle de ces derniers. Ainsi, une relation de dominance au sens de Pareto **[22]**.

2.3.1 Définitions :

- **Définition 1 (Dominance Pareto) :** Soit $x^{(i)}$, $x^{(j)}$ sont des solutions du **PMO**, $x^{(i)}$ domine $x^{(j)}$, si les conditions suivantes sont vérifiées **[23]** :

$$f_q\left(x^{(i)}\right) \leq f_q\left(x^{(j)}\right) \forall\ q\ \in \{1, \ldots\ldots, m\}. \qquad \mathbf{1}$$

$$\exists q\ \in \{1, \ldots\ldots, m\}\ tel\ que\ f_q\left(x^{(i)}\right) < f_q\left(x^{(j)}\right). \qquad \mathbf{2}$$

Si la solution $x^{(i)}$ domine la solution $x^{(j)}$ on écrit $x^{(i)} < x^{(j)}$.

- **Définition 2 (Vecteur non-dominé) :** On dit qu'une solution x est non-dominé s'il n'existe pas une autre solution x' tel que $x' < x$.

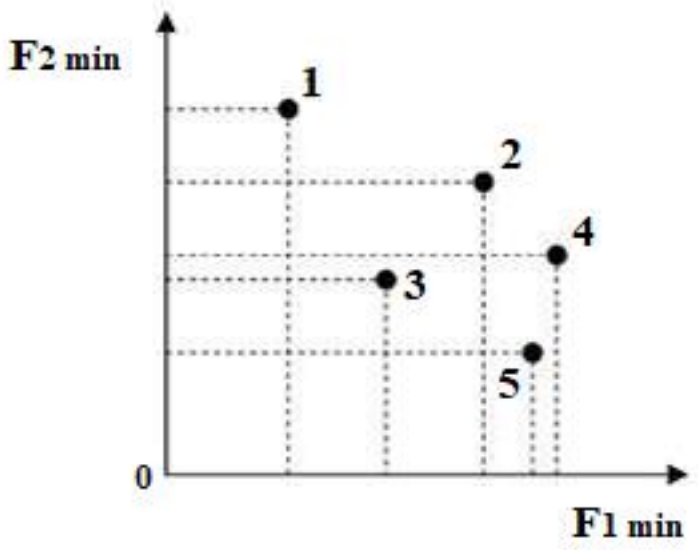

Figure 20 : exemple des solutions non dominées [23]

L'exemple ci-dessus, montre que les points 1,3 et 5 ne sont dominés par aucun autre. Alors que le point 2 est dominé par le point 3, et le point 4 est dominé par les points 3 et 5.

- **Définition 3 (Solution Pareto optimale) :** Une solution est dite Pareto-optimal si elle n'est dominée par aucune autre solution réalisable. On définit l'ensemble optimal de Pareto P* **[23]** :

$$P^* = \{x \in P | x \; est \; Pareto \; optimal\}$$

Et le Front Pareto par :

$$F_{P^*} = \{f(x) \in R^n | x \; est \; Pareto \; optimal\}$$

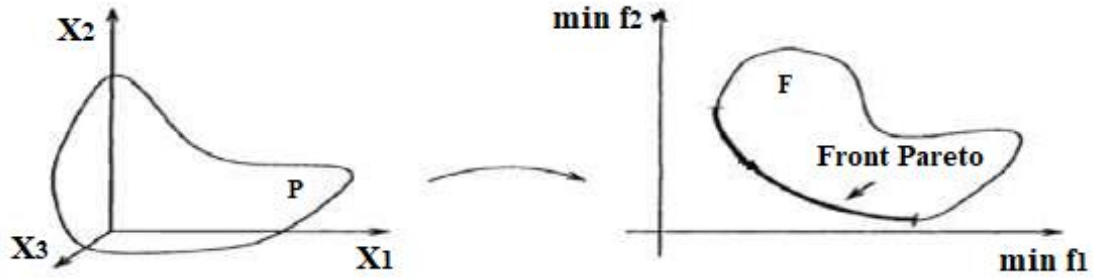

Figure 21 : Espaces de décision et d'objectif, avec le front Pareto pour un problème de minimisation de deux objectives avec trois variables de décision. [23]

- **Définition 4 (Dominance faible) :** Un vecteur objectif z ∈ Z domine faiblement un vecteur objectif

$$z' \in Z \; ssi \; \forall \; i \in \{1,2, \ldots\ldots, n\}, z_i \leq z'_i. \qquad \mathbf{4}$$

Cette relation sera notée $z \geq z'$.

- **Définition 5 (Point idéal) :** Le point idéal $z^* = (z_1^*, z_2^*, \ldots\ldots\ldots, z_n^*)$ est le vecteur qui minimise chaque fonction objective individuellement,

$$z_i^* = min_{x \in X} \; f_i(x) \; pour \; i \in \{1,2, \ldots\ldots\ldots, n\}. \qquad \mathbf{5}$$

3. Méthodes de résolution [24]

Les problèmes d'optimisation sont résolus par des méthodes d'optimisation. Une classification est illustrée à la **Figure 22.**

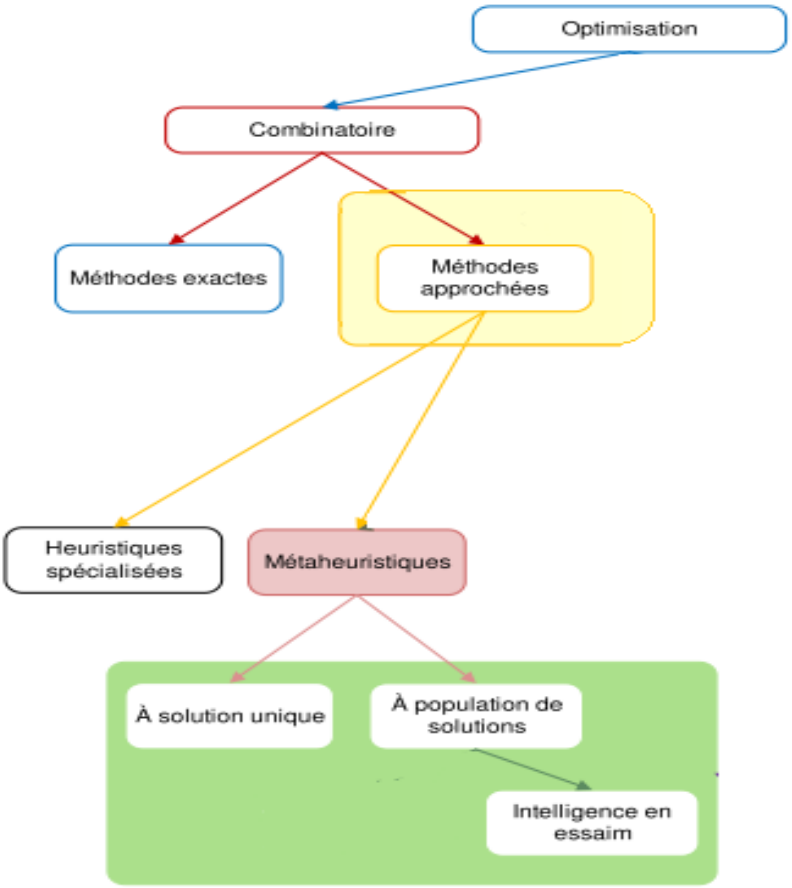

Figure 22 : Classification des méthodes d'optimisation [24]

3.1 L'optimisation combinatoire

L'optimisation combinatoire regroupe une large classe de problèmes d'optimisation. Un problème d'optimisation combinatoire **(POC)** est un problème d'optimisation dont la résolution se ramène à l'examen d'un nombre fini de combinaisons. Bien souvent cette résolution se heurte à une explosion du nombre de combinaisons à explorer **[25]**.

Plusieurs méthodes ont été proposées pour la résolution de problème d'optimisation combinatoire.

3.1.1 Méthodes exactes

Elles garantissent de trouver l'ensemble des solutions optimales, ces méthodes explorent tout le domaine des solutions et retournent l'optimum théorique, c'est-à-dire exact. Cependant leur approche par énumération accroît leur temps d'exécution d'une manière exponentielle en fonction de la taille du problème **[26]**. Parmi les méthodes de résolution exacte on trouve : Branch and Bound **(BB),** Programmation dynamique **[27]**.

Les méthodes exactes ne sont efficaces que pour les instances de problèmes de petites tailles, on va s'intéresser qu'aux méthodes approchées.

3.1.2 Méthodes approchées

Elles offrent la possibilité de disposer d'une solution de bonne qualité (optimale ou proche de l'optimale) en un temps réduit. Ceci est avantageux lorsque la taille du problème à traiter est importante. Ces méthodes utilisent des approches de recherche pseudo aléatoires et des techniques d'optimisation qui explorent et exploitent l'espace des solutions en se basant sur l'expérience apprise afin d'atteindre une solution dite de bonne qualité **[28]**.

Les méthodes approchées peuvent être classées **en deux classes** :

a) Les heuristiques

Les méthodes dites heuristiques sont des méthodes spécifiques à un problème particulier. Elles nécessitent des connaissances du domaine de problème traité. En fait, les heuristiques se basent sur l'expérience et les résultats acquis afin d'améliorer les recherches futures **[29]**.

b) Les métaheuristiques

Les méta-heuristiques sont une famille d'algorithmes stochastiques destinés à la résolution des problèmes d'optimisation. Leur particularité réside dans le fait que celles-ci sont adaptables à un grand nombre de problème sans changements majeurs dans leurs algorithmes. Elles sont donc composées d'éléments

invariants, ainsi que des éléments spécifiques au problème considéré, tel que la représentation ou l'évaluation d'une solution **[30]**.

A l'inverse des méthodes exactes, les méta-heuristiques permettent de résoudre des problèmes complexes de grandes tailles en délivrant des solutions satisfaisantes en un temps de calcul raisonnable. Néanmoins, il n'existe pas de garantie quant à leur optimalité **[31]**.

Deux types de méta-heuristique peuvent être distingués : **les méta-heuristiques à base de solution unique** (tels que l'algorithme de recuit simulé) et **les métaheuristiques à base de population** (tels que les algorithmes évolutionnaires).

- ## Les métaheuristiques à base de solution unique

Les métaheuristiques à base de solution unique appelées aussi méthodes de recherches locales sont des méthodes itératives consistant à traiter une seule solution en essayant de l'optimiser (la remplacer par une autre issue de son voisinage **[32]**). Un exemple illustrant ce principe est donné en **Figure 23**, où le passage d'une solution ***s*** à une solution ***s'*** dans l'espace ***X*** se fait sur la base d'un ensemble de modifications élémentaires (en spécifiant une fonction de voisinage appelée ***N*(*s*)**). Ce processus d'exploration est réitéré jusqu'à ce qu'un ou plusieurs critères d'arrêt soient satisfaits. Les passages successifs d'une solution à une solution voisine définissent un chemin (appelé trajectoire) **[33]**. Les algorithmes les plus représentatifs de cette catégorie sont le recuit simulé **(SA)** et la recherche tabou **(TS)**.

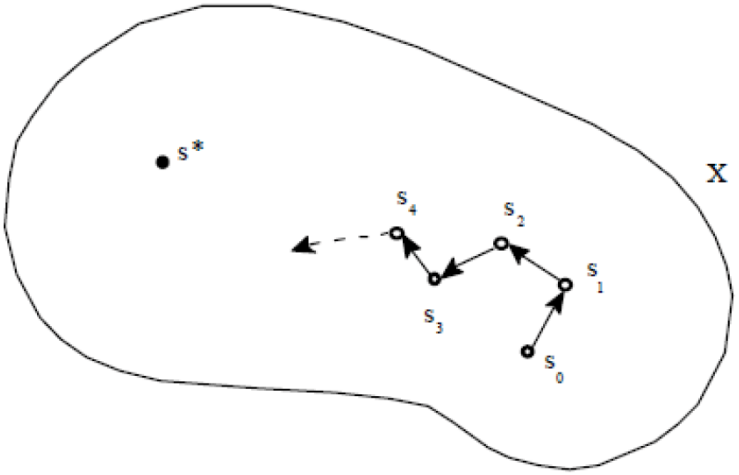

Figure 23 : Exploration de l'espace X par une approche locale [32]

1) Le Recuit Simulé

Le recuit simulé **(SA)** est un algorithme d'optimisation métaheuristique (itératif) qui apporte une solution au problème par l'attraction des optimums locaux. Une solution voisine meilleure que la solution actuelle sera toujours acceptée. Une solution voisine moins bonne que la solution actuelle sera parfois acceptée selon la probabilité **[34]**.

Le déroulement de cet algorithme :

- On se donne un réel aléatoire **T>0** : la température.
- Des transformations définissant le voisinage **V(s)** de **s**.
- On tire au sort une transformation **s => s'**dans **V(s)**.
- Soit Δ**f = f(s') - f(s)** la variation de coût.
- Si $\Delta f \leq 0$ on effectue la transformation : **s** devient **s'**.
- Si $\Delta f \geq 0$ on accepte la transformation : **s** devient **s'**avec la probabilité **Expo(-**Δ**f/T)**. (On tire au sort un nombre **x** dans **[0,1]** avec une loi uniforme et on accepte si $x \leq exp(-\Delta f/T)$.
- On diminue légèrement la température. Par exemple **T=0.999** **T** (en général on la diminue par palier).
- On recommence tant que **T** n'a pas atteint un seuil fixé.

2) La recherche tabou

Est une méthode métaheuristique de la recherche locale utilisée pour résoudre des problèmes complexes et/ou de très grande taille (souvent NP-durs). La recherche-tabou a plusieurs applications en programmation non linéaire **(PNL)**. L'algorithme tabou choisit le meilleur voisin non tabou, même si celui-ci dégrade la fonction de coût **[35]**.

- **Les métaheuristiques à population de solutions**

Contrairement aux algorithmes qui partent d'une solution unique, les métaheuristiques à population de solutions améliorent, au fur et à mesure des itérations une population de solutions **[36]**. Dans cette catégorie, on distingue **les algorithmes d'intelligence en essaim** qui proviennent d'analogies avec des phénomènes biologiques naturels **(voir la Figure 24)**.

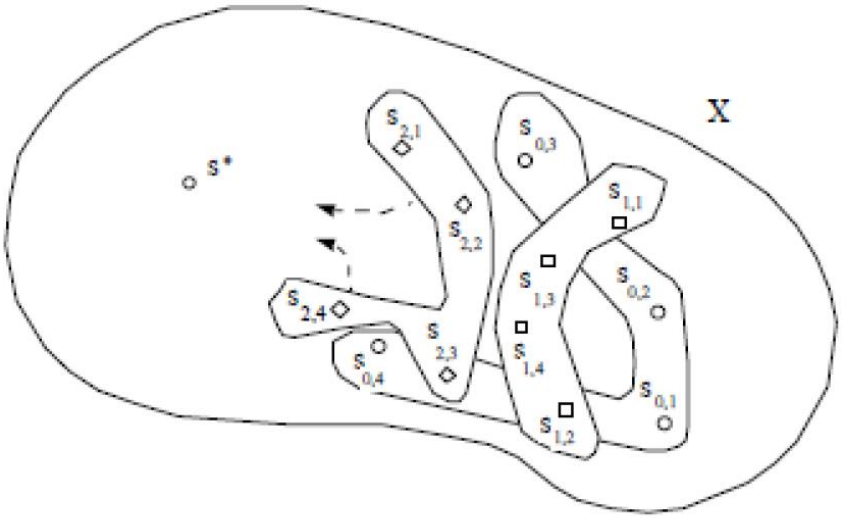

Figure 24 : Exploration de l'espace X par une approche évolutive [36]

L'intelligence en essaim

L'intelligence en essaim **(SI)** est née de la modélisation mathématique et informatique des phénomènes biologiques rencontrés en éthologie **[37]**. Elle recouvre un ensemble d'algorithmes, à base de population d'agents simples (entités capables d'exécuter certaines opérations), qui interagissent localement les uns avec les autres et avec leur environnement. Ces entités, dont la capacité

individuelle est très limitée, peuvent conjointement effectuer de nombreuses tâches complexes nécessaires à leur survie. Bien qu'il n'y ait pas de structure de contrôle centralisée qui dicte la façon dont les agents individuels devraient se comporter, les interactions locales entre les agents conduisent souvent à l'émergence d'un comportement collectif global et autoorganisée. Exemples phares d'algorithmes de l'intelligence en essaim sont les algorithmes de colonies de fourmis **ACO**.

- **L'optimisation par les colonies de fourmis ACO [38]**

Algorithmes de fourmis, également connus sous le nom l'optimisation de colonies de fourmis **ACO**, sont une classe d'algorithmes de recherche heuristiques biologiquement inspirés du comportement des colonies de fourmis, et en particulier de leur recherche de nourriture. L'idée derrière cette approche est que les fourmis peuvent communiquer entre elles par des moyens indirects en modifiant la concentration de produits chimiques hautement volatils appelés phéromones chez leur voisin environnement. Cette forme de communication indirecte médiatisée par l'environnement s'appelle la stigmergie, et typique des insectes. Les principales étapes de l'**ACO** sont :

(i) L'utilisation de simulations répétées réalisées par une population d'agents artificiels appelés "fourmis" pour générer de nouvelles solutions au problème.
(ii) L'utilisation par les agents de recherche locale stochastique pour construire les solutions dans de manière progressive.
(iii) L'utilisation des informations collectées lors de simulations passées (phéromones artificielles) pour orienter le futur cherché de meilleures solutions.

Plusieurs algorithmes de fourmis utilisent de la structure montrée dans l'algorithme 1.

Algorithme 1 : ACO méta-heuristique

1 : initialiser les paramètres et les traînées de phéromones ;

2 : tant que la condition de résiliation n'est pas remplie

3 : construire des solutions de fourmis ;

4 : recherche locale (facultatif) ;

5 : mettre à jour les pistes de phéromones ;

6 : fin tant que ;

Dans l'approche des colonies de fourmis artificielles, chaque fourmi construit une solution en utilisant deux types d'informations accessibles localement : informations spécifiques au problème et informations ajoutées par les fourmis lors des itérations précédentes de l'algorithme. En fait, alors que construction d'une solution, chaque fourmi collecte des informations sur caractéristiques du problème et sur ses propres performances, et utilise ces informations pour modifier la représentation du problème, vu localement par les autres fourmis. La représentation du problème est modifiée de telle manière que les informations contenues dans les bonnes solutions.

- **La relation de l'ACO avec la biologique de colonie de fourmis**

Pour connaitre la relation il faut faire une comparaison par un tableau entre les colonies de fourmis artificielles et les vraies colonies de fourmis pour définir quelque notion.

Tableau 2.1 : comparaison de la métaphore de la colonie de fourmis [39]

Monde réel **Colonie de fourmis**	**ACO**
Fourmi	Paquet de fourmi
Chemin	Canal de Noc (Les liens de communication)
Aliments	IP destinataire
Phéromone	Une fonction dans l'algorithme d'ACO appelée phéromone
La phéromone est déposée dans les Allers-retours.	Les phéromones ne déposent souvent sur le trajet de retour qu'après la construction et l'évaluation de la solution.
L'effet de la phéromone, qui dépend de la quantité et de la qualité de l'aliment est mis à jour au fur et à mesure.	L'effet de la phéromone est mis à jour au fur et à mesure que la fourmi revient de son chemin, sachant que la quantité de phéromone est inversement proportionnelle à la longueur du chemin stocké dans sa mémoire.
Les vraies fourmis n'ont pas de mémoire.	Elle a une mémoire qui stocke les chemins parcourus. Elle doit être réutilisé lors du retour, et pour déterminer la longueur du trajet et la quantité de phéromone qui il doit être déposé.
La prédation et la concurrence avec d'autres colonies réduisent leur protection.	Les contraintes environnementales n'existent pas dans le monde industriel.

D'après le **(Tableau 2.1)**, le **ACO** est un algorithme qui construire par les recherches sur la vie réel de la colonie de fourmis. Les notions qui en peuvent définir :

- Les phéromones déposent sur le trajet de retour qu'après la construction et l'évaluation de la solution.
- Les contraintes environnementales n'existent pas dans le monde industriel.

- La fonction phéromone contient des caractéristiques de stigmergie (phéromone) de la fourmi réelle.
- Le paquet fourmis a des caractéristiques de la fourmi réelle.

4. Conclusion :

Dans ce chapitre nous avons présenté le problème d'optimisation, une classification des méthodes d'optimisation avec une définition sur l'**ACO**. On s'est intéressés aux méta-heuristiques qui constituent une classe de méthodes approchées adaptables pour résoudre le problème d'optimisation.

Nous traiterons dans ce projet certaines phases de conception dans un cadre multi-objectif. Dans le chapitre suivant, on va s'approfondir à la phase de mapping qui est une phase centrale lors de la conception des Nocs.

Chapitre 3

Le mapping dans les réseaux sur puce : état de l'art

1. Introduction

Les réseaux sur puce permettent d'interconnecter plusieurs modules sur puce et représentent une solution pour les limites des architectures actuelles, mais la conception reste complexe et délicate, donc il est nécessaire d'automatisé les phases de conception par des méthodes et des outils.

L'un des problèmes consiste de placer l'ensemble des composants communiquant sur la topologie du réseau de telle sorte que le coût de communication soit minimisé. Ce problème on ne peut pas le résoudre par une méthode exacte car son temps d'exécution est très lent, donc les méthodes approchées comme les méta-heuristiques sont utilisées ici.

Les méta-heuristiques donnent une solution de ce problème avec temps d'exécution réalisable et des contraintes sur la solution trouvée.

Ce chapitre est consacré à définir quelques méta-heuristiques et heuristiques qui peuvent résoudre le problème de mapping. Mais d'abord, on va définir la phase de mapping avec le problème de mapping, les types de Mapping (statique et dynamique), et le mapping dans les Noc 3D.

2. Phase de mapping

2.1 Définir la phase mapping

La phase de mapping consiste à mapper les IPs d'une application sur l'architecture du réseau sur puce de sorte à minimiser le coût de communications, la consommation d'énergie, la latence du système et respecter les contraintes telles que la bande passante ou le temps réel.

La **(figure 25)** montre le problème de mapping d'une application.

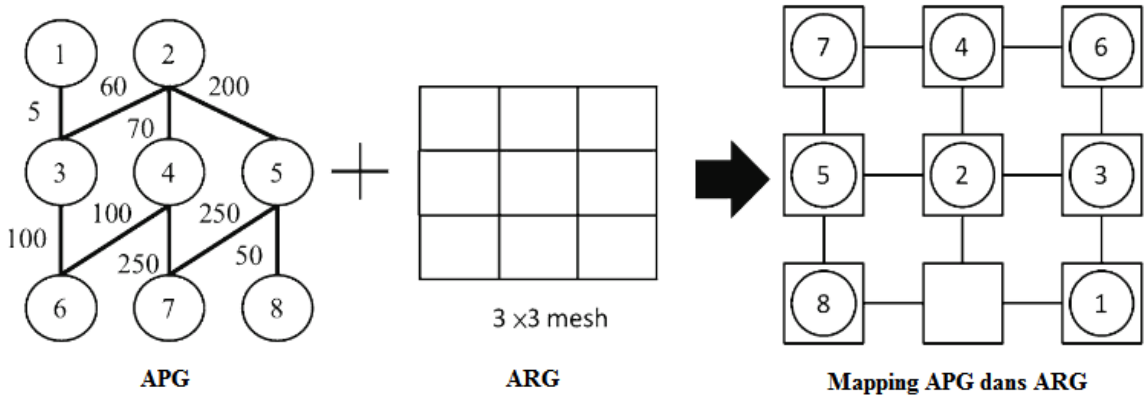

Figure 25 : Exemple d'un mapping

2.2 Classification des techniques de mapping [40]

Les techniques de mapping dans les réseaux sur puce peuvent être regroupées en techniques dynamique et techniques statique : le mapping dynamique est également appelé mapping en temp d'exécution ou mapping en temps réel et en ligne, où l'affectation des tâches d'application aux IPs de Noc sont exécutés en temps réel, cependant le mapping statique effectue le mapping des IPs en mode hors ligne au moment de la conception. La **(figure 26)** montre la classification des techniques de mapping d'application dans les Nocs.

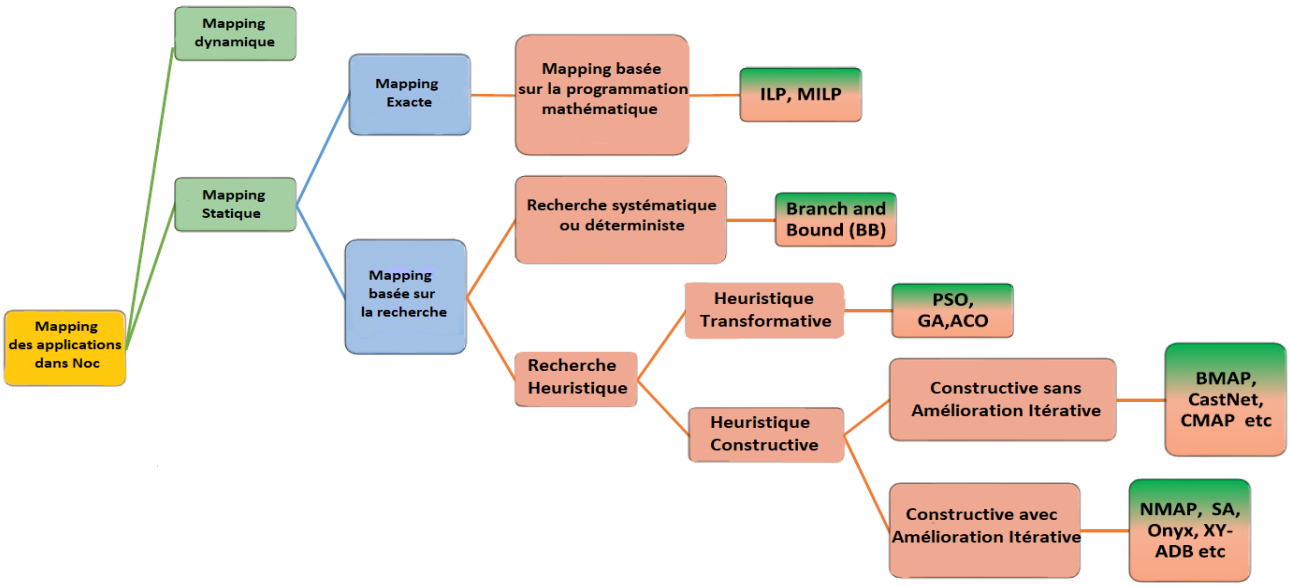

Figure 26 : Classification des techniques de mapping des applications dans les nocs [40]

2.2.1 Techniques de mapping dynamique :

- Une technique basée sur la sélection du premier nœud est la stratégie proactive de sélection de région **MapPro** présentée dans **[41]**. La technique exploite le temp d'inactivité entre deux demandes de mapping d'application consécutives pour trouver des régions candidates appropriées pour le mapping. L'idée d'effet d'entraînement définit les nœuds voisins adjacents pour le mapping. La technique **MapPro** offre une diminution du temps d'exécution et une faible congestion.

  L'idée d'effet d'entraînement : Lorsqu'une nouvelle application arrive et est mappée sur le système, elle affecte ses nœuds inoccupés voisins en termes de probabilité de congestion et de dispersion. Cela se propage de manière similaire à l'effet d'entraînement de la pierre dans l'eau, de sorte que l'impact diminue avec la distance du point d'impact.

- Une technique de mapping dynamique sensible à la chaleur avec un algorithme de mapping **MinEnergy** a été présentée dans **[42]**. Les candidatures sont prises en compte pour le mapping selon l'approche premier arrivé, premier servi. Si plusieurs applications demandent un mapping à la fois, les applications sont classées en fonction des exigences de communication et de calcul. L'application avec la demande la plus élevée est d'abord mappée sur un nœud avec le plus grand nombre de voisins libres, le reste des applications est mappé autour de lui.

- Une approche dynamique de mapping d'application **Liso** (forme en L isolée) a été proposée dans **[43]**. Liso fournit une communication sans interférence entre plusieurs applications avec des ressources communes en divisant le Noc en différentes régions sécurisées isolées.

- Dans **[44]**, un mapping dynamique des tâches avec une technique de spéculation sur la congestion **DTMCS** a été discuté. Dans **DTMCS** le nombre de tâches de l'application entrante est comparé aux nœuds inactifs disponibles et le processus de mapping est exécuté si le nombre de tâches n'est pas supérieur aux nœuds libres disponibles. La tâche source est placée en premier lieu et la distance de la tâche de destination est décidée en

fonction de la distance de Manhattan. La distance entre deux points est définie comme la distance de Manhattan. L'algorithme proposé peut réduire la congestion globale.

2.2.2 Techniques de mapping statique

Les techniques de mapping d'application statique peuvent être classées en deux branches principales nommées : mapping exacte (mathématique) et mapping basé sur la recherche (heuristique), comme illustré à la **(Figure 26)** le mapping basé sur la recherche peut en outre être classée en recherche déterministe et recherche heuristique.

2.2.2.1 Mapping exact (mathématique)

Le mapping exact ou le mapping basé sur la programmation mathématique offre une solution de mapping optimale.

- Dans **[45]**, les auteurs ont rapporté une technique de mapping d'application basée sur la programmation linéaire mixte en nombres entiers **(MILP)** pour les architectures hétérogènes. Il s'agit d'une combinaison de processus matériels et logiciels (des tâches qui peuvent être implémentées soit dans le matériel soit dans le logiciel ou doivent être subdivisées en sous-tâches) et s'exécute de manière itérative jusqu'à ce que les résultats requis soient atteints. Dans cette technique multiprocesseur, certains processeurs sont spécifiques à l'application, tandis que d'autres sont programmables.

- Dans **[46]**, une approche de mapping en deux phases pour une architecture Noc hétérogène a été présentée. Pour la phase initiale, les IPs sont mappés par une approche de mapping gourmande, et plus tard dans la deuxième phase, les emplacements des IPs sont améliorés en appliquant la recherche-Tabou (tabu-search).

  La recherche-Tabou : est une méthode de recherche-locale introduite comme une technique pour surmonter l'optimalité locale. L'idée de base est de « se souvenir » des solutions qui ont été visitées au cours de l'algorithme, afin d'en

déduire les directions prometteuses pour une recherche ultérieure. Ainsi, pas seulement l'investigation locale du voisinage de la solution courante pilote la recherche mais aussi la mémoire.

- Une technique basée sur **MILP** a calculé la taille et la position des composants et des IPs du réseau. Alors que les techniques **MILP** produisent des solutions de mapping optimales, le principal goulot-d 'étranglement (bottleneck) est le temps d'exécution. Pour minimiser le temps de traitement, une formulation MILP **[47]**, a séparé le graphe de tâches d'application en différents clusters et a défini une topologie personnalisée (une topologie peut être personnalisée à partir d'une ou plusieurs topologies régulières, dont des liens sont ajoutés ou supprimés) **[27]**.

- Le travail présenté dans **[48]** propose une solution de mapping d'application basée sur ILP sensible aux conflits pour minimiser les conflits de réseau. Différents types de conflits de réseau tels que basés sur le chemin (path-based), basés sur la source (source-based) et basés sur la destination (destination-based), comme décrit dans la **(Figure 27)**, sont discutés et le résultat montre que la latence des paquets est améliorée en diminuant les conflits du réseau, mais avec une énergie de communication élevée.

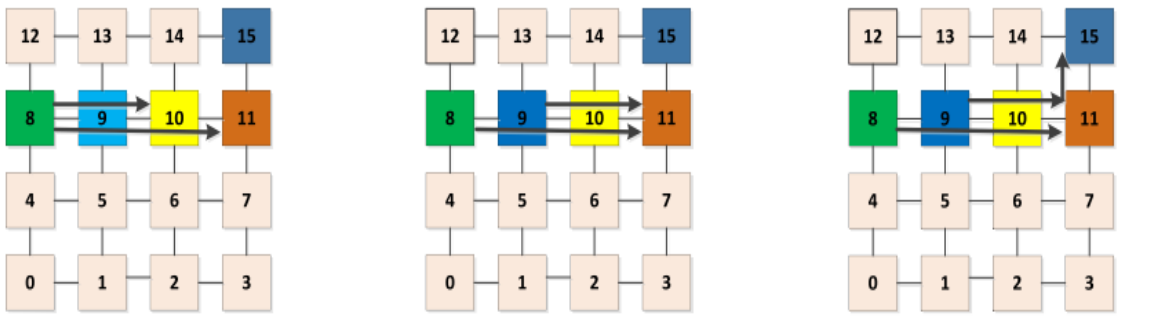

Figure 27 : Différents types de conflits de réseau [48]

- Les techniques basées sur l'ILP définissent de mapping optimal. Cependant, le temps d'exécution est très élevé. Pour répondre au problème du temps d'exécution élevé, une approche basée sur le clustering a été présentée dans **[49]**. Comme dans **[47]**, le réseau maillé est divisé en petites zones ou petites

mailles appelées clusters **[49]**. Les clusters sont mappés sur des petits maillages correspondants, et à la fin tous les petits maillages sont combinés pour obtenir le résultat final. Il a été rapporté que le temps d'exécution s'améliorait avec un inconvénient qui est l'augmentation du coût de communication.

2.2.2.2 Mapping basé sur la recherche

Le mapping basé sur la recherche peut être séparé en deux branches : la recherche systématique ou déterministe et la recherche heuristique.

2.2.2.2.1 La recherche systématique et déterministe

Les algorithmes de mapping d'applications utilisant l'approche Branch-and-Bound **(BB)** appartiennent à la recherche déterministe. Cette approche peut être utilisée pour les petites applications car pour les grandes applications, cela prend plus de temps d'exécution.

- Dans **[50]**, les auteurs ont proposé un schéma basé sur le **(BB)** efficace pour résoudre le problème de mapping des applications sensibles à l'énergie. Ici, l'objectif principal était de réduire l'énergie de communication totale des architectures de réseau sur puce régulières.

- Un IP avec un volume de communication élevé peut entraîner une charge de trafic importante, qui peut ensuite devenir un point-chaud (hotspot) sur les nœuds de routage. Les résultats de points-chauds peuvent provoquer une augmentation de la densité de puissance qui peut perturber la fiabilité du système. Pour résoudre ce problème, de nouvelles interfaces réseau (NI) et un algorithme de mapping IP équilibré en trafic **TBMAP** ont été présentés dans **[51]**. **TBMAP** a utilisé une technique de **(BB)** améliorée et modifiée. Le trafic est équilibré sans compromettre les performances du réseau. Cependant, certains chemins peuvent devenir plus longs pour atteindre un équilibre ce qui peut entraîner des coûts de communication élevés.

- Dans **[52]**, les auteurs ont proposé un algorithme de mapping de force brute multi-objectif segmenté à bande passante contrainte **SBMAP** pour Noc.

SBMAP réduit la complexité de calcul et l'énergie de communication de la conception Noc. Cet algorithme sépare l'application en plusieurs segments et applique une recherche de système modulaire pour trouver la solution de mapping optimisée.

2.2.2.2.2 Recherche heuristique :

Diverses solutions de mapping de recherche heuristique ont été proposées dans Noc. Ces approches peuvent être ordonnées comme des heuristiques transformatrices et constructives.

2.2.2.2.2.1 Heuristique transformative :

L'heuristique transformatrice transforme certaines solutions existantes pour obtenir des solutions améliorées pour le mapping des IPs dans les Nocs. Quelques exemples importants sont l'algorithme génétique **(GA)**, l'algorithme des colonies de fourmis **(ACO)** et l'algorithme des essaims de particules **(PSO)**.

- Dans le travail **[53]**, un mapping d'application basé sur **GA** en deux étapes qui minimise le temps d'exécution a été proposé. Dans la première étape, les tâches d'application sont allouées aux IPs en supposant que tous les retards (delays) de bord sont persistants et égaux au retard de bord global. Dans la deuxième étape, pour réduire le retard total du système, les IPs sont affectés à des tuiles de Noc en fonction du retard réel du modèle de trafic réseau. Les facteurs de retard tels que la longueur des paquets, les conflits de réseau n'ont pas été pris en compte dans ce travail.

- Un mapping basé sur un algorithme génétique **(GA) [54]** prend en compte les facteurs de retard importants mentionnés ci-dessus et effectue le mapping avec un retard moyen réduit. La population correspond aux positions critiques du Noc. Initialement, la population est sélectionnée au hasard et après cela, pour former une nouvelle génération, un croisement multipoint est utilisé avec la population sélectionnée au hasard. La fonction de fitness, dans ce cas, est le temps d'attente moyen. Le nombre d'IPs dans le graph d'IP de contrainte est égal à la taille d'un chromosome. Le chromosome avec le

temps d'attente le plus court participe au croisement. La mutation est exécutée de manière aléatoire pour atteindre des minima locaux ou globaux et ce processus est exécuté de manière répétée jusqu'à ce que le temps d'attente moyen le plus bas reste constant pendant un nombre spécifique d'itérations.

- Un schéma hybride multi-objectifs présenté dans **[55]**, où la technique **PSO** avec l'algorithme du plus court chemin de Dijkstra est utilisée pour décider du chemin le plus court entre les IPs et pour améliorer les performances globales.

- Les travaux proposés dans **[56]** ont calculé le mapping des IPs pour Noc en utilisant la technique **PSO**. Plusieurs applications benchmark en temps réel (real-time) utilisées pour évaluer les performances et les résultats ont été comparées aux approches de mapping des IPs précédentes.

- Selon les travaux présentés dans **[57]**, plusieurs solutions de mapping sont générées et le meilleur mapping est décidé sur la base de trois contraintes de performance. Ce sont le coût de communication, le facteur de contention (facteur de conflit) et l'indice de robustesse.
- Une nouvelle technique de mapping d'IP rapportée dans **[58]**, où l'algorithme **NMAP** modifié en combinaison avec la technique **PSO** est utilisé pour trouver la meilleure solution de mapping.

- Dans **[59]**, les auteurs ont discuté d'une nouvelle technique de mapping des IPs sur le Noc basé sur **BFT** (ButterflyFat-Tree), où ils ont utilisé la stratégie **DPSO** (Discrete Particle Swarm Optimization) pour le mapping des IPs.

- Dans **[60]**, les auteurs ont rapporté une technique hybride pour le mapping des IPs pour Noc en combinaison avec la recherche-Tabu pour élargir l'espace de recherche. L'approche de déviation a forcé les particules d'essaim à découvrir plus d'espace de solution avant de converger vers le meilleur résultat global.

- Une stratégie de mapping en deux phases basée sur **PSO** utilisée pour la conception reconfigurable afin de réduire le coût de communication a été discutée dans **[61]**. Un mapping global est réalisé en fusionnant plusieurs applications au cours de la phase initiale et des multiplexeurs sont utilisés pour reconfigurer en déplaçant les IPs vers des nœuds de routage proches. L'algorithme proposé n'a été comparé à aucune approche existante.

- Les besoins en bande passante ont été minimisés par l'approche de mapping basée sur l'**ACO [62]**, c'est une technique d'optimisation probabiliste inspirée du comportement des fourmis biologiques pour trouver des chemins entre la colonie et une source de nourriture, chaque fourmi fait le mapping selon une probabilité donnée et une table de recherche tabou (tabu-search).

- Dans le travail **[63]**, une approche multi-objectifs est appliquée pour découvrir la solution optimale de mapping. Cette approche utilise une heuristique basée sur un algorithme de colonie de fourmis multi-objectifs pour explorer l'espace de mapping et trouver le front de Pareto optimal qui optimise la consommation d'énergie et la température du point-chaud (hotspot).

2.2.2.2.2.2 Heuristique constructive

Dans ce type d'heuristique, des solutions partielles sont générées successivement et à la fin, la solution décisive pour le mapping d'IP est atteinte. L'heuristique constructive est de deux types principaux, les heuristiques avec amélioration itérative (a) et les heuristiques sans amélioration itérative (b).

a) Heuristique constructive avec amélioration itérative

Dans cette approche, une solution préliminaire sur certains critères prédéfinis est utilisée pour mapper les IPs du graphe d'IP sur un graph de topologie. Après le placement initial, une procédure d'amélioration itérative est effectuée sur le mapping initial pour obtenir la solution finale améliorée.

- Dans **[64]**, une méthode de mapping triphasé pour minimiser le délai de communication moyen (communication delay) a été proposée et qui satisfait la contrainte de bande passante.

- Une approche heuristique constructive en temps polynomial nommée **MOCA** pour les architectures Noc à base de maillage avec de faibles besoins en énergie a été présentée dans **[65]**. Il est rapporté que l'algorithme **MOCA** est moins complexe que celui de **NMAP [64]**.

- Une approche de mapping basée-sur-les-clusters combinée à un recuit simulé a été présentée dans **[66]**. Initialement, le mapping basé sur les clusters est effectué puis amélioré en appliquant une approche de recuit simulé pour dériver une solution de mapping finale.

- Dans **[67]**, un algorithme heuristique constructif à bande passante moins complexe **(Onyx)** a été proposé. Dans cette technique, dans un premier temps, l'IP ayant la bande passante de communication maximale est placé au centre du réseau. Les IPs à mapper ensuite sont classés en fonction de leur demande de communication avec les IPs précédemment mappés.

- Dans **[68]**, des listes de priorités sont créées en fonction de la bande passante de communication et du degré d'interconnexion entre les nœuds avant le mapping. Les tâches sont ensuite mappées en zigzag à partir d'un coin et se terminent sur un autre coin du Noc basé sur le maillage.

- L'algorithme **LMAP** a été rapporté dans **[69]**. Un schéma a été utilisé pour découvrir le placement des IPs en examinant leur demande de bande passante. La phase d'amélioration itérative sert ensuite à affiner le mapping préliminaire.

- Le travail présenté dans **[70]**, propose d'extraire un graphe abstrait du graphe de base d'IP. Une heuristique a utilisé ce graphe abstrait pour le mapping afin d'améliorer la latence.

b) Heuristique constructive sans amélioration itérative

Le mapping est effectué sur la base de certains critères prédéfinis en affectant les IPs (un par un) sur le graph de topologie Noc et le placement initial des IPs reste inchangé.

- **PMAP** un algorithme de mapping en deux étapes a été proposé dans **[71]**, où les clusters ayant une communication élevée sont affectés sur des nœuds parallèles.

- **SMAP [72]** est un algorithme de mapping d'IP basé sur la simulation qui offre une réduction de l'énergie de communication et du temps d'exécution. Dans cette stratégie, le IP prioritaire est placé au centre et les autres IPs sont placées en spirale à partir de l'IP initialement mappé jusqu'à la limite du Noc.

- Un schéma de mapping a été proposé dans **[73]**. Les IPs sont classés en fonction de la bande passante de communication entre eux. Sous réserve du classement du IPs, les ensembles de IPs avec la communication la plus élevée sont fusionnés deux par deux à chaque itération.

- L'approche de mapping **RMAP [74]** envisage des erreurs transitoires survenant dans les commutateurs et les canaux du Noc causées par la propagation des données sur les canaux. Dans **RMAP**, le graph d'IP est divisé en deux sous-domaines et les canaux avec moins de charges de communication sont utilisés pour acheminer les paquets de manière redondante afin d'assurer la fiabilité.

- **CastNet [75]** a maintenu une liste de signification des IPs d'application basée sur la bande passante de communication globale et moyenne. Le mapping initial est effectué sur la base d'une liste de priorités. Les IPs initiaux à mapper sont sélectionnés parmi différents groupes de symétrie. Cette procédure a été répétée pour le reste des IPs. Un ensemble de solutions de mapping est créé et la liste de priorités est mise à jour après chaque mapping.

Au final, parmi l'ensemble de solutions de mapping, la meilleure solution sélectionnée comme solution de mapping d'IP finale.

- Dans **[76]**, l'arbre couvrant maximal est construit en extrayant la partie la plus importante de l'application en fonction du coût de communication moyen. Le mapping est décidé en fonction des effets du coût de communication, du chemin le plus long vers les descendants et du degré du noyau. Enfin, la tuile la plus appropriée à mapper est sélectionnée, celle qui a le plus de voisins libres et de connexions disponibles.

- L'algorithme en deux étapes **mapGtoM** a été proposé dans **[77]** en 2019. Un mapping d'IP constructif est effectué en sélectionnant les tuiles avec une symétrie de tuiles comme tuiles initiales pour le mapping et les IPs voisins sont mappés proches les uns des autres. Plusieurs solutions de mapping sont générées et la meilleure solution est sélectionnée en fonction de l'énergie de communication totale.

2.3 Le mapping dans les topologies 3D

2.3.1 La technologie 3D

La technologie d'intégration 3D est une nouvelle façon d'augmenter les performances du système sans mise à l'échelle en raison de plusieurs caractéristiques de l'intégration 3D, nous mentionnons ce qui suit :

- La longueur de fil réduite.
- Un délai d'interconnexion réduit.
- Un nombre accru d'interconnexions entre couches.

Plusieurs puces en silicium sont empilées et connectés en utilisant les interconnexions verticales. La longueur, le retard, la puissance et la consommation d'énergie des interconnexions verticales et horizontales est généralement asymétrique. Les interconnexions verticales sont plus performantes que les horizontales. Pour sa propriété inhérente, la technologie de l'intégration 3D peut être un bon choix pour concevoir un système complexe basé sur le Noc 3D **[78-79]**.

2.3.1.1 Les avantages de 3D par rapport à 2D

Dans les réseaux 3D, la longueur des liaisons physiques reliant les nœuds du réseau devient plus courte (ce qui réduit la latence et la puissance), et l'acheminement des données à travers le réseau passe par un nombre de sauts réduit, ce qui améliore les performances du réseau sur puce 3D **[80].**

L'utilisation des liens verticaux (TSV) entre les couches offrent une bande passante plus élevée et une faible latence.

La recherche **[81]** montre que le Noc 3D consomme moins d'énergie par paquet par rapport au Noc 2D. Il y a une diminution de la consommation d'énergie lorsque la dissipation de puissance et la latence est diminuée ce qui conduit à réduire le nombre de sauts.

2.3.1.2 Through-Silicon-Vias (TSVs)

Le paradigme **Noc 3D** est considéré comme l'un des architectures les plus avancées. Les **TSVs** constituent l'un des principaux supports de communication inter-couches. Elles fonctionnent comme des liens intermédiaires dans les **Noc 3D** et elles fournissent une bande passante d'interconnexion élevée et une densité d'intégration élevée **[82]**.

2.3.1.3 Les topologies 3D standard

Nous mentionnerons ci-dessous les topologies 3D qui seront utilisées pour le mapping des applications :

2.3.1.3.1 Topologie 3D mesh

La topologie 3D mesh a plus d'un 2D mesh dans lequel les nœuds sont reliés par des liaisons verticales (TSV). Un montant supplémentaire de la dimension 'Z' est ajouté aux 'X' et 'Y' dans la dimension en général.

Les Noc 3D peuvent être utilisés pour construire des systèmes hétérogènes qui offrent également un meilleur rendement à faible coût. Le mécanisme de tolérance aux pannes est la meilleure technique pour surmonter l'imprévisible qui se produit dans les Noc 3D. Les circuits intégrés 3D réduisent la longueur et le nombre d'interconnexions, ce qui réduit à son tour le retard de fil et augmente

la localité spatiale avec des interconnexions verticales (augmentation au niveau de surface).

Dans **[83]** on déduit une technique pour diminuer le nombre de TSV **(voir la figure 28).** La paire de canaux verticaux unidirectionnels est remplacée par un canal bidirectionnel, qui peut être dynamiquement reconfiguré en sortant ou en entrant. Cependant les performances se dégradent considérablement.

Dans **[84]**, ont conseillé un mécanisme de tolérance aux pannes pour les liaisons multi-bits basées sur les TSV, qui sont des nœuds de routage pour les liaisons Noc 3D **[85]**.

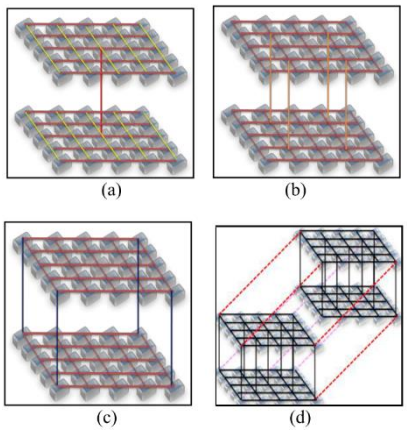

Figure 28 : Défèrent technologie de topologie 3D mesh (2021)

- **a** : Modèle Noc 3D à 2 couches et un seul canal vertical (TSV) au centre de la liaison sphérique.
- **b** : Modèle Noc 3D à 2 couches et 4 canaux verticaux (TSV) au milieu des liaisons de la sphère.
- **c** : Modèle Noc 3D à 2 couches et 4 canaux verticaux (TSV).
- **d** : Modèle Noc 3D à 4 couches (2 modèle Noc 3D) chaque couche reliée avec TSV **[86]**.

2.3.1.3.2 3D OASIS-NoC

L'architecture OASIS 3D-NoC est préférée aux implémentations 2D-NoC pour les deux raisons énoncées ci-dessous :

OASIS-NoC dispose d'un réseau à topologie maillée 4x4 avec un nombre maximum de cinq ports, un pour chaque direction (local, nord, sud, est et ouest) utilisant la commutation par chaque interrupteur. Chaque commutateur est désigné par ses coordonnées XY (x-addr et y-addr) dans le réseau. Ces adresses 3 bits sont embarquées dans chaque flit d'entrée 76 bits portant des

informations sur la direction du port suivant (Next-Port 5 bits). La queue indique la fin d'un paquet (1 bit), et la charge utile (64 bits).

Une bande passante globale inférieure est obtenue par rapport à un maillage 3D complet en raison de la présence de plusieurs blocs IP par commutateur **[81]**.

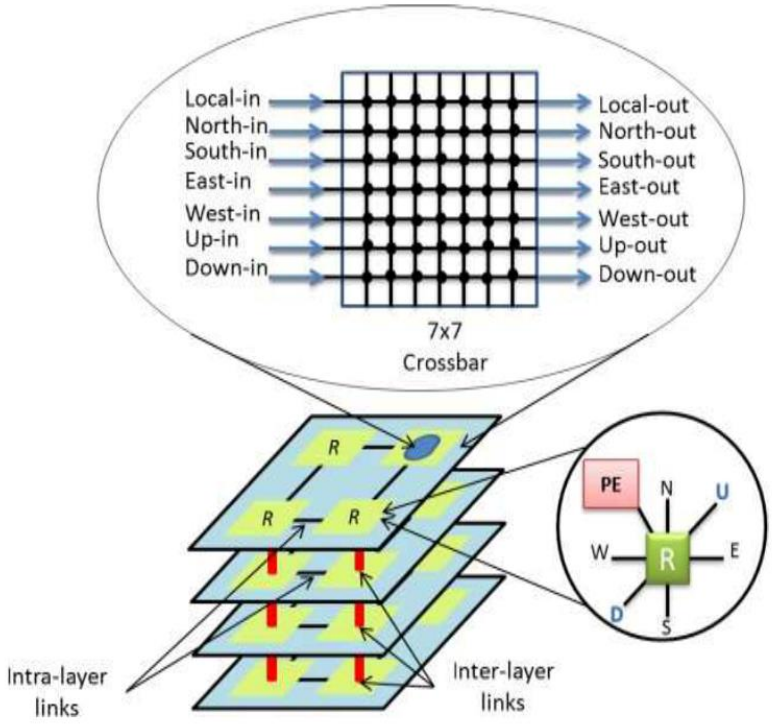

Figure 29 : OASIS 3D NoC

2.3.1.3.3 3D Torus

La topologie torus est similaire à une topologie maillée qui a des bords enveloppants connectés aux nœuds terminaux, comme illustré sur la **figure 30** ci-dessous **[81]**.

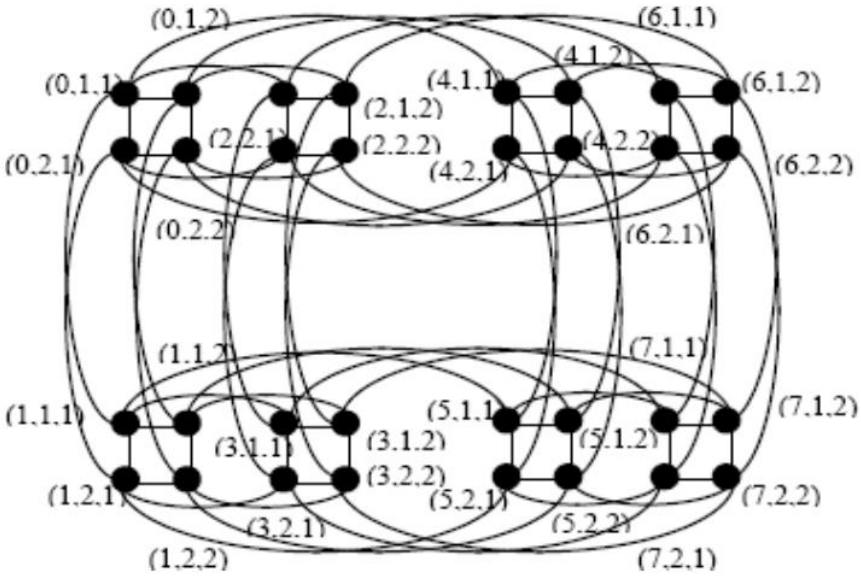

Figure 30 : 3D Torus

Le diamètre global de la topologie est réduit et l'utilisation de fils plus courts réduit le nombre de sauts et le coût du réseau est la moitié du maillage. Il est difficile d'implémenter la mise à l'échelle dans cette topologie. Le taux de progression de la topologie 3D-torus est supérieur à celui du 2D-torus.

2.3.1.3.4 Architecture Hypermesh 3D

La topologie Hyper-mesh est une combinaison de deux topologies d'interconnexion, à savoir l'hypercube et le maillage, qui présentent les caractéristiques des deux (**voir la figure 31**). La topologie hypercube à une large bande passante avec une structure récursive et a une tolérance maximale aux pannes **[81]**.

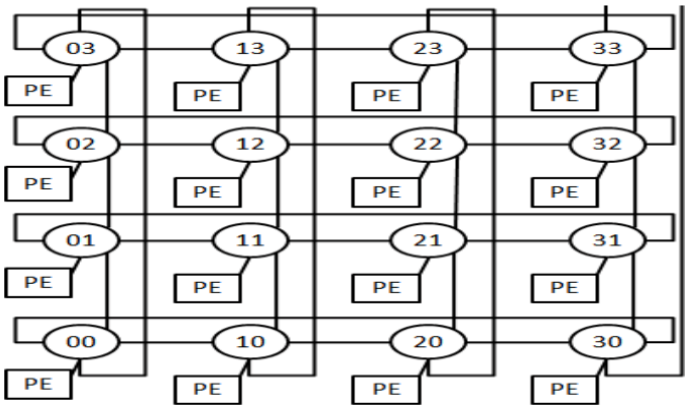

Figure 31 : Architecture Hypermesh 3D

2.3.2 Le mapping des applications sur les topologies 3D

Toutes les techniques de mapping mentionnées précédemment sont applicables aux systèmes basés sur Noc 2D. Ces techniques peuvent également être étendues aux systèmes basés sur Noc 3D. La **(figure 32)** représente un exemple de mapping sur une architecture 3D.

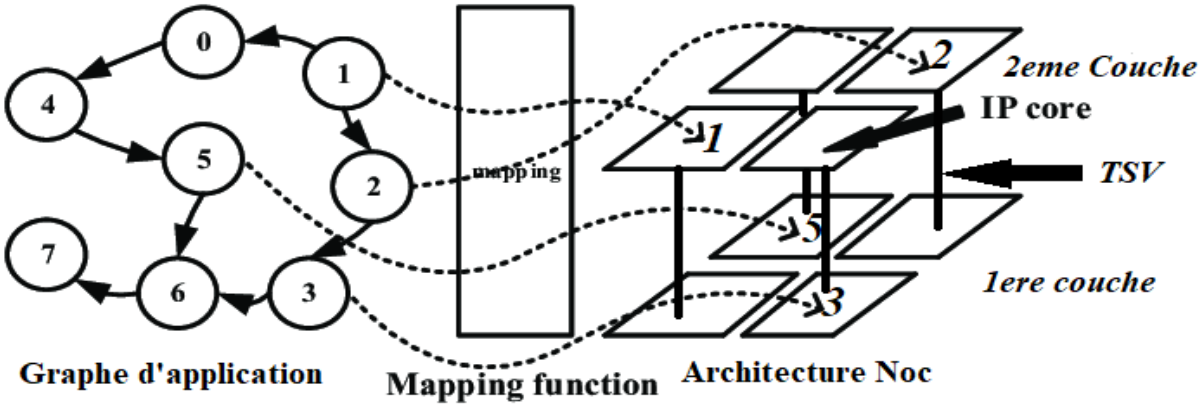

Figure 32 : Exemple de mapping sur l'architecture 3D Noc

- Un algorithme de mapping d'IP sensible à l'énergie pour les Nocs 3D hétérogène est présenté dans **[86]**.

- Dans **[87]**, un algorithme de mapping d'exécution efficace pour réduire à la fois la latence de communication et le temps d'exécution global de l'application sous contrainte thermique pour les Nocs 3D est proposé.

- Le travail dans **[88]** présente une approche multi-objectif basée sur la connaissance **KBMA** pour un mapping réussi des blocs IP aux tuiles Noc 3D avec des architectures Noc standard comme le maillage (mesh), le tore (torus), l'anneau (ring) et le BFT.

3. Synthèse :

Le tableau suivant montre une comparaison entre les caractéristiques des travaux trouvée dans la littérature sur la phase de mapping.

Tableau 3.1 : Caractéristiques des travaux sur la phase de mapping

Réf	Classe de Mapping	Méthode utilisée [40]	Objectif	Topologie visée	Les performances considérées [40]
[41]	Dynamique	MapPro	mono-objectif	2D Mesh	Réduction de la latence moyenne
[42]	Dynamique	MinEnergy	Multi-objectif	2D Mesh	Energie, latence moyenne et débit
[43]	Dynamique	Liso	mono-objectif	2D Mesh	Réduction des interférences de communication
[44]	Dynamique	DTMCS	mono-objectif	2D Mesh	Réduction de la latence moyenne
[45]	Statique	MILP	mono-objectif	Standard hétérogène	Amélioration de temps d'exécution
[46]	Statique	MILP	Multi-objectif	Personnalisé	Amélioration de la surface et de la puissance
[47]	Statique	MILP	mono-objectif	Personnalisé	Minimisation de la consommation d'énergie
[48]	Statique	ILP	mono-objectif	2D Mesh	Réduction de la latence de paquet
[49]	Statique	ILP	mono-objectif	2D Mesh	Réduction de temps d'exécution
[50]	Statique	(BB)	mono-objectif	2D Mesh	Réduction de l'énergie de communication totale
[51]	Statique	(BB) TBMAP	mono-objectif	2D Mesh	Equilibrage de la charge de trafic
[52]	Statique	(BB) SBMAP	Multi-objectif	2D Mesh 2D Torus	Réduction de la consommation

					d'énergie et de cout de communication
[53]	Statique	GA	mono-objectif	2D Mesh	Minimise le temps d'exécution
[54]	Statique	GA	mono-objectif	2D Mesh	Minimise le temps d'exécution
[55]	Statique	PSO	Multi-objectif	2D Mesh	Réduction du temps d'exécution et de l'énergie
[56]	Statique	PSO	Multi-objectif	2D Mesh	Amélioration de la latence du réseau, du débit et de cout de communication
[57]	Statique	PSO	mono-objectif	2D Mesh	Minimisation de cout de communication
[58]	Statique	PSO	mono-objectif	2D Mesh	Réduction de cout de communication
[59]	Statique	DPSO	mono-objectif	2D Mesh Tree	Réduction de cout de communication
[60]	Statique	DPSO	mono-objectif	2D Mesh	Réduction de cout de communication
[61]	Statique	PSO	mono-objectif	2D Mesh Tree	Réduction de cout de communication
[62]	Statique	ACO	mono-objectif	2D Mesh	Réduction de la bande passante de communication
[63]	Statique	MOACA	Multi-objectif	2D Mesh	Amélioration de la consommation d'énergie et de la température des points-chaud
[64]	Statique	NMAP	Multi-objectif	2D Mesh	Amélioration de la bande passante et du cout de communication
[65]	Statique	MOCA	mono-objectif	2D Mesh	Réduction de la consommation d'énergie

[66]	Statique	CSA	mono-objectif	2D Mesh	Réduction de cout de communication
[67]	Statique	Onyx	mono-objectif	2D Mesh	Amélioration de cout de communication
[68]	Statique	Crinkle	Multi-objectif	2D Mesh	Amélioration de cout de communication, de la consommation d'énergie et du temps d'exécution
[69]	Statique	LMAP	Multi-objectif	2D Mesh	Amélioration de cout de communication et de la latence moyenne du réseau
[70]	Statique	XY-ADB	mono-objectif	2D Mesh	Amélioration de la latence moyenne
[71]	Statique	PMAP	mono-objectif	2D Mesh 2D Ring 2D Tree	Amélioration de cout de communication
[72]	Statique	SMAP	mono-objectif	2D Mesh	Réduction de l'énergie de communication
[73]	Statique	BMAP	Multi-objectif	2D Mesh	Amélioration de l'équilibre de la charge de trafic et de nombre de sauts
[74]	Statique	RMAP	mono-objectif	2D Mesh	Réduction de délai moyen de livraison des paquets
[75]	Statique	CastNet	mono-objectif	2D Mesh	Amélioration de cout de communication
[76]	Statique	CHMAP	mono-objectif	2D Mesh	Amélioration de cout de communication
[77]	Statique	mapGtoM	Multi-objectif	2D Mesh	Amélioration de l'énergie de communication et du temps CPU
[86]	Statique	----------	mono-objectif	3D	Amélioration de la consommation d'énergie

[87]	Dynamique	----------	Multi-objectif	3D	Réduction de la latence de communication et du temps d'exécution
[88]	Statique	KBMA	Multi-objectif	3D 2D Torus 2D Ring 2D BFT	Amélioration de La puissance, la surface et le délai

D'après les travaux de recherche mentionnés dans le tableau ci-dessus, nous notons que :

❖ Concernant le type de mapping

- Le mapping dynamique se fait en temps réel. Dans ce cas, le mapping consiste uniquement à joindre une ou plusieurs tâches de l'application aux tuiles de l'architecture ou IPs. C'est-à-dire, au moment de l'exécution de l'application, les tâches sont mappées aux IPs.

- Dans le mapping statique, le placement se fait lors de la conception de l'architecture. Dans ce cas, le mapping consiste à affecter chaque IP de l'application à une et une seule tuile de l'architecture.

Le **(tableau 3.2)** montre une comparaison entre le mapping statique et dynamique.

Tableau 3.2 : Comparaison entre mapping statique et mapping dynamique [89]

	Mapping statique	**Mapping dynamique**
Mapping des IP	**+**	**_**
Mapping des tâches	**+**	**+**
Le mapping est exécuté au moment de la conception	**+**	**_**
Le mapping est exécuté au moment de l'exécution	**_**	**+**
Topologie standard	**+**	**+**
Topologie personnalisée au moment de la conception	**+**	**_**
Mono-application ou multi-applications	**+**	**+**

❖ Concernant le choix de topologie

- La topologie peut suivre la classification suivante : régulière ou irrégulière, directe ou indirecte et finalement standard ou bien personnalisé.

- La topologie 2D Mesh est la plus utilisé dans les réseaux sur puce. Ceci est dû à la facilité de son implémentation et son indépendance de la taille du réseau.

- La topologie Noc standard est utilisée pour différents applications grâce à son évolutivité et sa flexibilité.

- Les topologies Noc personnalisées est souvent associé à des applications spécifiques.

Le **(tableau 3.3)** montre les avantages et les inconvénients des différents types de topologies.

Tableau 3.3 : Comparaison entre les topologies standards et les topologies 3D personnalisées

	Avantage	Inconvénient
2D Standard	- La réutilisation. - la facilité de son implémentation.	- Non scalable. - Performance limité par rapport au topologie 3D.
3D Standard	- Scalabilité. - La réutilisation.	- La surface. - La difficulté de son implémentation.
3D Personnalisée	- Scalabilité. - Amélioration de performance par rapport au topologie 2D. - Réduction de surface.	- Non réutilisable. - La difficulté de son implémentation.

❖ Concernant la phase de mapping

Les méthodes exactes comme **MILP** et **ILP** et les méthodes déterministes construites autour de **BB** comme **TBMAP** et **SBMAP** pour des applications avec un grand nombre de nœuds prennent trop de temps de calcul pour générer la solution optimale, ce qui signifie que ces méthodes ne sont pas pratiques pour les Noc ayant un grand nombre d'IPs.

Les heuristiques constructives sont les plus rapides et peuvent être plus utiles lorsque le temps d'exécution est crucial : Les heuristiques constructives avec amélioration itérative comme **NMAP**, **MOCA**, **CSA**, **Onyx**, **Crinkle**, **LMAP**, **XY_ADB** dépendent fortement de la solution initiale pour l'améliorer itérativement et produire la solution finale. En revanche les heuristiques constructives sans amélioration itérative comme **PMAP**, **SMAP**, **BMAP**, **RMAP**, **CastNet**, **CHMAP**, **mapGtoM** fonctionnent avec certains critères prédéfinis pour parvenir à une bonne solution.

Les heuristiques transformatives comme **GA**, **PSO**, **DPSO**, **ACO**, **MOACA** obtiennent de meilleures solutions la plupart du temps dans un temps d'exécution raisonnable. Elles permettent une meilleure exploration de l'espace de recherche, car de multiples solutions évoluent simultanément avec des interactions mutuelles entre elles. Une observation générale à propos de l'**ACO** est qui est une technique d'optimisation probabiliste qui utilise une probabilité de distribution pour distribuer les IPs aux tuiles. L'**ACO [62]** utilise une probabilité de distribution qui dépend aux phéromones **(voir l'optimisation par les colonies de fourmis ACO de chapitre 2)** uniquement.

Nous proposons dans le cadre de notre travail de traiter en parallèle la phase de mapping et la personnalisation de la topologie en considérant la technologie 3D. Notre idée considère plusieurs objectifs à optimiser en même temps : le coût de communication (délai/énergie) pendant le placement des composants (mapping). Ainsi que, la surface du réseau en optimisant le nombre de liens verticaux qui permettra au final de spécifier une topologie 3D partiellement connectée.

4. Conclusion

Dans ce chapitre nous avons présenté un état de l'art sur la phase de mapping dans les réseaux sur puce. On a abordé le processus de mapping. La classification des techniques de mapping disponibles en différentes catégories et présentée en fonction de leurs algorithmes de mapping proposés dans la littérature, les techniques de mapping dynamique sont présentées tandis que les approches de mapping statique sont discutées en détail. Dans la dernière partie de ce chapitre nous avons introduit le principe du réseau sur puce 3D, nous avons cité les avantages qu'il présente par rapport à la structure classique 2D et aussi nous avons présenté quelques approches de mapping des IPs sur les réseaux sur puce 3D.

La synthèse présentée dans ce chapitre nous a permet de concrétise et de proposer une nouvelle solution (algorithmique, application) qui sera détaillée dans le chapitre suivant.

Chapitre 4

Conception de la technique proposée

1. Introduction

La topologie la plus adoptée par la plupart des travaux de recherche est la topologie Mesh à 2 dimensions, mais cette topologie a montré des insuffisances surtout dans le cas d'un réseau de grande taille **(voir la synthèse de chapitre 3)**. Ces insuffisances pourraient être contournées par la nouvelle technologie des circuits intégrés à 3 dimensions **(3D IC).** Toutefois cette technologie a aussi montré ses limites au niveau de la technologie de fabrication dû à l'emploi massif des **TSVs** nécessaires à la communication inter- couches.

Notre proposition consiste à traiter en parallèle la phase de mapping et la personnalisation de la topologie en utilisant la technologie 3D. Cette problématique est considérée comme un problème d'optimisation multi objectifs, de ce fait, nous allons utiliser une méthode d'optimisation multi-objectif, basé sur l'**ACO** et à l'approche **Pareto** afin de trouver des solutions de compromis (non dominées) qui optimisent toutes nos fonctions objectives.

2. Formulation du problème

Traiter la phase de mapping avec la personnalisation de la topologie en considérant la technologie 3D est classé comme un problème d'optimisation combinatoire multi-objectifs vu le nombre des critères qu'on cherche à optimiser.

2.1 Graphes d'application et d'architecture

La modélisation de l'application en tant qu'IP exécutant les tâches et les ressources d'une architecture est réalisée sous forme de graphes dirigés. Le

mapping implique de placer les composants du graphe d'application (IP) à chaque sommet du graphe d'architecture (Tuile) tout en minimisant les chemins de communication entre les IPs **[13, 91].**

Un graphe d'application (APG)

G = G (C, A) est un graphe où chaque lien est affecté d'un poids qui représente le volume de communication (nombre de paquets envoyés) entre *ci* à *cj*.

Un graphe d'architecture (ARG)

G = G (T, P) est un graphe où chaque vertex *ti* représente une tuile de l'architecture, et chaque arc directionnel *pij* représente le chemin de routage entre *ti* et *tj*.

En utilisant les deux graphiques, le problème du mapping est d'assigner chaque IP de l'application à une tuile d'architecture afin de minimiser le coût des communications. Par conséquent, les IP qui ont un volume de communication élevé doivent être placés aussi près que possible les uns des autres. La **(figure 25)** montre un exemple sur la façon de mappé le graphe d'IP sur un graphe d'architecture.

2.2 Les fonctions objectives

Deux fonctions objectives seront traitées dans notre travail, qui sont : le coût de communication et la surface consommée.

$x \in X = [x_1, x_2, \dots\dots, x_n]$, tels que $F(x) =$ le coût de communication à optimisé.

-X est ensemble des solutions de mapping générer par l'algorithme.

$y \in Y = [1, Max(TSV)]$, tels que $S(x) =$ le coût en surface à optimisé.

-Y représente le nombre des liens verticaux dans la topologie du réseau. Où :

$$\text{Max(TSV)} = \frac{\text{Nombre Total Des IPs}}{2}$$

2.2.1 Le coût de communication F(x) :

La fonction objective **F(x)** du coût de communication qui assure la résolution du problème du mapping (placement des IPs) d'une manière optimale.

$$F(x) = \min\{Cout : \text{ cout de communication entre les IPs}\}$$

Où le coût des communications pour chaque solution est calculé comme suit :

$$\mathbf{Cost} = \sum_{\mathbf{K=1}}^{|\mathbf{E}|} \mathbf{value}(\mathbf{e_{(i,j)}}) * \mathbf{hopcount}(\mathbf{t_i}, \mathbf{t_j}) \qquad \textbf{(1)}$$

Ici, **|E|** représente le nombre d'arêtes dans le graphe d'IP.

$value(d^k)$, représente le volume de transfert entre les IPs i et j qui sont mappés aux tuiles respectivement t_i et t_j.

$hopcount(t_i, t_j)$ **,** est le nombre de sauts entre les nœuds de topologie t_i *et* t_j.

Le $hopcount(t_i, t_j)$ qui peut être calculé sur deux façon :

- Quand la communication est entre deux tuiles qui se trouvent dans le même plan, la formule mathématique est :

$$hopcount(t_i, t_j) = dManhattan(t_i, t_j) = |x_2 - x_1| + |y_2 - y_1|$$

- Quand la communication se fait entre deux tuiles qui se trouvent dans différents plans la formule mathématique

$$hopcount = dManhattan(t_i, t_k) + dManhattan(t_k, t_j) + \alpha$$

Avec t_k est la position du lien vertical et α est le coût du lien vertical.

Puisqu'un TSV offre des liaisons verticales courtes et rapides par rapport au fil 2D long, le coût de communication verticale est inférieur au coût de communication horizontale selon **[91, 92]** est :

$$\text{Cout Communication} = \text{Cout Communication Horizontal} + \alpha\ \text{Cout Communication Vertical}$$

Nous avons fixé dans la phase de test les valeurs de α à 1 et 0.8.

2.2.2 Le coût de surface S(x) :

La fonction objective **S(x)** indique la surface utilisée par les **TSVs** de l'architecture qui assure l'utilisation d'un minimum nombre de liens verticaux (nombre **TSV**).

$$S(x) = \min\{\text{Surf: surface de silicium utilisé} = \text{optimisé nombre des liens vertical}\}$$

3. Supposition

Notant que les critères physiques de positionnement de tuiles, liens, l'espace à respecter entre les couches empilées, taille des routeurs, commutateurs, sont ignorés on s'intéresse seulement au coût de communication (énergie, latence) et au coût de surface (nombre de TSV).

Nous avons considéré un réseau 3D constitué de 2 plans seulement (niveaux) reliés par des liens verticaux. Le nombre de plans est limités en raison des problèmes thermiques engendrés par l'empilement de ces plans **[93]**. Chaque plan contient un nombre de tuiles (tel qu'une tuile est un placement candidat d'un lien vertical) qui est égal à la moitié de nombre de tuiles totale de la topologie.

Les IPs sont mappés sur les tuiles suivant une topologie Mesh (mapping), et les TSVs sont affectées aux tuiles (candidates) qui sont dans des positions identiques et qui se trouvent dans différents plans (personnalisation de topologie).

La **(Figure 33)** représente un exemple illustrant une application à 8 IPs mappé sur une topologie 3D en utilisant 2 TSVs.

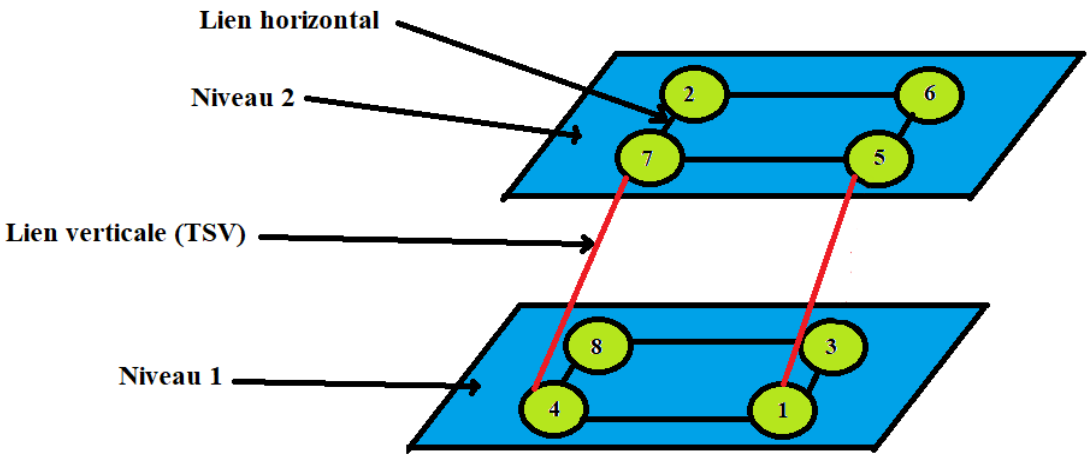

Figure 33 : Représentation explicatif de notre supposition

4. L'optimisation par colonie de fourmis

ACO est un algorithme basé sur la population, il exploite une population de fourmis pour explorer les régions prometteuses de l'espace de recherche. Les fourmis sont capables de résoudre collectivement des problèmes complexes, tel que le problème de plus court chemin. Pour cela, elles communiquent entre elles de façon locale et indirecte, grâce à une hormone volatile, appelée phéromone : au cours de leur progression, les fourmis déposent une trace de phéromone, elles choisissent ensuite leur chemin de façon aléatoire, selon une probabilité dépendant de la quantité de phéromone précédemment déposée.

4.1 Algorithme ACO classique

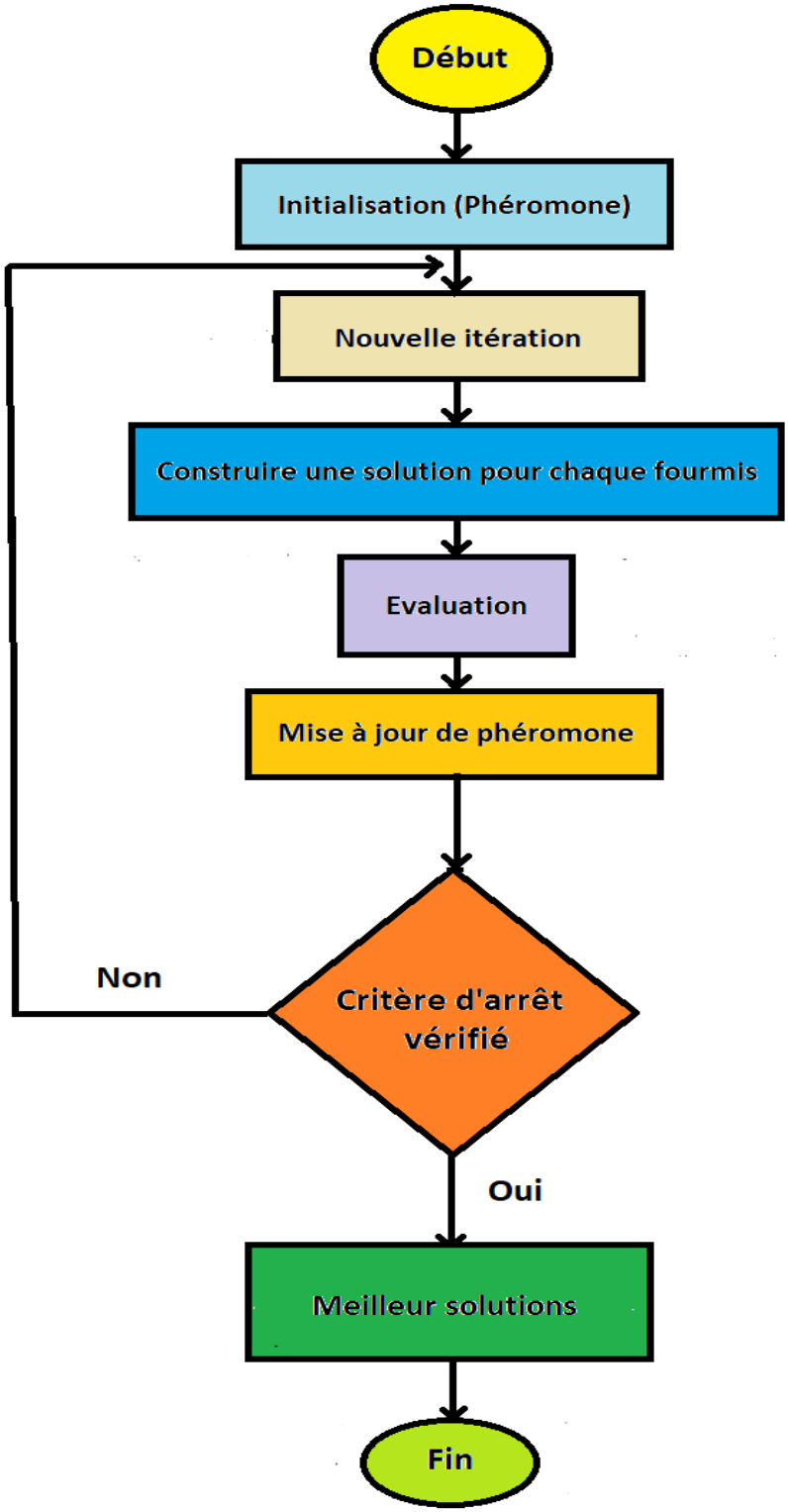

Figure 34 : Schéma de principe de l'algorithme ACO

Début

Initialiser les paramètres et Ia taille K de la colonie de fourmis ;

Pour chaque itération cycle ;

Pour chaque fourmi k ;

Construire une solution-k

Calculer f(X) de solution-k

Fin pour

Mise à jour de phéromone

Fin pour

Retourner les meilleures-solutions

Fin

4.2 L'application de l'ACO à un problème

Afin d'utiliser **ACO** efficacement, plusieurs paramètres sont mis en jeu, ils doivent être définis avec succès. Parmi ces paramètres, on peut citer essentiellement :

4.2.1 La fonction objective

Elle dépend également du problème. Elle représente un moyen pour mesurer la qualité de chaque solution. Cette fonction doit être capable de créer un ordre total dans l'espace des solutions.

4.2.2 La taille de la population

Ce paramètre a une influence sur le comportement de l'algorithme. En effet, une petite population ne crée pas assez d'interactions garantissant le bon fonctionnement de l'algorithme.

4.2.3 Le paramètre α

Pour représenter le degré de l'importance des informations dans le chemin.

4.2.4 Le paramètre β

Pour représenter le degré d'importance de facteur directeur.

4.2.5 Le paramètre ρ

Pour représenter le degré d'information qui diminuent avec le temps.

4.2.6 Le paramètre τ (0)

Pour représenter la valeur initiale de l'intensité des phéromones.

4.2.7 Critère d'arrêt

Le critère d'arrêt diffère suivant le problème d'optimisation posé et les contraintes de l'utilisateur. De ce fait, il est fortement conseillé de doter l'algorithme d'une porte de sortie en définissant un nombre maximum d'itération. L'algorithme doit alors s'exécuter tant que l'un des critères de convergence suivant n'a pas été atteint **[94]**. Cela peut être :

- Un nombre fixe d'itérations.
- En fonction de la fonction objectif.

5. Optimisation multi-objectif par colonie de fourmis

L'algorithme **ACO** classique doit être modifié pour être adapté à la résolution des problèmes d'optimisation multi-objectifs. Comme on a vu dans le **(chapitre 2),** l'ensemble de solutions d'un problème multi-objectifs ne se compose pas d'une seule solution, pour trouver cet ensemble de solutions on va utiliser l'approche **Pareto** qui cherche l'ensemble de Pareto optimale.

6. Résolution du problème

6.1 Présentation de la méthode ACO proposée

6.1.1 Représentation d'une solution (fourmi)

Une solution dans le problème de mapping et de personnalisation de topologie peut être formulée par deux vecteurs d'entier de mêmes tailles, où : le premier

vecteur correspond au mapping des cœurs IPs sur les tuiles (les tuiles sont numérotées dans l'ordre croissant du coin supérieur gauche au coin inférieur droit et la numérotation augmente de la couche la plus basse à la couche la plus élevée) et le deuxième vecteur à l'affectation de TSVs sur des positions candidates. Par exemple, la **(Figure 35)** montre une application de 8 cœurs mappé sur une topologie Mesh **2×2×2** avec 2 TSVs (le même exemple illustré dans la **(Figure 33)**).

	Plan 1				Plan 2			
Indices de vecteurs (Tuile)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Vecteur-1 (IP)	**8**	**3**	**4**	**1**	**2**	**6**	**7**	**5**
Vecteur-2 (TSV)	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**

Figure 35 : Représentation d'une solution

Dans cet exemple : **vecteur-1[i]** note le noyau mappé sur la tuile **i**, et **vecteur-2[i]** note s'il existe un TSV sur la tuile **i**.

6.1.2 Le fonctionnement de l'ACO [95]

L'algorithme **ACO**, est implicitement paramétré par le problème à résoudre. Nous considérons que chaque fourmi construit une solution à travers d'assigner chaque IP de l'application à une tuile d'architecture. Les traces de phéromone sont associées aux assignements d'IP dans les tuiles.

ACO est aussi paramétré par le nombre de fourmis et le nombre de structures de phéromone considérées. Les traces de phéromone sont initialisées d'abord à τ_0. Par la suite, à chaque **cycle** chaque fourmi construit une solution et les traces de phéromone sont mises à jour. Pour éviter une convergence prématurée, les traces de phéromone sont bornées avec deux bornes τ_{min} et τ_{max}. L'algorithme s'arrête lorsqu'un nombre maximum de **cycles** est atteint.

6.1.2.1 Construction d'une solution

6.1.2.1.1 Placement des TSVs

Pour le placement des TSVs, la procédure de résolution est la suivante : Une **Tuile** est sélectionnée d'un ensemble facultatif (l'ensemble de tuiles candidates) en fonction de la probabilité **(2)** pour affecter le **TSV1**, et la tuile est ajoutée a une liste **Tabou**. Ensuite une autre **Tuille** est sélectionné en fonction de la probabilité précédente pour affecter le **TSV2**, et aussi la **Tuile** est ajoutée à la liste **Tabou**. l'étape a été implémentée jusqu'à ce que tous les **TSVs** aient été affectés aux tuiles et que la liste **Tabou** soit pleine. Dans un **cycle** (une itération), chaque **fourmi** n'implémente ce processus qu'une seule fois et **M** (nombre de fourmis) solutions pour le problème de personnalisation de topologie peuvent être obtenues.

(2) désigne la probabilité que la fourmi **k** affecte le TSV **(i)** a la tuile **(j)** dans le cycle de temps **t** :

$$p_{i,j}^{k}(t)=\begin{cases}\dfrac{[\tau_{i,j}(t)]^{\alpha}\times[\eta_{i,j}]^{\beta}}{\sum_{j\in tabu}[\tau_{i,j}(t)]^{\alpha}\times[\eta_{i,j}]^{\beta}}, j\notin tabu\,k\\ 0, j\notin tabu\,k\end{cases} \quad \textbf{(2)}$$

Tabu k est utilisé pour enregistrer les **Tuiles** qui portent de TSV pour la fourmi **K** et appelé la liste **Tabou**. De plus, **τi,j(t)** signifie l'intensité de la phéromone du l'affectation de TSV **(i)** à la tuile **(j)** sur le **cycle** du temps **t**.

$$nij = \frac{Nombre\ de\ TSVs}{Center(j)} \quad \textbf{(3)}$$

(3) est appelée connaissance préalable ou visibilité, ce qui signifie : les informations d'inspiration d'affectation du TSV **(i)** au tuile **(j)**. Une fonction **(4)** est définie ici :

(4) représente le degré central de la tuile **ri** dans le Noc et signifie la capacité de communication, plus la valeur est faible, plus la capacité de communication est forte.

$$center(i) = \sum_{j=1}^{N} h_{i,j} \qquad \textbf{(4)}$$

$h_{i,j}$ represente la distance de Manhattan entre les tuiles **ri** et **rj,** et **N** represente le nombre maximum de tuiles.

6.1.2.1.2 Mapping des IPs

La procédure de résolution est la suivante : Un **IP** est sélectionné d'un ensemble facultatif en fonction de la probabilité **(5)** est attribuée à la tuile **r1**, et le **IP** est ajouté a une liste **Tabou**. Ensuite un autre **IP** non attribué est sélectionné en fonction de la probabilité précédente est attribuée à la tuile **r2**, et aussi le **IP** est ajouté à la liste **Tabou**. L'étape a été implémentée **N** fois (nombre maximum des IPs) jusqu'à ce que tous les **IPs** aient été affectés aux tuiles et que la liste **Tabou** soit pleine. Dans un **cycle** (une itération), chaque **fourmi** n'implémente ce processus qu'une seule fois et **M** (nombre de fourmis) solutions pour le problème de mapping peuvent être obtenues.

(5) désigne la probabilité que la fourmi **k** assigne le IP **(vj)** a la tuile **(ri)** dans le cycle de temps **t** :

$$p_{i,j}^{k}(t) = \begin{cases} \dfrac{[\tau_{i,j}(t)]^{\alpha} \times [\eta_{i,j}]^{\beta}}{\sum_{j \in tabu} [\tau_{i,j}(t)]^{\alpha} \times [\eta_{i,j}]^{\beta}}, j \notin tabu\ k \\ 0, j \notin tabu\ k \end{cases} \qquad \textbf{(5)}$$

Tabu k (1, 2, …, N) est utilisé pour enregistrer le **IP** attribué par la fourmi **K** et appelé la liste **Tabou**. De plus, **τi,j(t)** signifie l'intensité de la phéromone du l'assignement de l'IP **vj** à la tuile **ri** sur le **cycle** du temps **t**.

$$\eta_{i,j} = vip(j)center(i) \quad \textbf{(6)}$$

(6) est appelée connaissance préalable ou visibilité, ce qui signifie : les informations d'inspiration d'assignement du cœur IP **vj** au tuile **ri**. et il détermine la distribution de probabilité avec la force de phéromone. Les informations d'inspiration ont une grande influence sur la vitesse de convergence de l'algorithme, qui est considérée délibérément. Deux fonctions **(7)** **(8)** sont définies ici :

(7) représente le degré central de la tuile **ri** dans le Noc et signifie la capacité de communication, plus la valeur est faible, plus la capacité de communication est forte.

$$center(i) = \sum_{j=1}^{N} h_{i,j} \quad \textbf{(7)}$$

hi,j represente la distance de Manhattan entre les tuiles **ri** et **rj,** et **N** represente le nombre maximum de tuiles.

(8) représente le degré d'importance de l'IP **vi** dans le graphe d'application (APG), plus la valeur est grande, plus elle est importante.

$$vip(i) = \sum_{j=1}^{N} w_{i,j} + \sum_{j=1}^{n} w_{j,j} \quad \textbf{(8)}$$

Wi,j est le volume de communiation entre les IPs **Vi** et **Vj,** et **N** represente le nombre maximum de IPs .

Il est évident qu'un mapping optimal attribu toujours le **IP** le plus important a la tuile avec la capacité de communication la plus forte. Par conséquent, la définition des informations d'inspiration est comme ceci **(6).**

6.1.2.2 Mise à jour de phéromone

Avec l'avancement du processus d'évolution, la phéromone disparaît progressivement. Le paramètre ρ est utilisé pour représenter le degré de persistance de la phéromone **($0 < \rho < 1$)**. Lorsque toutes les fourmis ont terminé un **cycle**, la phéromone doit être mise à jour, et le contenu des informations de chaque chemin d'affectation (affectation des IPs aux tuiles et affectation des TSVs aux tuiles candidates) doit être mis à jour à travers les deux équations **(9) (10).**

6.1.2.2.1 Placement des TSVs

$$\tau_{i,j}(t+1) = \rho \times \tau_{i,j}(t) + \Delta\tau_{i,j} \qquad \textbf{(9)}$$

Dans l'équation **(9),** $\Delta\tau_{i,j}$ signifie le gain d'information et est représenté par **(11).**

$$\Delta\tau_{i,j} = \sum_{k=1}^{M} \Delta\tau_{i,j}^{k} \qquad \textbf{(11)}$$

$\Delta\tau_{i,jk}$ signifie le contenu d'information de la fourmi **k** dans le chemin d'affectation de TSV **(i)** au tuile **(j) (i ---> j)** dans ce **cycle**, et l'équation de calcul est : **(12)**

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{1}{\cos(k)}, map(k) includs(v_j \rightarrow r_j) \\ 0, else \end{cases} \quad \textbf{(12)}$$

Dans l'équation **(12)**, **cos(k)** est le coût de surface (nombe de TSVs) de la fourmi **k** complétant le programme de distribution de TSVs. Il est clair que la solution optimale a le coût de surface minimum, et donc qu'elle a la plus grande contribution à la mise à jour d'information. Afin d'éviter la stagnation de la recherche, l'intensité des phéromones est bornée avec deux limites **T_{min}** et **T_{max}** et l'équation de calcul est l'équation **(13)** et l'équation **(14)**.

$$T_{min} = T_{max} / 5 \quad \textbf{(13)} \qquad T_{max} = \frac{1}{1-\rho} \times \frac{1}{BestSolution} \quad \textbf{(14)}$$

Dans l'équation **(14) Best-Solution** est le coût de surface de la solution optimale pour le moment présent.

6.1.2.2.2 Mapping des IPs

$$\tau_{i,j}(t+1) = \rho \times \tau_{i,j}(t) + \Delta\tau_{i,j} \quad \textbf{(10)}$$

Dans l'équation **(10)**, **$\Delta\tau_{i,j}$** signifie le gain d'information et est représenté par **(15).**

$$\Delta\tau_{i,j} = \sum_{k=1}^{M} \Delta\tau_{i,j}^{k} \quad \textbf{(15)}$$

$\Delta\tau_{i,jk}$ signifie le contenu d'information de la fourmi **k** dans le chemin d'affectation de l'IP **vj** au tuile **ri (vj ---> ri)** dans ce **cycle**, et l'équation de calcul est : **(16)**

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{1}{\cos(k)}, map(k) includs(v_j \rightarrow r_j) \\ 0, else \end{cases} \quad \textbf{(16)}$$

Dans l'équation **(16)**, **cos(k)** est le coût de la fourmi **k** complétant le programme de distribution (une solution). Et qui peut être calculé par l'équation **(1)**. Il est clair que la solution optimale a le coût minimum, et donc qu'elle a la plus grande contribution à la mise à jour d'information. Afin d'éviter la stagnation de la recherche, l'intensité des phéromones est bornée avec deux limites T_{min} et T_{max} et l'équation de calcul est l'équation **(17)** et l'équation **(18)**.

$$T_{min} = T_{max} / 5 \quad \textbf{(17)}$$

$$T_{max} = \frac{1}{1-\rho} \times \frac{1}{BestSolution} \quad \textbf{(18)}$$

Dans l'équation **(18) Best-Solution** est le coût de la solution optimale pour le moment présent.

6.1.3 Algorithme MOACO proposée

On propose deux versions de **MOACO :**

- La première version qui s'appelle **MOACO-1** est utilisé pour optimiser le coût de communication pour les différents nombres de TSVs, c'est à dire

pour un nombre de TSV maximal égale à 4 : la fourmi-1 fait le mapping des IPs sur une topologie personnalisée avec 1 TSV, et la fourmi-2 fait le mapping des IPs sur une topologie personnalisée avec 2 TSV, … etc.

- La deuxième version qui s'appelle **MOACO-2** est utilisé pour optimiser le coût de communication et le coût de surface (Nombre de TSVs) comme montré dans **(6.1.2.1 Construction d'une solution)**.

Les calculs de formules **(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15) (16) (17) (18)** sont ignorés dans le pseudo code.

```
Début
    Initialisation () ;
    Pour chaque itération cycle
        Pour chaque fourmi k
            Solution-procédure () ;
            Evaluation () ; // Calculer les deux fonctions objectives
            Pareto-itération () ; //Remplir la liste de Pareto-itération par les
                                  //solutions non dominé de cette itération
        Fin pour
        Pareto-globale () ; // Remplir la liste de Pareto-globale par les
                            //solutions non dominé à partir de la liste Pareto-itération
        Mise-à-jour-phéromone () ;
    Fin pour
    Retourner L'ensemble-Pareto-optimale ;
Fin
```

6.1.3.1 MOACO-1

- Initialisation () ;

```
Début
    Initialiser_TAU_ij_t (tau_0) ; // initialiser la matrice de phéromone
    Initialiser_ Pij_k_t_vecteur ; // initialiser le vecteur de probabilités
```

Initialiser_tabu_k_liste ; // initialiser la liste Tabou

Initialiser_mapping_vecteur ; // initialiser le vecteur de mapping

Initialiser_pos_TSV_vecteur ; // initialiser le vecteur de TSVs

Initialiser_pareto_solutions_Liste ; // initialiser la liste de Pareto

Initialiser_tuile_vide_liste ; // initialiser la liste des tuiles vides

Fin

- Solution-procédure () ;

Début

Vider_mapping_vecteur ; //vider le vecteur de mapping

Vider_ pos_TSV_vecteur ; //vider le vecteur de TSVs

Vider_tabu_k_liste ; //vider la liste Tabou

Marquer_fourmi_tuile ; //associer chaque fourmi dans une tuile aléatoirement

Nombre_TSV_fourmi ; //associer tous les nombres de TSVs possible à chaque
//fourmi

Random_pos_TSVs (Nombre_TSV_fourmi, pos_TSV_vecteur) ; //remplir le vecteur
//de positions de TSVs aléatoirement à partir de nombre de TSV

Tant que tabu_k_liste n'est pas pleine

Remplir_ Pij_k_t_vecteur (Pij_k_t_vecteur) ; // remplir le vecteur des
//probabilités

Mapping (mapping_vecteur, Pij_k_t_vecteur) ; // mapper un IP à partir de
//la probabilité

Ajouter_IP (tabu_k_liste) ; // ajouter l'IP mappé a la liste Tabou

Random_tuiles (i, tuile_vide_liste) ; // marquer la tuile non vide et donner
//une autre tuile aléatoirement

Fin Tant que

Fin

- Mise-à-jour-phéromone () ;

Début

Pour chaque tuile ri

Pour chaque IP vj

//Modifier la matrice de phéromone par Tau(t+1), tel que

// T_min < Tau(t+1) < T_max

TAU_ij_t[ri][vj]=phéromone_bornes (T_min, T_max, TAU_ij_t_plus_un ()) ;

Fin pour

Fin pour

Fin

6.1.3.2 MOACO-2

- Initialisation () ;

Début

Initialiser_TAU_ij_t (tau_0) ; // initialiser la matrice de phéromone

Initialiser_TAU_ij_t_TSV (tau_0) ; // initialiser la matrice de phéromone de TSVs

Initialiser_ Pij_k_t_vecteur ; // initialiser le vecteur de probabilités

Initialiser_ Pij_k_t_vecteur_TSV ; // initialiser le vecteur de probabilités de TSVs

Initialiser_tabu_k_liste ; // initialiser la liste Tabou

Initialiser_tabu_k_liste_TSV ; // initialiser la liste Tabou de TSVs

Initialiser_mapping_vecteur ; // initialiser le vecteur de mapping

Initialiser_pos_TSV_vecteur ; // initialiser le vecteur de TSVs

Initialiser_pareto_solutions_Liste ; // initialiser la liste de Pareto

Initialiser_tuile_vide_liste ; // initialiser la liste des tuiles vides

Fin

- Solution-procédure () ;

Début

Vider_mapping_vecteur ; //vider le vecteur de mapping

Vider_ pos_TSV_vecteur ; //vider le vecteur de TSVs

Vider_tabu_k_liste ; //vider la liste Tabou

Vider_tabu_k_liste_TSV ; //vider la liste Tabou de TSVs

Marquer_fourmi_tuile ; //associer chaque fourmi dans une tuile aléatoirement

Random_nombre_TSVs (Nombre_TSV_max) ; //retourner un nombre aléatoire de //TSVs pour chaque fourmi

//Pour l'affectation des TSVs

Tant que tabu_k_liste_TSV n'est pas pleine

Remplir_ Pij_k_t_vecteur_TSV (Pij_k_t_vecteur_TSV) ; // remplir le vecteur //des probabilités de TSVs

Affecter_TSV (pos_TSV_vecteur, Pij_k_t_vecteur_TSV) ; // affecter un TSV //à partir de la probabilité de TSVs

Ajouter_TSV (tabu_k_liste_TSV) ; // ajouter le TSV affecté à la liste Tabou de //TSVs

Fin Tant que

//Pour le mapping des IPs

Tant que tabu_k_liste n'est pas pleine

Remplir_ Pij_k_t_vecteur (Pij_k_t_vecteur) ; // remplir le vecteur des //probabilités

Mapping (mapping_vecteur, Pij_k_t_vecteur) ; // mapper un IP à partir de //la probabilité

Ajouter_IP (tabu_k_liste) ; // ajouter l'IP mappé a la liste Tabou

Random_tuiles (i, tuile_vide_liste) ; // marquer la tuile non vide et donner //une autre tuile aléatoirement

Fin Tant que

Fin

- Mise-à-jour-phéromone () ;

Début

//Pour le mapping des IPs

```
Pour chaque tuile ri
    Pour chaque IP vj
        //Modifier la matrice de phéromone par Tau(t+1), tel que
        // T_min < Tau(t+1) < T_max
        TAU_ij_t[ri][vj]=phéromone_bornes (T_min, T_max, TAU_ij_t_plus_un ()) ;
    Fin pour
Fin pour

//Pour l'affectation des TSVs
Pour chaque TSV tsv
    Pour chaque tuile ri
        //Modifier la matrice de phéromone de TSVs par Tau_TSV(t+1), tel
        //que   T_min_TSV < Tau_TSV(t+1) < T_max_TSV
        TAU_ij_t_TSV[ri][tsv]=phéromone_bornes(T_min_TSV,T_max_TSV,TAU_ij_
        t_plus_un_TSV ()) ;
    Fin pour
Fin pour
Fin
```

6.1.4 Comparaison entre MOACO proposé et ACO

Le tableau suivant **(Tableau 4.1)** présente quelques différents points entre les deux algorithmes (**MOACO** proposé et **ACO [95]**).

Tableau 4.1 : Comparaison entre MOACO et ACO

ACO [95]	MOACO proposée
Mono-objectif	Multi-objectifs
Un ensemble de solutions similaires (avec le même coût).	Ensemble Pareto optimale (solution de compromis).

La capacité de communication de chaque tuile est statique dans tous les cycles.	La capacité de communication de chaque tuile est différente a cause de nombre et de positionnement de TSVs.
les informations d'inspiration d'assignement d'un IP sur une tuile sont statique dans tous les cycles.	les informations d'inspiration d'assignement d'un IP sur une tuile sont différents a cause de nombre et de positionnement de TSVs.
Chaque fourmi propose une solution de mapping avec le même nombre des TSVs	Les fourmis proposent des solutions de mapping avec un nombre des TSVs différent.

Il est évident que l'algorithme classique **ACO** doit être modifié pour être adapté à la résolution des problèmes d'optimisation multi-objectifs. L'ensemble des solutions d'un problème avec multiples objectifs ne se compose pas d'une seule solution, donc il est nécessaire de trouver un ensemble de solutions de compromis qui est l'ensemble de Pareto dans notre cas. Notre **MOACO** proposé distribue les fourmis sur les tuiles de façon aléatoire afin de bien explorer l'espace de recherche. La capacité de communication **(7)** de chaque tuile dans le **MOACO** proposé dépend du nombre et positions des TSVs, ce qui permet de changer les informations d'inspiration **(6)** d'assignement d'un IP sur une tuile. Dans notre **MOACO** proposé, les fourmis dans chaque cycle traitent différents cas de problème a cause de nombre aléatoire et de positionnement des TSVs.

6.2 Implémentation

6.2.1 L'outil utilisé

Java : Le langage Java est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, présenté officiellement le 23 mai 1995.

Notre choix s'est porté sur le langage de développement Java pour les raisons suivantes **[96]** :

- **Java est portable :** en effet, Java est indépendante de la plateforme, il peut se déplacer d'un système informatique à un autre.

- **Java est facile à apprendre :** le langage Java a été conçu pour être plus facile à utiliser, à écrire, à compiler, à déboguer et à apprendre que les autres langages de programmation.
- **Java est robuste :** les compilateurs Java fait une vérification rapide du code, et sont capables de détecter des problèmes qui seraient apparus au cours de l'exécution dans les autres langages.
- **Il est orienté objet :** il permet d'écrire des programmes de manière modulaire et facilite leurs extensions.

6.2.2 NetBeans

C'est un environnement de développement fondé par SUN Microsystems en juin 2000. Un outil pour les programmeurs pour écrire, compiler, déboguer, et déployer des programmes. Il comprend les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, éditeur graphique d'interfaces).

7. Architecture de la technique proposée :

Notre application se base sur un système qui nécessite des entrées pour produire des sorties. Nous avons modélisé notre architecture avec trois principaux modules reliés entre eux par des fonctions comme suit :

- **Module 1** : les applications qui sont sous forme de graphes des IPs :

  Benchmarks aléatoires : **TG7**

  Benchmarks standard (Applications réelles) : **VOPD**, **MPEG**, **PIP**, **MWD**.

- **Module 2** *:* les architectures **Noc** qui constituent un ensemble des **tuiles** reliés entre eux par des liens.

- **Module 3 :** l'algorithme de la colonie de fourmis mono-objectif (**ACO**) et multi-objectif (**MOACO**) et son adaptation sur notre problème.
- **F1 (Lire un fichier Txt) :** une fonction permet de lire un **fichier.txt** qui décrit une application.
- **F2 (Choix d'une topologie) :** une fonction permet de choisir une topologie **Noc** parmi les topologies proposées (**2D ou 3D**).
- **F3(Afficher les résultats dans une interface graphique) :** permet d'afficher la solution finale du mapping obtenue par la colonie de fourmis dans une interface graphique qui contient :

  1) Un tableau des affectations des **IPs** dans l'architecture **Noc** et ses positions.

  2) Des cases des résultats pour le temps d'exécution + le coût de communication + le coût de surface (nombre des **TSVs)**.

La **figure 36** illustre un schéma qui détaille l'architecture globale du projet.

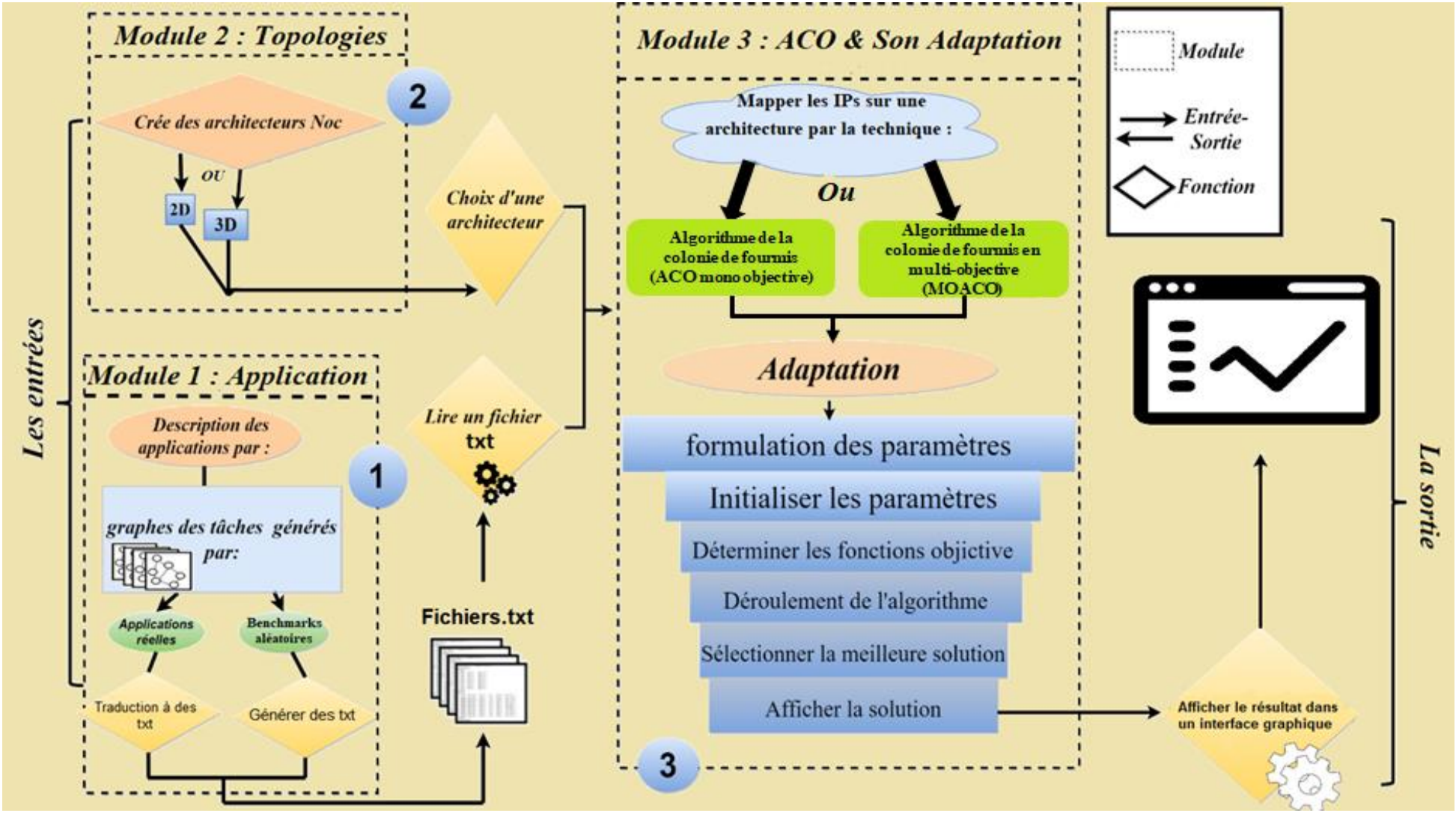

Figure 36 : L'architecture globale du projet

8. Conclusion

Dans ce chapitre, on a expliqué comment nous avons adapter l'**ACO** afin de résoudre le problème de mapping et le choix de la topologie 3D. Nous avons commencé par introduire notre problème en formule mathématique, puis on a présenté notre supposition, détaillé la conception avec L'**ACO** afin de résoudre notre problématique. Le chapitre suivant représente les différents benchmarks sur lesquelles on a effectué les tests et les résultats obtenus.

Chapitre 5

Tests et résultats

1. Introduction

Après avoir implémenté les différentes techniques étudiées dans le chapitre précédent, nous présentons les résultats obtenus lors de nos expérimentations effectuées dessous. Nous commençons par présenter les différents benchmarks sur lesquels nous avons effectué nos tests, puis nous étudions les résultats obtenus après la réalisation des tests sur ces benchmarks.

2. Environnement et hypothèse

Pour l'implémentation on a choisi le langage **JAVA** car il est multiplateforme (langage interprété). Les tests sont réalisés sur une machine équipée d'un processeur Intel Core i3-7020U 2.30 GHz et d'une RAM de 4 GO sous le système d'exploitation Windows 10 de 64 bits.

3. Présentation des benchmarks

Dans notre expérimentation on a utilisé **4** benchmarks standards qui sont **VOPD**, **MWD**, **PIP**, **MPEG** et **1** aléatoire qui est **TG7**.

3.1 Benchmark VOPD

L'application VOPD est composée de 16 composants (IPs) qui communiquent entre eux à travers 21 liens.

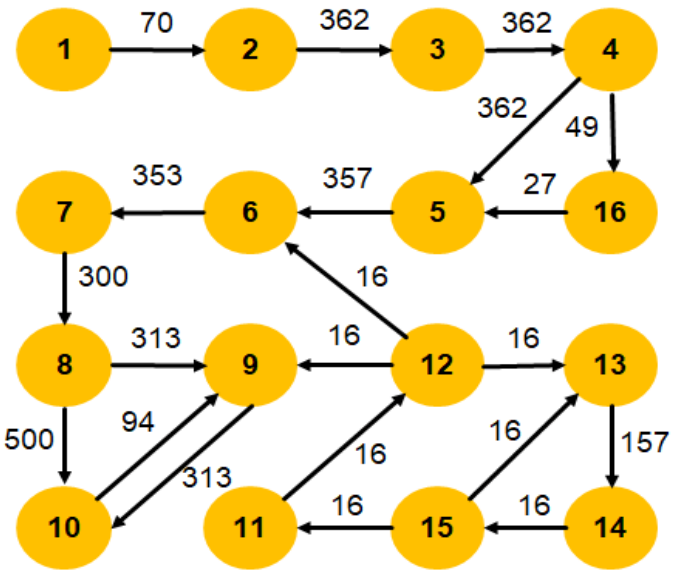

Figure 37 : Benchmark VOPD

3.2 Benchmark MPEG

Le benchmark MPEG possède 12 nœuds et 13 liens, la figure suivante montre le graphe d'application de ce benchmark.

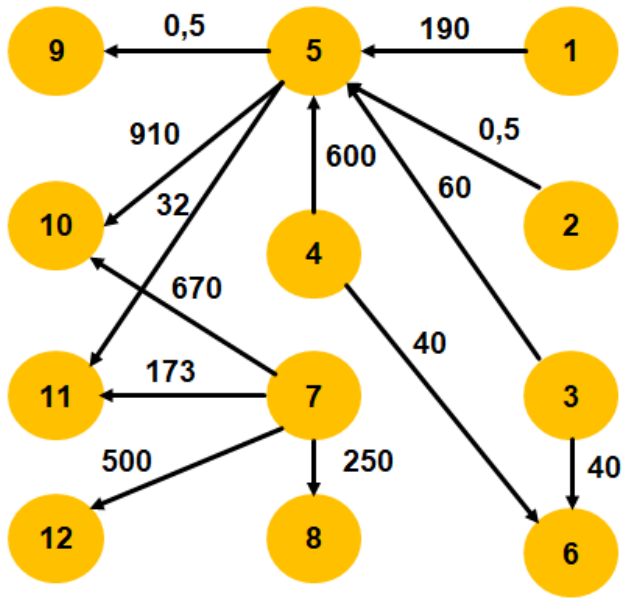

Figure 38 : Benchmark MPEG

3.3 Benchmark MWD

Le benchmark MWD possède 12 nœuds et 13 liens, la figure suivante montre le graphe d'application de ce benchmark.

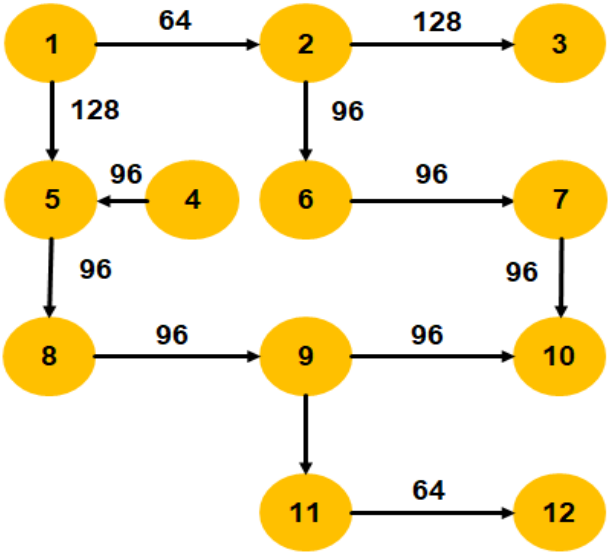

Figure 39 : Benchmark MWD

3.4 Benchmark PIP

Le benchmark PIP possède 8 nœuds et 7 liens, la figure suivante montre le graphe d'application de ce benchmark.

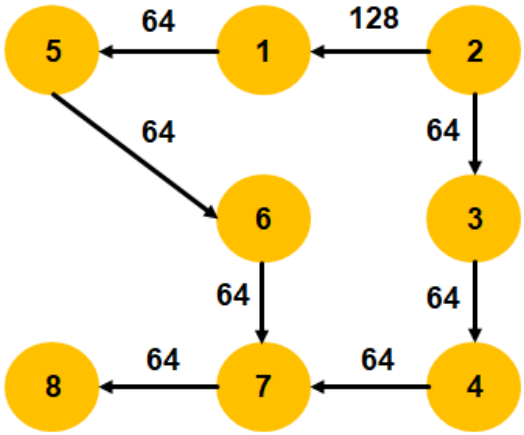

Figure 40 : Benchmark PIP

3.5 Benchmark Aléatoire

TGFF a été développé par R.P. Dick and D.L. Rhodes en 1998 pour faciliter la standardisation des benchmarks aléatoires pour la planification et la recherche d'allocation en générale, et pour la recherche coopérative hardware-software en particulier. **TGFF** est convenable pour plusieurs applications qui nécessitent la génération des graphes de pseudo-aléatoire **[100].** Après, K. Vallerio a modernisé et amélioré le code et la documentation **[101].**

TG7 est un benchmark aléatoire généré par la **TGFF**, ce benchmark possède 30 nœuds et 36 liens.

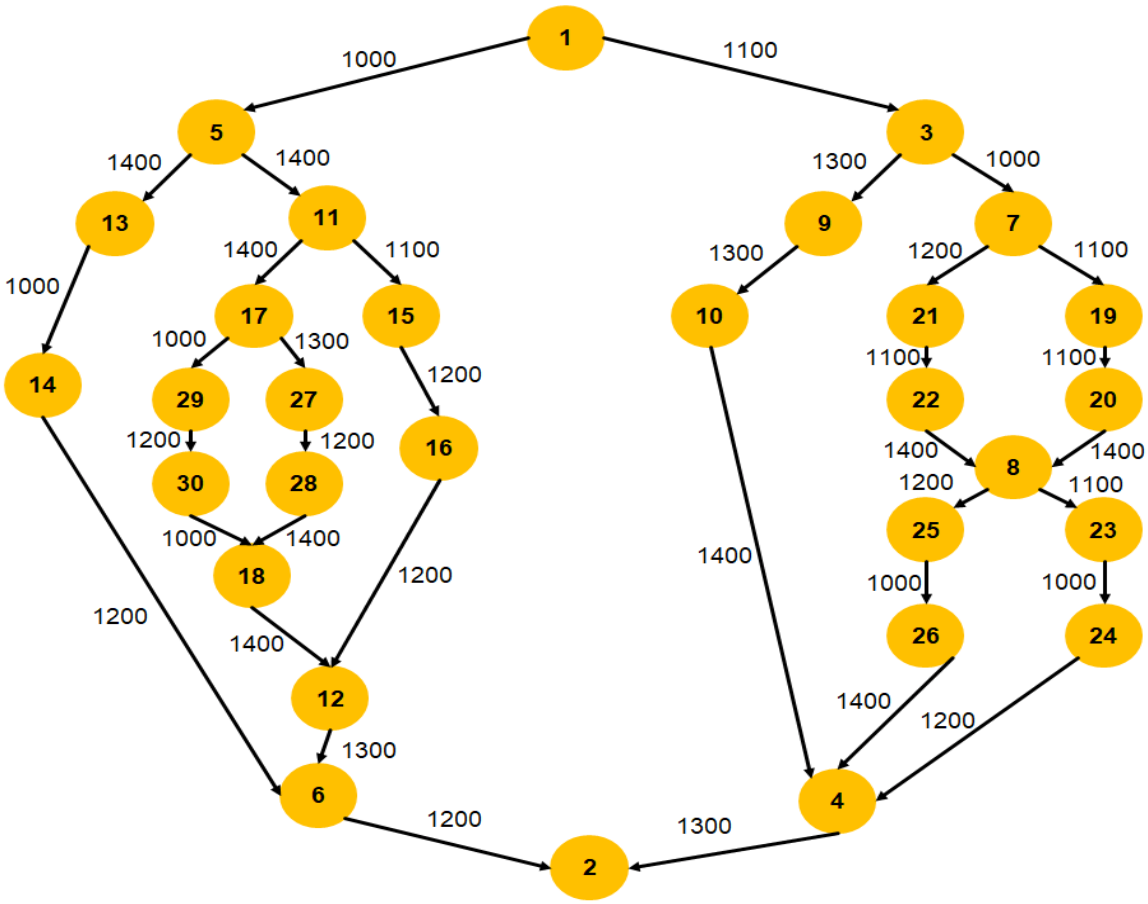

Figure 41: Benchmark TG7

3.6 Architecture des benchmarks

Les applications sont mappées sur des structures de mapping 2D et 3D avec des tailles indiquées dans le **(Tableau 5.1)**.

Tableau 5.1 : Différentes applications et tailles des topologies 2D et 3D Mesh

Benchmark	2D MESH	3D MESH
PIP	**4x2**	**2x2x2**
MWD	**4x4**	**2x3x2**
MPEG	**4x4**	**2x3x2**
VOPD	**4x4**	**2x4x2**
TG7	**6x5**	**5x3x2**

4. Etude et résultats des tests de la méthode MOACO

4.1 Mono-objectif

La recherche **(ACO) [95]** en implémenter une probabilité de distribution qui dépend d'une heuristique et de phéromone pour la distribution des IPs aux tuiles.

- On veut savoir si cette méthode donne des résultats pertinents sans et avec l'heuristique **(6)** dans le cadre de l'objectif (coût de communication). Selon la formule de probabilité **(2)**, si on affecte la valeur **0** au paramètre β**,** La probabilité ne dépend que de phéromone comme celle de l'**ACO [62]**. On va fixer les paramètres **(TAU_0 = 1**, **ALPHA = 1**, **BETA = 3.5**, **P = 0.7) [95]** avec une population égale à **30** et un nombre d'itération égal à **1000**.

- On veut savoir aussi si la méthode donne des résultats pertinents si le % de TSVs utilisé n'est pas 100% : on fixe le pourcentage des TSVs selon les pourcentages fixés dans la littérature (25%, 50%, 100%) et on calcule le coût de communication et le temps CPU.

Le **(Tableau 5.2)** résume les résultats des tests dans les différents benchmarks.

Tableau 5.2 : Résultats de L'ACO version mono-objectif

Benchmark	Topologie	α	Le coût de communication (probabilité sans heuristique)	Le coût de communication (probabilité avec heuristique)	Le temps d'exécution (s) (probabilité sans heuristique)	Le temps d'exécution (s) (probabilité avec heuristique)
PIP	2D		640	640	1.61	1.68
	3D 25%	0.8	1420.8	1420.8	1.79	1.82
		1	1536	1536	1.78	1.87
	3D 50%	0.8	972.8	972.8	1.92	1.93
		1	1024	1088	1.87	1.96
	3D 100%	0.8	563.2	563.2	1.63	1.67
		1	640	640	1.61	1.64
MWD	2D		1248	1440	4.84	4.92
	3D 25% (1-TSV)	0.8	2943.99	2943.99	5.30	5.37
		1	3168	3168	5.31	5.27
	3D 50%	0.8	2111.99	1984	5.58	5.88
		1	2240	2144	5.60	5.59
	3D 100%	0.8	1177.6	1184	4.81	4.97
		1	1216	1216	4.81	4.96
MPEG	2D		3567	3862.5	4.89	4.86
	3D 25% (1-TSV)	0.8	6499.79	6499.79	5.21	5.32
		1	7193	7193	5.22	5.28
	3D 50%	0.8	4861.79	5322.29	5.42	5.70
		1	5555	5734.5	5.36	5.43
	3D 100%	0.8	3456.5	3535.20	5.00	4.90
		1	3567	3752	4.87	4.85
VOPD	2D		4231	4479	10.95	10.93
	3D 25%	0.8	7546.8	7603.8	11.91	11.99
		1	8175	8250	11.92	12.03
	3D 50%	0.8	6356.80	6790.8	12.65	12.90
		1	6836	6971	12.29	12.42
	3D 100%	0.8	3878.40	4103	10.90	11.35
		1	4141	4378	10.93	11.07

TG7	2D		100600	105300	68.02	68.50
	3D 25% (3-TSV)	0.8	143200	150800	72.34	75.07
		1	154500	158500	73.04	75.26
	3D 50% (7-TSV)	0.8	124100	129500	78.87	79.76
		1	137100	138000	77.96	79.28
	3D 100%	0.8	83980	84340	66.78	68.52
		1	89400	91700	66.23	68.45

❖ Discussion

Le **(Tableau 5.2)** illustre que l'**ACO** qui utilise une probabilité sans heuristique **(ACO-1)** améliore dans la plupart du temps le coût : prenons par exemple les benchmarks **MPEG** et **VOPD** presque dans tous les cas où on utilise l'**ACO-1**, le coût est mieux que le coût de l'**ACO** qui utilise une probabilité avec heuristique **(ACO-2)** en un temps d'exécution réduit. Cependant, si on prend par exemple le benchmark **MWD** avec une affectation de 50% de TSV, le coût de l'**ACO-2** égal à (1984), par contre le coût de l'**ACO-1** égal à (2111.99).

On constate aussi que, quand on passe d'une topologie 3D 100% connectée verticalement vers une topologie 3D 25% connectée verticalement, le coût de communication augmente d'environ 114% en moyenne, par exemple si on compare nos résultats avec ceux de **[102]** pour une topologie 3D 25% connectée verticalement, on trouve que notre **ACO** ne donne pas de résultats intéressants (4119) de **[102]** contre (8175). Et si on passe d'une topologie 3D vers une topologie 2D le coût de communication augmente avec l'augmentation du nombre d'IPs : 2.17% pour le benchmark **VOPD** (16 IPs) et 12% pour le benchmark aléatoire **TG7** (30IPs).

On a comparé nos résultats avec différentes techniques, pour les benchmarks **PIP** et **MPEG**, On a remarqué que notre **ACO-1** donne de meilleurs résultats : (640) pour **PIP** qui est l'optimum, et (3567) contre (3633) de **ACO [62]** pour **MPEG** (selon le Survey **[103]**). Pour les benchmarks **MWD** et **VOPD** nos résultats sont proches de l'optimal (1248) contre 1184 de **ILP [49]** pour **MWD** et (4231) contre (4119) de **ILP [49]** pour **VOPD**.

Le **(Tableau 5.3)** illustre une comparaison entre les résultats de coût de communication dans les différents benchmarks de quelques techniques de littérature et notre **ACO** avec et sans heuristique de probabilité.

Tableau 5.3 : Comparaison avec littérature

Technique de mapping	Coût de communication	Benchmark
ACO (probabilité sans heuristique)	640	PIP
ACO (probabilité avec heuristique)	640	
NMAP [64]	640	
LMAP [69]	640	
ACO (probabilité sans heuristique)	1248	MWD
ACO (probabilité avec heuristique)	1440	
NMAP [64]	1184	
LMAP [69]	1248	
ILP [49]	1184	
ACO (probabilité sans heuristique)	3567	MPEG
ACO (probabilité avec heuristique)	3862.5	
Onyx [67]	3567	
XY_ADB [70]	4368	
CastNet [76]	3852	
MapGtoM [78]	3567	
ILP [49]	3567	
ACO [62]	3633	
PSO [56]	3567	
ACO (probabilité sans heuristique)	4231	VOPD
ACO (probabilité avec heuristique)	4479	
NMAP [64]	4265	
Onyx [67]	4577	
Crinkle [68]	4157	
LMAP [69]	4189	
XY_ADB [70]	4568	
RMAP [75]	6323	
CastNet [76]	4135	
MapGtoM [78]	4135	
ILP [49]	4119	

SBMAP [52]	4125	
PSO [56]	4119	

4.2 Multi-objectif

4.2.1 Effet de variation de la taille de population

Le but de cette étude est de tester l'effet de variation de la taille de population sur le coût. Dans ce qui suit on va fixer les paramètres **(TAU_0 = 1**, **ALPHA = 1**, **BETA = 3.5**, **P = 0.7) [95]**.

- $Nombre\ de\ TSV\ moyen = \frac{\sum TSV\ des\ solutions\ des\ Archives}{\sum Taille\ des\ Archives}$
- $Coût\ moyen = \frac{\sum Coût\ des\ solutions\ des\ Archives}{\sum Taille\ des\ Archives}$
- $Nombre\ de\ solution\ optimal\ moyen\ = \frac{\sum Taille\ des\ Archives}{Nombre\ d\ 'executions}$

Le **(Tableau 5.4)** représente les résultats obtenus après **5** exécutions en variant la taille de la population avec un nombre d'itération égal à **100**.

***Tableau 5.4 :* Résultats de variation de la taille de population**

Benchmark	Taille de la population	MOACO-1			MOACO-2		
		Nombre de solution optimal moyen	Coût de communication moyen (α = 1)	Nombre de TSV moyen	Nombre de solution optimal moyen	Coût de communication moyen (α = 1)	Nombre de TSV moyen
PIP	50	4	944	2	3	1058.13	2
	100	4	931.84	2	3	1049.6	2
	250	4	934.40	2	3	1036.8	2
	500	4	918.40	2	3	1036.8	2
	1000	4	918.40	2	3	1036.8	2
MWD	50	5	2219.85	3	5	2306.66	3
	100	6	2165.54	3	5	2291.99	3
	250	6	2131.19	3	5	2265.04	3
	500	6	2101.75	3	5	2242.66	3

	1000	6	2069.75	3	5	2174.71	3
MPEG	50	5	5520.35	3	3	5772.41	2
	100	5	5415.97	3	3	5659.16	2
	250	6	5270.37	3	3	5638.36	2
	500	5	5216.158	3	4	5444.71	3
	1000	5	5122.415	3	3	5427.06	3
VOPD	50	7	7887.59	4	5	8367.26	3
	100	8	7835.38	4	6	7994.66	4
	250	7	7576.59	4	6	7792.15	4
	500	8	7392.24	4	6	7656.67	4
	1000	7	7313.17	4	6	7565.76	4

❖ Discussion

On a constaté que l'augmentation de la taille de population réduit le coût pour tous les benchmarks testés. Plus la taille de la population est grande, plus le coût de communication réduit, plus on a la chance de tomber sur des résultats optimaux, plus le nombre de solution optimal moyen augmente. Si la taille de la population est trop grande la convergence va être lente et l'inverse si la taille de la population est très petite y a des risques de tomber sur local optimum (retrouver les mêmes solutions à chaque fois).

Dans nos tests la taille de population idéale est égale à **1000**, car elle donne plus de solutions optimales et le coût de communication/nombre de TSV est le minimum avec cette valeur.

4.2.2 Effet de variation du nombre d'itération

Le **(Tableau 5.5)** représente les résultats obtenus après **5** exécutions en variant le nombre d'itération avec une taille de population égal à **1000**.

Tableau 5.5 : **Résultats de variation du nombre d'itération**

Benchmark	Nombre d'itération	MOACO-1			MOACO-2		
		Nombre de solution optimal moyen	Coût de communication moyen (α = 1)	Nombre de TSV moyen	Nombre de solution optimal moyen	Coût de communication moyen (α = 1)	Nombre de TSV moyen

PIP	100	4	918.40	2	3	1036.8	2
	200	4	918.40	2	3	1036.8	2
	500	4	918.40	2	3	1036.8	2
	800	4	918.40	2	3	1036.8	2
MWD	100	6	2069.75	3	5	2174.71	3
	200	6	2035.19	3	5	2167.03	3
	500	6	2022.84	3	5	2143.99	3
	800	6	1999.88	3	5	2127.35	3
MPEG	100	5	5122.41	3	3	5427.06	3
	200	5	5060.50	3	4	5378.74	3
	500	5	5030.65	3	4	5346.27	3
	800	5	4987.13	3	4	5365.38	3
VOPD	100	7	7313.17	4	6	7565.76	4
	200	8	7190.48	4	6	8111.51	4
	500	8	7007.66	4	7	7858.24	4
	800	7	7572.45	4	6	7820.28	4

❖ Discussion

On remarque que, pour le benchmark **PIP** 100 itérations est suffisante pour avoir des bons résultats en termes de coût de communication et de nombres de TSVs. Cependant pour les benchmarks **MWD**, **MPEG** et **VOPD** l'augmentation du nombre d'itérations influence positivement sur le coût et le nombre de TSV. Si on compare le nombre de solutions optimales obtenues, on voit qu'il augmente avec l'augmentation de nombre d'itérations pour les benchmarks **MPEG** et **VOPD**. Dans notre série de test le nombre d'itérations idéale est **500** par rapport au nombre de solutions optimales trouvées. Plus le nombre de solutions est grand, plus un concepteur de circuit a le choix, c'est-à-dire choisir une solution qui optimise l'un des objectifs ou une solution qui donne un compromis entre les deux.

5. Etude comparative

Le **(Tableau 5.6)** montre les meilleurs paramètres trouvés afin d'obtenir des résultats intéressants en termes de coût de communication et de nombre de TSV. Les benchmarks utilisés sont **PIP**, **MWD**, **MPEG**, **VOPD**, **TG7**.

Tableau 5.6 : Récapitulatif des résultats pour le choix des paramètres

	La taille de la population	Nombre d'itération	α	β	ρ	$\tau(0)$
PIP	**250 et 500**	**100**	**1**	**3.5**	**0.7**	**1**
MWD	**1000**	**800**	**1**	**3.5**	**0.7**	**1**
MPEG	**250 et 1000**	**500 et 800**	**1**	**3.5**	**0.7**	**1**
VOPD	**1000**	**100 et 500**	**1**	**3.5**	**0.7**	**1**
TG7	**1000**	**800**	**1**	**3.5**	**0.7**	**1**

Le tableau **(Tableau 5.7)** représente une comparaison entre les deux méthodes **MOACO-1** et **MOACO-2** en fonction du coût de communication et de nombre de TSVs.

Tableau 5.7 : Comparaison entre les archives des deux méthodes

Benchmark	MOACO-1			MOACO-2		
	Nombre de solution optimale	Coût de communication	Nombre de TSVs	Nombre de solution optimale	Coût de communication	Nombre de TSVs
PIP	**4**	**1536**	**1**	**3**	**1536**	**1**
		1024	**2**		**1024**	**2**
		896	**3**		**896**	**3**
		640	**4**			
MWD	**6**	**3168**	**1**	**4**	**3168**	**1**
		2528	**2**		**2496**	**2**
		2208	**3**		**2208**	**3**
		2080	**4**		**2016**	**4**
		1952	**5**			
		1280	**6**			
MPEG	**6**	**7193.0**	**1**	**4**	**7193**	**1**
		5955.5	**2**		**5714.5**	**2**
		5795.5	**3**		**5635.5**	**3**
		5734.5	**4**		**5555**	**4**
		5675	**5**			

		3772.5	6			
VOPD	8	11110	1	6	10817	1
		8744	2		8671	2
		8169	3		7901	3
		7465	4		7352	4
		7398	5		6937	5
		7040	6			
		6824	7		6669	7
		4852	8			
TG7	13	192600	1	12	190100	1
		170000	2		168900	2
		159500	3		154000	3
		153300	4		153600	4
		151300	5		149500	5
		145600	6		147200	6
		140000	7		138800	7
			8		137400	8
		132000	9		134400	9
			10		128500	10
		130800	11		124400	11
		127900	12			12
		125100	13		115900	13
		117500	14			14
		97600	15			15

Le résultat des deux méthodes est représenté sous forme de graphe ayant comme axes les deux objectifs à optimiser (coût de communication et coût de surface (nombre de TSVs)).

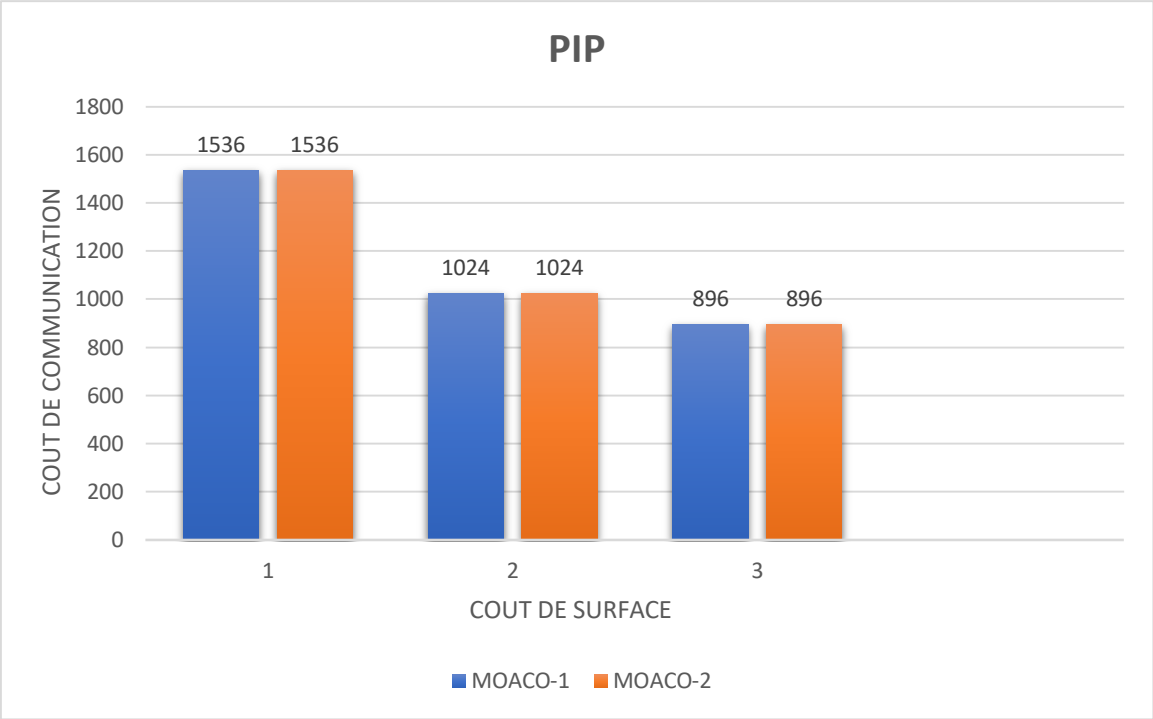

Figure 42 : Comparaison entre MOACO-1 et MOACO-2 avec PIP

La **(Figure 42)** montre le front de Pareto du benchmark **PIP**, on peut remarquer que les résultats des 2 méthodes sont les mêmes.

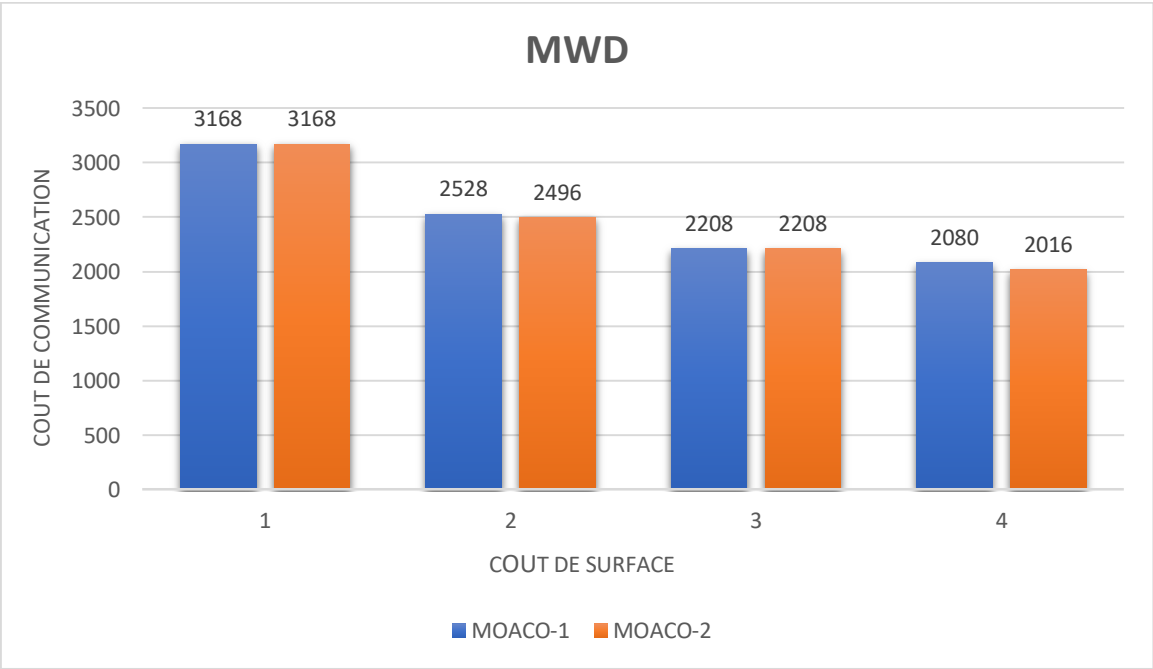

Figure 43 : Comparaison entre MOACO-1 et MOACO-2 avec MWD

La **(Figure 43)** illustre que pour le benchmark **MWD** avec la méthode **MOACO-2** lors de l'utilisation de 2 et 4 TSVs on obtient un coût minimal égal à 2496 avec 2 TSVs et 2016 avec 4 TSVs, en revanche avec **MOACO-1** on obtient un coût égal à 2528 avec 2 TSVs (1.28% de différence) et 2080 avec 4 TSVs (3.17% de différence). La différence est presque négligeable en prenant en considération le nombre de solutions de la méthode **MOACO-1**.

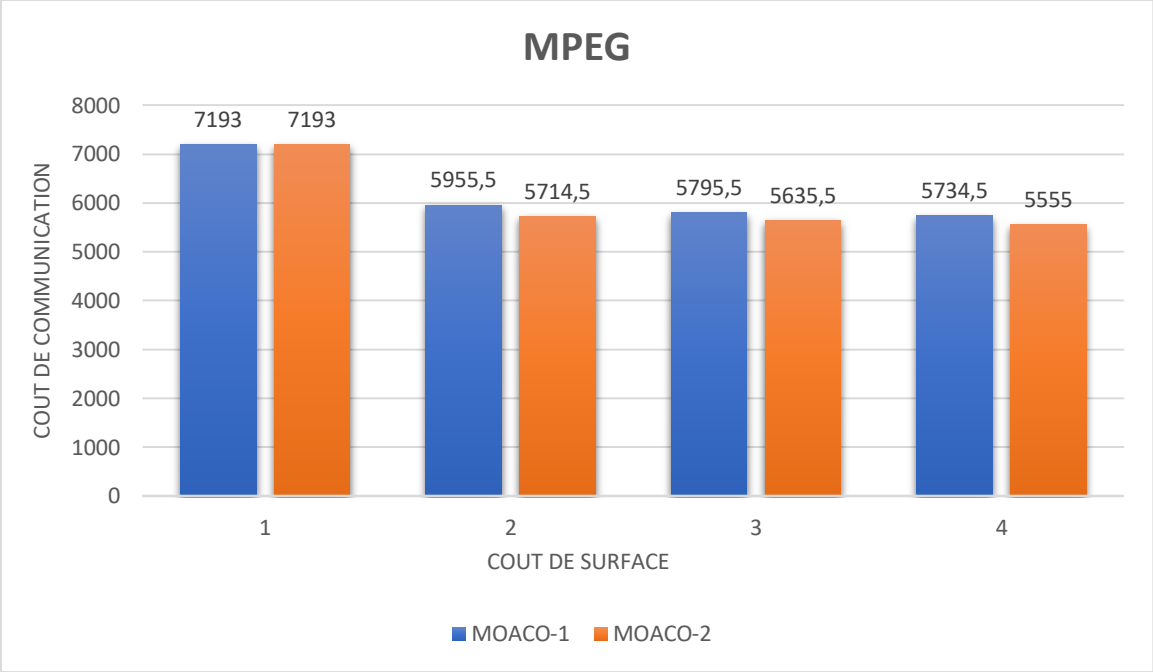

Figure 44 : Comparaison entre MOACO-1 et MOACO-2 avec MPEG

D'après la **(Figure 44)** on remarque que la méthode **MOACO-2** donne des résultats un peu mieux que la méthode **MOACO-1** : (4.21% de différence) pour 2 TSVs, (2.83% de différence) pour 3 TSVs, (3.23% de différence) pour 4 TSVs. Cependant la méthode **MOACO-1** produit 6 solutions différentes et la méthode **MOACO-2** produit 4 solutions.

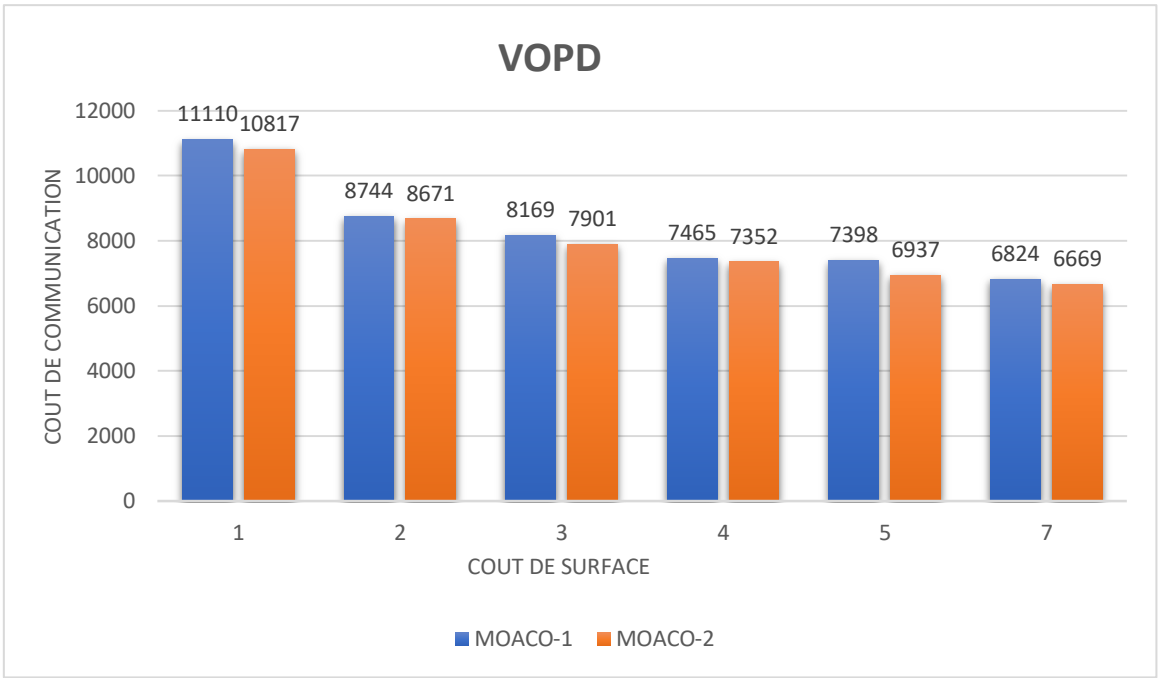

Figure 45 : Comparaison entre MOACO-1 et MOACO-2 avec VOPD

Pour le graphe du benchmark **VOPD** on remarque que la pire augmentation du coût de communication est de 8%, et la moyenne de l'augmentations dans toutes les solutions est de 3.26%. Donc les deux méthodes donnent des résultats proches avec un nombre de solutions plus pour la méthode **MOACO-1**.

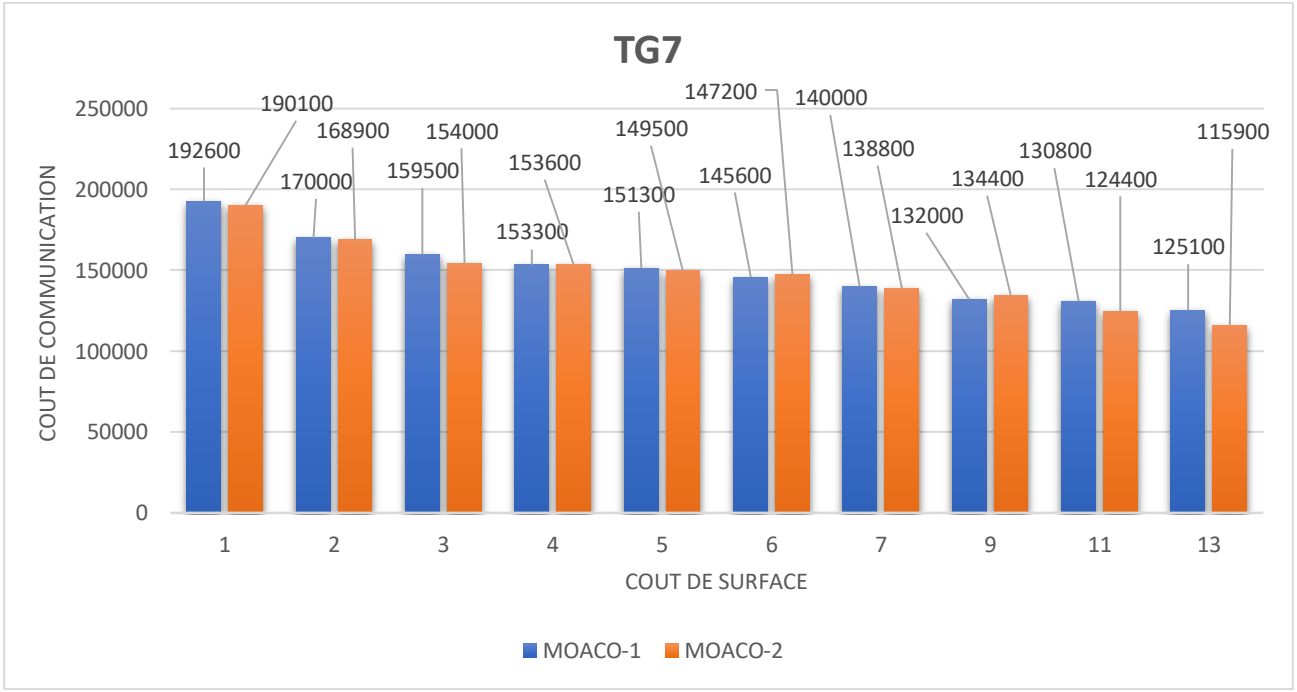

Figure 46 : Comparaison entre MOACO-1 et MOACO-2 avec TG7

Pour le dernier graphe du benchmark **TG7** on peut remarquer qu'il existe des solutions où le **MOACO-2** est meilleure que **MOACO-1** (la moyenne de l'augmentations dans toutes ces solutions est de 2.95%), comme il existe aussi des solutions où le **MOACO-1** est meilleure que **MOACO-2** (la moyenne de l'augmentations dans toutes ces solutions est de 1.03%) avec un nombre de solutions plus.

❖ Discussion

D'après les graphes précédents et le **(Tableau 5.7)** on peut remarquer que les deux méthodes donnent des résultats intéressants. Pour quelques Benchmarks on peut observer une différence où la méthode **MOACO-2** est meilleure au terme de coût de communication, cependant la méthode **MOACO-1** donne des solutions plus ce qui permet d'offrir le choix au concepteur de circuit.

Pour la comparaison avec la littérature, il n'existe pas une méthode qui optimise les deux objectifs à la fois (coût de communication et coût de surface (nombre de TSVs)), donc on les a testés pour une future comparaison.

La méthode **ACO-2** utilise une probabilité avec une heuristique **(6)** pour mapper les IPs aux tuiles, et la méthode **MOACO-2** utilise deux probabilités avec deux heuristiques différentes **(3)** et **(6)** : la première pour l'affectation des TSVs aux tuiles candidates et la deuxième pour le mapping des IPs aux tuiles. Le **(Tableau 5.8)** et la **(Figure 47)** montrent une comparaison entre les deux méthodes (**ACO-2** et **MOACO-2**).

Tableau 5.8 : Comparaison entre ACO-2 et MOACO-2

Benchmark	Le coût de communication de ACO-2	Le coût de communication de MOACO-2	Nombre de TSVs
PIP	1536	1536	1TSV ⇔ 25%
	1088	1024	2TSV ⇔ 50%
MWD	3168	3168	1TSV ⇔ 25%
	2144	2208	3TSV ⇔ 50%
MPEG	7193	7193	1TSV ⇔ 25%

	5734.5	5635.5	3TSV ⇔ 50%
VOPD	8250	8671	2TSV ⇔ 25%
	6971	7352	4TSV ⇔ 50%
TG7	158500	154000	3TSV ⇔ 25%
	138000	138800	7TSV ⇔ 50%

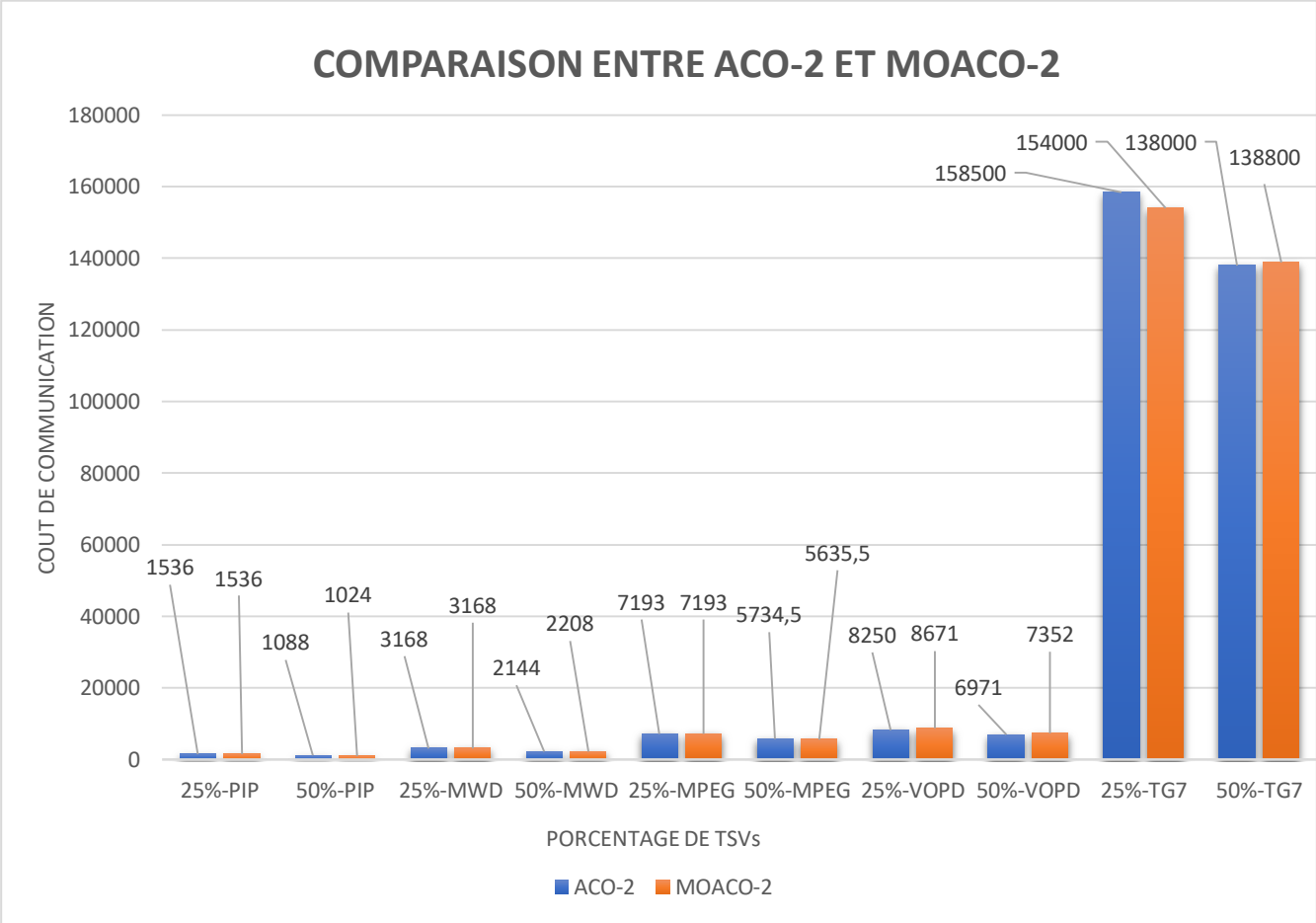

Figure 47 : Comparaison entre ACO-2 et MOACO-2

On peut remarquer que les deux méthodes donnent presque les mêmes résultats : Il existe des solutions où le **ACO-2** est meilleure que **MOACO-2** (la moyenne de l'augmentations dans toutes ces solutions est de 3.52%), comme il existe aussi des solutions où le **MOACO-2** est meilleure que **ACO-2** (la moyenne de l'augmentations dans toutes ces solutions est de 3.64%). Cependant le **MOACO-2** produit des solutions avec des pourcentages (configuration TSV) hors ceux déjà mentionner dans le **(Tableau 5.8)** (25% et 50%), des solutions avec les pourcentages suivantes : (33.33%, 66.66%) pour les benchmarks **MWD** et **MPEG**,

(75%) pour le benchmark **PIP**, (12.5%, 37.5%, 62.5%, 87.5%) pour le benchmark **VOPD**, (6.66%, 13.33%, 26.66%, 33.33%, 40%, 53.33%, 60%, 66.66%, 73.33%, 86.66%) pour le benchmark **TG7**.

6. Conclusion

Dans ce chapitre, nous avons commencé par présenter les benchmarks utilisés, puis on a effectué une série de tests de validation de notre proposition et une étude de paramétrage des deux méthodes proposées, on a terminé avec une étude comparative entre ces deux méthodes. Nous avons constaté que notre solution fournit des solutions de compromis intéressantes entre les deux objectifs (coût de communication et coût de surface) en un temps raisonnable.

CONCLUSION GÉNÉRALE

Le concept de réseaux sur puce est apparu pour résoudre les problèmes liés aux limites du bus partagé, son principe est d'introduire un Réseau de routeurs dans une puce afin d'interconnecter les différents composants intégrés.

Les concepteurs intègrent de plus en plus d'unité de traitement dans les systèmes sur puce, la quantité de donnée échangée dans les systèmes a augmenté, ainsi la conception de ces systèmes devient complexe. Dans ce contexte, le problème de mapping est apparu comme un problème NP-difficile, qui tente à satisfaire plusieurs objectifs.

Le but principal de notre travail présenté dans ce document est d'implémenter une technique qui traite le problème de mapping en réduisant le coût de communication et le coût de surface.

D'après les recherches effectuées, nous avons remarqué que la méthode d'optimisation en multi-objectif par colonies des fourmis (**MOACO**) n'avait jamais été utilisée pour résoudre le problème de mapping dans une architecture Noc 3D, basée sur la minimisation de nombre des **TSV** (la surface) et le coût de communication.

Dans nos perspectives, nous voulons d'abord approfondir notre expertise dans ce domaine et valider notre contribution par la communauté scientifique. Ensuite, améliorer notre contribution en :

- Procédant au reste des phases de conception des réseaux sur puce **3D**, tel que le routage et l'ordonnancement.
- Améliorant notre solution algorithmique pour optimiser autre objectif (consommation d'énergie) avec les deux objectifs (coût de communication et coût de surface).
- Explorant d'autres méthodes d'optimisation combinatoire liées aux problèmes de mapping et l'utilisation de la technologie **3D**.

Bibliographie

1. Hu, J., Y. Deng, and R. Marculescu. *System-level point-to-point communication synthesis using floorplanning information [soc]*. in *Proceedings of ASP-DAC/VLSI Design 2002. 7th Asia and South Pacific Design Automation Conference and 15h International Conference on VLSI Design*. 2002. IEEE.
2. Jovanovic, S., *Architecture reconfigurable de système embarqué auto-organisé*. 2009, Université Henri Poincaré-Nancy 1.
3. Dally, W.J. and B. Towles. *Route packets, not wires: on-chip inteconnection networks*. in *Proceedings of the 38th annual design automation conference*. 2001.
4. Lemaire, R., *Conception et modélisation d'un système de contrôle d'applications de télécommunication avec une architecture de réseau sur puce (NoC)*. 2006, Institut National Polytechnique de Grenoble-INPG.
5. Chatmen, M.F., *Conception d'un réseau sur puce optimisé en latence*. 2016, Université de Bretagne Sud.
6. Brière, M., *Flot de conception hiérarchique d'un système hétérogène: Prototypage virtuel d'un réseau d'interconnexion optique intégré*. 2005, Ecully, Ecole centrale de Lyon.
7. Koch-Hofer, C., *Modélisation, validation et présynthèse de circuits asynchrones en SystemC*. 2009, Institut National Polytechnique de Grenoble-INPG.
8. De Micheli, G., et al. *Networks on chips: From research to products*. in *Proceedings of the 47th Design Automation Conference*. 2010.
9. Moraes, F., et al., *HERMES: an infrastructure for low area overhead packet-switching networks on chip.* 2004. **38**(1): p. 69-93.
10. Nilsson, E. and J.J.R.I.o.T. Oberg, Tech. Rep, *PANACEA-A case study on the PANACEA NoC-a Nostrum Network on Chip prototype.* 2006. **229**.
11. Neeb, C. and N.J.J.o.S.a. Wehn, *Designing efficient irregular networks for heterogeneous systems-on-chip.* 2008. **54**(3-4): p. 384-396.
12. Delorme, J., *Méthodologie de modélisation et d'exploration d'architecture de réseaux sur puce appliquée aux télécommunications*. 2007, INSA de Rennes.
13. Toubaline Nesrine et al ”A Classiffcation and Evaluation Framework for NoC Mapping Strategies” Journal of Circuits, Systems, and ComputersVol. 26, No. 2 (2017)
14. Dehyadgari, M., et al. *Evaluation of pseudo adaptive XY routing using an object oriented model for NOC*. IEEE.
15. Rantala, V., T. Lehtonen, and J. Plosila, *Network on chip routing algorithms*. 2006: Citeseer.
16. Andriahantenaina, A. and A. Greiner. *Micro-network for SoC: Implementation of a 32-port SPIN network*. in *2003 Design, Automation and Test in Europe Conference and Exhibition*. 2003. IEEE.
17. Balamurugan, K., et al., *Roadmap for Machine Learning based Network-on-Chip (M/L NoC) technology and its analysis for researchers.* 2022.
18. MERDJAOUI Brahim, « Optimisation multi-objectif par algorithmes génétiques et approche Pareto des paramètres d’usinage sous contraintes des limitations de productions », Thèse de Magister, Université M’HAMED BOUGARA Boumerdes, 2006.

19. Wafa, A., *OPTIMISATION MULTI-OBJECTIF: ÉTUDES DE CAS*. 2021, Université de Biskra.
20. Collette, Y. and P. Siarry, *Optimisation Multiobjectif*. 2002.
21. Nabil, M., *Contribution à l'optimisation de la puissance réactive en présence de dispositif de compensation dynamique (FACTS)*. 2013, Université Mohamed Khider-Biskra.
22. Laetitia Jourdan. Métaheuristiques Coopératives : du déterministe au stochastique. Modélisation et simulation. Université des Sciences et Technologie de Lille - Lille I, 2010.
23. Olga roudenko, « Application des algorithmes Evolutionnaires aux problèmes d'Optimisation Multi-Objectif avec contraintes», thèse de doctorat Evolutionnaires aux l'Ecole polytechnique, 2004.
24. Boussaid, I., *Perfectionnement de métaheuristiques pour l'optimisation continue*. 2013, Paris Est.
25. LABED SAID, «Méthodes bio-inspirées hybrides pour la résolution de problèmes complexes ».Thèse de Doctorat. Université Constantine 2, 2013.
26. REKKAB Mohammed Nabil, «Conception d'un simulateur de mapping dynamique sur une architecture hétérogène MP-SOC basée NOC», thèse de magistère, 2013.
27. Taileb, Selma et Abbad, Salim, "Optimisation multiobjectif des performances des réseaux sur puces" Mémoire Master, Université Blida 1,2020.
28. Yann collette, Patrick Siary, «Optimisation multiobjectif », ÉDITIONS EYROLLES, 2002.
29. BOUMAAZA Farid, « Mapping multi-objectifs d'applications intensives sur architecture MPSoC», thèse de magistère, Université d'Oran, 2013
30. Touati Lyes, « Développement d'une plateforme hétérogène de mapping dans les réseaux sur puce (NoCs)Mémoire d'ingéniorat, Ecole nationale supérieure d'informatique,2012.
31. Arnaud Liefooghe,«Métaheuristiques pour l'optimisation multiobjectif», thèse de Docteur, l'Université Lille 1,2009.
32. François-Xavier Jrisarri. Stratégies de calcul pour l'optimisation multiobjéctif des structures composites. PhD thesis, 2009.
33. Widmer, M. (2001). Les métaheuristiques : des outiles performants pour les problémes industriels. 3ème Conférence Francophone de MOdélisation et SIMulation MOSIM.
34. E. Bonomi et J.-L. Lutton ,"le recuit simulé pour la science", Juillet 1988.
35. Glover, F. ; Laguna, M. ; *Tabu search*, 1997, Kluwer Academic Publishers, Dordrecht.
36. Jlhem Boussaid. Perfectionnement de méthaheuristiques pour l'optimisation continue.PhD thesis, Paris Est, 2013.

37. E. Bonabeau, M. Dorigo, & G. Theraulaz. Swarm intelligence : from natural to artificial systems. Oxford University Press, Inc., New York, NY, USA, 1999. ISBN 0195131592.

38. Junior, L.S., N. Nedjah, and L.d.M. Mourelle. *ACO approach in static routing for network-on-chips with 3D mesh topology*. in *2013 IEEE 4th Latin American Symposium on Circuits and Systems (LASCAS)*. 2013.

39. H.-K. Hsin, E.-J. Chang, C.-H. Chao, and A.-Y. Wu, "Regional ACO-based routing for load-balancing in NoC systems," in Proc. IEEE 2nd World Congr. Nat. Biol. Inspired Comput. (NaBIC), Fukuoka Japan. 2010 pp. 370-376.

40. Amin, W., et al., *Performance evaluation of application mapping approaches for network-on-chip designs.* 2020. **8**: p. 63607-63631

41. Haghbayan, M.-H., et al., *MapPro: Proactive Runtime Mapping for Dynamic Workloads by Quantifying Ripple Effect of Applications on Networks-on-Chip*, in *Proceedings of the 9th International Symposium on Networks-on-Chip*. 2015, Association for Computing Machinery: Vancouver, BC, Canada. p. Article 26.

42. Reza, M.F., D. Zhao, and M. Bayoumi. *Dark silicon-power-thermal aware runtime mapping and configuration in heterogeneous many-core NoC*. in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2017.

43. Sadeghi, M.S., S.B. Sarmadi, and S. Hessabi, *Toward On-chip Network Security Using Runtime Isolation Mapping.* 2019. **16**(3 %J ACM Trans. Archit. Code Optim.): p. Article 28.

44. Chao, H.-L., S.-Y. Tung, and P.-A. Hsiung, *Dynamic Task Mapping with Congestion Speculation for Reconfigurable Network-on-Chip.* 2016. **10**(1 %J ACM Trans. Reconfigurable Technol. Syst.): p. Article 3.

45. Bender, A., *MILP based task mapping for heterogeneous multiprocessor systems*, in *Proceedings of the conference on European design automation*. 1996, IEEE Computer Society Press: Geneva, Switzerland. p. 190–197.

46. Murali, S., L. Benini, and G.D. Micheli, *Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees*, in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*. 2005, Association for Computing Machinery: Shanghai, China. p. 27–32.

47. Srinivasan, K., K.S. Chatha, and G. Konjevod. *Linear programming based techniques for synthesis of network-on-chip architectures*. in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.* 2004.

48. Chen-Ling, C. and R. Marculescu. *Contention-aware application mapping for Network-on-Chip communication architectures*. in *2008 IEEE International Conference on Computer Design*. 2008.

49. Tosun, S., *Cluster-based application mapping method for Network-on-Chip.* Advances in engineering software (1992), 2011. **42**(10): p. 868-874.

50. Hu, J. and R. Marculescu, *Energy-aware mapping for tile-based NoC architectures under performance constraints*, in *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*. 2003, Association for Computing Machinery: Kitakyushu, Japan. p. 233–239.

51. Ting-Jung, L., L. Shu-Yen, and A.Y. Wu. *Traffic-balanced IP mapping algorithm for 2D-mesh On-Chip-Networks*. in *2008 IEEE Workshop on Signal Processing Systems*. 2008.
52. Khan, S., et al., *Bandwidth-Constrained Multi-Objective Segmented Brute-Force Algorithm for Efficient Mapping of Embedded Applications on NoC Architecture.* IEEE Access, 2018. **6**: p. 11242-11254.
53. Tang, L. and S. Kumar. *A two-step genetic algorithm for mapping task graphs to a network on chip architecture*. in *Euromicro Symposium on Digital System Design, 2003. Proceedings.* 2003.
54. Wenbiao, Z., Z. Yan, and M. Zhigang. *An application specific NoC mapping for optimized delay*. in *International Conference on Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006.* 2006.
55. Benyamina, A., et al. *Mapping real time applications on NoC architecture with hybrid multi- objective algorithm*. in *International Conference on Metaheuristics and Nature Inspired, Computing*. 2010. Citeseer.
56. Sahu, P.K., et al. *Application Mapping onto Mesh Structured Network-on-Chip Using Particle Swarm Optimization*. in *2011 IEEE Computer Society Annual Symposium on VLSI*. 2011.
57. Fekr, A.R., et al. *Bandwidth/fault tolerance/contention aware application-specific NoC using PSO as a mapping generator*. in *Proc. of The World Congress on Engineering*. 2010.
58. Roy, A., K. Manna, and S. Chattapadhay. *Effect of core ordering on application mapping onto mesh based network-on-chip design*. in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2015.
59. Sahu, P.K., A. Sharma, and S. Chattopadhyay. *Application Mapping Onto Mesh-of-Tree Based Network-on-Chip Using Discrete Particle Swarm Optimization*. in *2012 International Symposium on Electronic System Design (ISED)*. 2012.
60. Obaidullah, M. and G.N. Khan, *Application mapping to mesh NoCs using a Tabu-search based swarm optimization.* Microprocessors and Microsystems, 2017. **55**: p. 13-25.
61. Upadhyay, M., et al. *Multi-application Based Network-on-Chip Design for Mesh-of-Tree Topology Using Global Mapping and Reconfigurable Architecture*. in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*. 2019.
62. Wang, J., et al., *Bandwidth-aware application mapping for NoC-based MPSoCs.* 2011. **7**(1): p. 152- 159.
63. Yanhua, L., et al. *Energy and thermal aware mapping for mesh-based NoC architectures using multi-objective ant colony algorithm*. in *2011 3rd International Conference on Computer Research and Development*. 2011.
64. Murali, S. and G.D. Micheli. *Bandwidth-constrained mapping of cores onto NoC architectures*. in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*. 2004.
65. Srinivasan, K. and K.S. Chatha, *A technique for low energy mapping and routing in network-on-chip architectures*, in *Proceedings of the 2005 international symposium on Low power electronics and design*. 2005, Association for Computing Machinery: San Diego, CA, USA. p. 387–392.
66. Lu, Z., L. Xia, and A. Jantsch. *Cluster-based Simulated Annealing for Mapping Cores onto 2D Mesh Networks on Chip*. in *2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. 2008.

67. Janidarmian, M., A. Khademzadeh, and M.J.I.E.E. Tavanpour, *Onyx: A new heuristic bandwidth-constrained mapping of cores onto tile-based Network on Chip.* 2009. **6**(1): p. 1-7.
68. Saeidi, S., A. Khademzadeh, and F.J.I.E.E. Vardi, *Crinkle: A heuristic mapping algorithm for network on chip.* 2009. **6**(24): p. 1737-1744.
69. Sahu, P.K., et al. *A new application mapping algorithm for mesh based Network-on-Chip design*. in *2010 Annual IEEE India Conference (INDICON)*. 2010.
70. Alikhah-Asl, E. and M. Reshadi. *XY-axis and distance based NoC mapping (XY-ADB)*. in *2016 8th International Symposium on Telecommunications (IST)*. 2016.
71. Koziris, N., et al. *An efficient algorithm for the physical mapping of clustered task graphs onto multiprocessor architectures*. in *Proceedings 8th Euromicro Workshop on Parallel and Distributed Processing*. 2000.
72. Saeidi, S., A. Khademzadeh, and A. Mehran. *SMAP: An Intelligent Mapping tool for Network on Chip*. in *2007 International Symposium on Signals, Circuits and Systems*. 2007.
73. Shen, W.-T., et al. *A new binomial mapping and optimization algorithm for reduced-complexity mesh-based on-chip network*. in *First International Symposium on Networks-on-Chip (NOCS'07)*. 2007. IEEE.
74. Patooghy, A., H. Tabkhi, and S.G. Miremadi. *RMAP: A Reliability-Aware Application Mapping for Network-on-Chips*. in *2010 Third International Conference on Dependability*. 2010.
75. Tosun, S., *New heuristic algorithms for energy aware application mapping and routing on mesh-based NoCs.* Journal of Systems Architecture, 2011. **57**(1): p. 69-78.
76. Cheng, C.-H. and W.-M. Chen, *Application mapping onto mesh-based network-on-chip using constructive heuristic algorithms.* The Journal of Supercomputing, 2016. **72**(11): p. 4365-4378.
77. Sharma, P.K., S. Biswas, and P. Mitra, *Energy efficient heuristic application mapping for 2-D mesh-based network-on-chip.* Microprocessors and Microsystems, 2019. **64**: p. 88-100.
78. K.Manna, J.Mathew "Design and Test Strategies for 2D/3D Integration for NoC-based Multicore Architectures" Springer Nature Switzerland AG 2020
79. R. J. Gutmannet al.,“Three-dimensional (3D) ICs : A technology platform for integrated systems and opportunities for new polymeric adhesives,"inProc. Conf. Polymers Adhesives Microelectron. Photon.,2001, pp. 173–180.
80. Pavlidis VF , Friedman EG . 3d topologies for networks-on-chip. IEEE Trans Very Large Scale Integr Syst 2007;15(10):1081–90 .
81. Naddunoori, M. and M. D (2020). Topological exploration for the efficient design of three-dimensional Network on Chip architectures. 2020 Third International Conference on Advances in Electronics, Computers and Communications (ICAECC).
82. Vivet, P., et al. (2011). 3D NoC using through silicon Via: An asynchronous implementation. 2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip.
83. Amir-Mohammad Rahmani ; Pasi Liljeberg ; Juha Plosila ; Hannu Tenhunen ,“BBVC-3D-NoC: ane_cient 3D NoC architecture using bidirectional bisynchronous vertical channels”. In IEEE Computer Society Annual Symposium on VLSI, pages 452-453, 2010.

84. Igor Loi ; Subhasish Mitra ; Thomas H. Lee ; Shinobu Fujita ; Luca Benini, “A low-overhead fault tolerance scheme for TSVbased 3D networkon chip links”. International Conference on Computer-Aided Design,pp.598–602, 2008.
85. Panem, C., et al. (2018). Evaluation of 3D & 4D Two Layer Mesh NoC for Fault Tolerance over Combinations of Vertical Channel. 2018.
86. Agyeman, M.O., A. Ahmadinia, and N. Bagherzadeh, *Energy and performance-aware application mapping for inhomogeneous 3D networks-on-chip.* Journal of Systems Architecture, 2018. **89**: p. 103-117.
87. Li, B., et al., *On Runtime Communication and Thermal-Aware Application Mapping and Defragmentation in 3D NoC Systems.* IEEE Transactions on Parallel and Distributed Systems, 2019. **30**(12): p. 2775-2789.
88. Alagarsamy, A., L. Gopalakrishnan, and S.-B. Ko, *KBMA: A knowledge-based multi-objective application mapping approach for 3D NoC.* 2019. **13**(4): p. 324-334.
89. N.Toubaline. *Aide a La Conception d ’ un Reseau Sur puce pour un système integre*. Université de Saad dahleb Blida. (2018).
90. Gupta, M., Bhargava, L. & Indu, S. Mapping techniques in multicore processors: current and future trends. *J Supercomput* **77**, 9308–9363 (2021).
91. F.BOUMAAZA ”Mapping multi-objectifs d’applications intensives sur architecture MPSoC" thèse de magistère 2013
92. Yann collette, Patrick Siary, «Optimisation multiobjectif », ÉDITIONS EYROLLES, 2002.
93. Buckler, M. et al. (2013). Low-power Networks-on-Chip : Progress and remaining challenges. IEEE International Symposium on Low Power Electronics and Design (ISLPED)
94. ZIELINSKI K., LAUR R., Stopping Criteria for Differential Evolution in Constrained Single-Objective Optimization. In : Advanced in Differential Evolution, the series Studies in Computational Intelligence, vol. 143, pp. 111-138 Springer, Berlin Heidelberg. 2008
95. He, T. and Y. J. E. R. M. č. n. p. o. i. s. a. a. k. Guo, materijala i novih tehnologija u području strojarstva, brodogradnje, temeljnih tehničkih znanosti, elektrotehnike, računarstva i građevinarstva (2013). "Power consumption optimization and delay based on ant colony algorithm in network-on-chip." **33**(3): 219-225.
96. Robert P. Dick, Embedded System Synthesis Benchmarks Suite (E3S),2008.
97. P. K. Sahu, et al ,"Application mapping onto mesh structured network-on-chip using particle swarm optimization," in Proc. IEEE Int. Symp. Very Large Scale Integr.Syst. 2011.
98. Sahu, P. K., et al. (2011). Application Mapping onto Mesh Structured Network-on-Chip Using Particle Swarm Optimization. 2011 IEEE Computer Society Annual Symposium on VLSI.
99. Murali, S. and G. D. Micheli (2004). Bandwidth-constrained mapping of cores onto NoC architectures.Proceedings Design, Automation and Test in Europe Conference and Exhibition.
100. Dick, R. P., et al. (1998). TGFF: task graphs for free.
101. Vallerio, K. (2008). Task Graphs for Free (TGFF v3 . 0). *Processing*, 1–9.
102. Manna, K., Mathew, J. (2020). A Constructive Heuristic for Designing a 3D NoC-Based Multi-Core Systems.In: Design and Test Strategies for 2D/3D Integration for NoC-based Multicore Architectures.

103. Sahu, P. K. and S. Chattopadhyay (2013). "A survey on application mapping strategies for Network-on-Chip design." Journal of Systems Architecture 59(1): 60-76.

Annexe A : Les résultats des archives

Les archives suivantes représentent les résultats obtenus par les deux méthodes **MOACO-1** et **MOACO-2** en fonction du coût de communication et de nombre de **TSVs**.

/********** BENCHMARK VOPD **********/

MOACO-1	MOACO-2
Debut	Debut
Solution : 1	**Solution : 1**
Fourmi : 444	Fourmi : 407
N_Cycle : 14	N_Cycle : 83
Coût_C : 7465.0	Coût_C : 6669.0
Coût_S : 4	Coût_S : 7
Mapping : [2, 5, 3, 8, 16, 13, 12, 11, 1, 4, 7, 10, 14, 6, 15, 9]	Mapping : [11, 6, 4, 3, 13, 10, 8, 16, 14, 5, 7, 2, 12, 9, 1, 15]
Positions_TSVs : [1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0]	Positions_TSVs : [0, 1, 2, 5, 6, 3, 4, 7, 0, 1, 2, 5, 6, 3, 4, 7]
Solution : 2	**Solution : 2**
Fourmi : 495	Fourmi : 134
N_Cycle : 116	N_Cycle : 195
Coût_C : 6824.0	Coût_C : 7901.0
Coût_S : 7	Coût_S : 3
Mapping : [4, 5, 6, 12, 16, 2, 8, 9, 14, 13, 7, 1, 3, 15, 10, 11]	Mapping : [5, 4, 3, 13, 9, 8, 14, 15, 16, 6, 11, 2, 1, 10, 7, 12]
Positions_TSVs : [1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1]	Positions_TSVs : [0, 1, 2, 0, 0, 3, 0, 0, 0, 1, 2, 0, 0, 3, 0, 0]
Solution : 3	**Solution : 3**
Fourmi : 369	Fourmi : 962
N_Cycle : 213	N_Cycle : 214
Coût_C : 11110.0	Coût_C : 7352.0
Coût_S : 1	Coût_S : 4
Mapping : [8, 5, 2, 14, 11, 9, 12, 13, 6, 10, 4, 15, 7, 3, 1, 16]	Mapping : [11, 10, 3, 2, 15, 9, 5, 6, 14, 8, 4, 16, 12, 13, 7, 1]
Positions_TSVs : [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	Positions_TSVs : [0, 1, 2, 0, 0, 3, 4, 0, 0, 1, 2, 0, 0, 3, 4, 0]
Solution : 4	**Solution : 4**
Fourmi : 275	Fourmi : 394
N_Cycle : 252	N_Cycle : 287
Coût_C : 8169.0	Coût_C : 10817.0
Coût_S : 3	Coût_S : 1
Mapping : [11, 12, 9, 10, 3, 5, 4, 14, 16, 7, 8, 13, 1, 6, 2, 15]	Mapping : [6, 10, 3, 2, 1, 13, 16, 12, 4, 8, 5, 7, 14, 9, 15, 11]
Positions_TSVs : [0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0]	Positions_TSVs : [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
Solution : 5	**Solution : 5**
Fourmi : 406	Fourmi : 690
N_Cycle : 268	N_Cycle : 309
Coût_C : 7040.0	Coût_C : 6937.0
Coût_S : 6	Coût_S : 5
Mapping : [15, 16, 4, 2, 9, 1, 6, 11, 14, 13, 3, 12, 10, 8, 7, 5]	Mapping : [16, 8, 6, 3, 11, 15, 1, 2, 9, 10, 5, 4, 12, 14, 7, 13]
Positions_TSVs : [0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0]	Positions_TSVs : [0, 1, 2, 5, 0, 3, 4, 0, 0, 1, 2, 5, 0, 3, 4, 0]
Solution : 6	**Solution : 6**
Fourmi : 685	Fourmi : 2
N_Cycle : 424	N_Cycle : 389
Coût_C : 7398.0	Coût_C : 8671.0
Coût_S : 5	Coût_S : 2
Mapping : [9, 3, 16, 13, 8, 7, 14, 15, 10, 5, 4, 2, 12, 6, 11, 1]	Mapping : [9, 8, 3, 5, 12, 7, 6, 14, 13, 10, 4, 1, 16, 11, 2, 15]
Positions_TSVs : [1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0]	Positions_TSVs : [0, 1, 2, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0]
	Fin

Solution : 7 Fourmi : 280 N_Cycle : 492 Coût_C : 4852.0 Coût_S : 8 Mapping : [14, 4, 16, 15, 11, 5, 6, 7, 3, 2, 9, 1, 13, 12, 10, 8] Positions_TSVs : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] **Solution : 8** Fourmi : 322 N_Cycle : 497 Coût_C : 8744.0 Coût_S : 2 Mapping : [9, 8, 7, 16, 13, 6, 3, 1, 12, 5, 10, 11, 14, 4, 2, 15] Positions_TSVs : [0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0] Fin	

/********** **BENCHMARK MPEG** **********/

MOACO-1	**MOACO-2**
Debut **Solution : 1** Fourmi : 209 N_Cycle : 110 Coût_C : 5675.0 Coût_S : 5 Mapping : [2, 1, 10, 6, 11, 12, 9, 4, 5, 3, 8, 7] Positions_TSVs : [0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1] **Solution : 2** Fourmi : 957 N_Cycle : 114 Coût_C : 5795.5 Coût_S : 3 Mapping : [3, 4, 9, 11, 5, 7, 6, 1, 8, 2, 10, 12] Positions_TSVs : [0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1] **Solution : 3** Fourmi : 140 N_Cycle : 115 Coût_C : 5955.5 Coût_S : 2 Mapping : [7, 4, 3, 12, 11, 6, 10, 5, 1, 8, 9, 2] Positions_TSVs : [1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0] **Solution : 4** Fourmi : 781 N_Cycle : 191 Coût_C : 7193.0 Coût_S : 1 Mapping : [3, 12, 2, 11, 5, 1, 6, 10, 9, 4, 7, 8] Positions_TSVs : [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0] **Solution : 5** Fourmi : 420 N_Cycle : 243 Coût_C : 3772.5	Debut **Solution : 1** Fourmi : 356 N_Cycle : 52 Coût_C : 7193.0 Coût_S : 1 Mapping : [11, 7, 10, 3, 1, 2, 4, 5, 12, 9, 8, 6] Positions_TSVs : [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0] **Solution : 2** Fourmi : 863 N_Cycle : 121 Coût_C : 5714.5 Coût_S : 2 Mapping : [8, 12, 6, 1, 10, 2, 9, 7, 11, 4, 5, 3] Positions_TSVs : [0, 1, 0, 0, 2, 0, 0, 1, 0, 0, 2, 0] **Solution : 3** Fourmi : 428 N_Cycle : 391 Coût_C : 5555.0 Coût_S : 4 Mapping : [3, 10, 6, 9, 7, 11, 1, 5, 4, 2, 12, 8] Positions_TSVs : [0, 1, 3, 4, 2, 0, 0, 1, 3, 4, 2, 0] **Solution : 4** Fourmi : 955 N_Cycle : 497 Coût_C : 5635.5 Coût_S : 3 Mapping : [11, 12, 8, 3, 10, 2, 9, 7, 6, 1, 5, 4] Positions_TSVs : [0, 1, 3, 0, 2, 0, 0, 1, 3, 0, 2, 0] Fin

Coût_S : 6 Mapping : [11, 1, 3, 8, 2, 9, 10, 5, 4, 7, 12, 6] Positions_TSVs : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] **Solution : 6** Fourmi : 472 N_Cycle : 458 Coût_C : 5734.5 Coût_S : 4 Mapping : [2, 10, 1, 9, 7, 8, 4, 5, 6, 3, 12, 11] Positions_TSVs : [0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1] Fin	

/********** BENCHMARK MWD **********/

MOACO-1	MOACO-2
Debut **Solution : 1** Fourmi : 624 N_Cycle : 23 Coût_C : 1280.0 Coût_S : 6 Mapping : [8, 9, 10, 5, 12, 11, 4, 3, 7, 1, 2, 6] Positions_TSVs : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] **Solution : 2** Fourmi : 554 N_Cycle : 192 Coût_C : 2528.0 Coût_S : 2 Mapping : [8, 3, 6, 9, 4, 7, 11, 10, 2, 5, 1, 12] Positions_TSVs : [0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0] **Solution : 3** Fourmi : 789 N_Cycle : 231 Coût_C : 2208.0 Coût_S : 3 Mapping : [3, 1, 12, 7, 5, 4, 6, 2, 11, 10, 9, 8] Positions_TSVs : [0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0] **Solution : 4** Fourmi : 478 N_Cycle : 233 Coût_C : 2080.0 Coût_S : 4 Mapping : [12, 5, 8, 2, 1, 10, 6, 4, 11, 3, 7, 9] Positions_TSVs : [0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1] **Solution : 5** Fourmi : 985 N_Cycle : 242 Coût_C : 3168.0 Coût_S : 1 Mapping : [4, 7, 3, 8, 5, 10, 12, 1, 11, 9, 2, 6]	Debut **Solution : 1** Fourmi : 344 N_Cycle : 1 Coût_C : 3168.0 Coût_S : 1 Mapping : [8, 2, 9, 11, 1, 3, 6, 5, 7, 12, 10, 4] Positions_TSVs : [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0] **Solution : 2** Fourmi : 991 N_Cycle : 23 Coût_C : 2208.0 Coût_S : 3 Mapping : [3, 6, 10, 8, 1, 9, 7, 2, 11, 4, 5, 12] Positions_TSVs : [0, 1, 3, 0, 2, 0, 0, 1, 3, 0, 2, 0] **Solution : 3** Fourmi : 256 N_Cycle : 125 Coût_C : 2496.0 Coût_S : 2 Mapping : [1, 2, 4, 6, 9, 12, 8, 5, 3, 7, 10, 11] Positions_TSVs : [0, 1, 0, 0, 2, 0, 0, 1, 0, 0, 2, 0] **Solution : 4** Fourmi : 747 N_Cycle : 355 Coût_C : 2016.0 Coût_S : 4 Mapping : [12, 3, 5, 7, 10, 4, 6, 2, 1, 11, 9, 8] Positions_TSVs : [0, 1, 3, 4, 2, 0, 0, 1, 3, 4, 2, 0] Fin

Positions_TSVs : [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0] **Solution : 6** Fourmi : 47 N_Cycle : 459 Coût_C : 1952.0 Coût_S : 5 Mapping : [4, 1, 2, 11, 12, 7, 5, 8, 3, 9, 10, 6] Positions_TSVs : [1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1] Fin	

/********** BENCHMARK PIP **********/

MOACO-1	MOACO-2
Debut **Solution : 1** Fourmi : 84 N_Cycle : 1 Coût_C : 640.0 Coût_S : 4 Mapping : [3, 8, 2, 1, 4, 7, 5, 6] Positions_TSVs : [1, 1, 1, 1, 1, 1, 1, 1] **Solution : 2** Fourmi : 125 N_Cycle : 1 Coût_C : 1536.0 Coût_S : 1 Mapping : [5, 2, 8, 6, 1, 7, 3, 4] Positions_TSVs : [0, 1, 0, 0, 0, 1, 0, 0] **Solution : 3** Fourmi : 103 N_Cycle : 28 Coût_C : 896.0 Coût_S : 3 Mapping : [3, 6, 1, 5, 4, 7, 2, 8] Positions_TSVs : [1, 1, 1, 0, 1, 1, 1, 0] **Solution : 4** Fourmi : 358 N_Cycle : 59 Coût_C : 1024.0 Coût_S : 2 Mapping : [5, 2, 7, 8, 6, 1, 4, 3] Positions_TSVs : [0, 1, 1, 0, 0, 1, 1, 0] Fin	Debut **Solution : 1** Fourmi : 12 N_Cycle : 1 Coût_C : 1536.0 Coût_S : 1 Mapping : [5, 7, 4, 1, 6, 2, 8, 3] Positions_TSVs : [0, 1, 0, 0, 0, 1, 0, 0] **Solution : 2** Fourmi : 422 N_Cycle : 6 Coût_C : 1024.0 Coût_S : 2 Mapping : [3, 1, 7, 8, 4, 2, 6, 5] Positions_TSVs : [0, 1, 2, 0, 0, 1, 2, 0] **Solution : 3** Fourmi : 431 N_Cycle : 73 Coût_C : 896.0 Coût_S : 3 Mapping : [8, 4, 2, 5, 3, 7, 1, 6] Positions_TSVs : [0, 1, 2, 3, 0, 1, 2, 3] Fin

/********** BENCHMARK TG7 **********/

MOACO-1	MOACO-2
Debut **Solution : 1** Fourmi : 816 N_Cycle : 24 Coût_C : 145600.0 Coût_S : 6 Mapping : [24, 19, 26, 16, 5, 17, 12, 8, 13, 27, 14, 10, 9, 1, 2, 15, 23, 29, 25, 30, 11, 22, 18, 6, 28, 4, 20, 3, 7, 21] Positions_TSVs : [0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0] **Solution : 2** Fourmi : 614 N_Cycle : 129 Coût_C : 117500.0 Coût_S : 14 Mapping : [12, 16, 11, 4, 14, 23, 9, 2, 22, 7, 30, 29, 21, 18, 27, 6, 15, 5, 1, 26, 13, 10, 8, 25, 24, 28, 19, 3, 20, 17] Positions_TSVs : [1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] **Solution : 3** Fourmi : 42 N_Cycle : 173 Coût_C : 127900.0 Coût_S : 12 Mapping : [22, 5, 7, 3, 30, 11, 26, 13, 15, 20, 14, 28, 2, 12, 25, 21, 1, 29, 17, 27, 19, 4, 8, 23, 9, 18, 16, 10, 24, 6] Positions_TSVs : [1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0] **Solution : 4** Fourmi : 283 N_Cycle : 184 Coût_C : 125100.0 Coût_S : 13 Mapping : [7, 21, 27, 22, 12, 18, 10, 14, 17, 24, 13, 11, 4, 6, 26, 23, 28, 25, 8, 20, 19, 29, 1, 9, 16, 3, 5, 15, 30, 2] Positions_TSVs : [1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1] **Solution : 5** Fourmi : 467 N_Cycle : 239 Coût_C : 170000.0 Coût_S : 2 Mapping : [20, 19, 21, 7, 27, 8, 3, 10, 14, 15, 12, 6, 13, 26, 28, 29, 23, 17, 22, 16, 11, 2, 30, 25, 24, 9, 4, 1, 18, 5] Positions_TSVs : [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0] **Solution : 6** Fourmi : 360 N_Cycle : 249 Coût_C : 97600.0 Coût_S : 15	Debut **Solution : 1** Fourmi : 63 N_Cycle : 3 Coût_C : 154000.0 Coût_S : 3 Mapping : [26, 3, 14, 25, 8, 7, 29, 16, 2, 30, 27, 18, 24, 11, 28, 22, 6, 9, 1, 21, 15, 20, 4, 19, 17, 5, 10, 23, 12, 13] Positions_TSVs : [0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0] **Solution : 2** Fourmi : 836 N_Cycle : 117 Coût_C : 128500.0 Coût_S : 10 Mapping : [10, 23, 24, 15, 9, 8, 29, 6, 20, 18, 5, 7, 13, 12, 21, 4, 25, 26, 17, 22, 27, 2, 3, 11, 30, 16, 28, 14, 19, 1] Positions_TSVs : [0, 7, 0, 6, 2, 8, 4, 1, 5, 9, 3, 10, 0, 0, 0, 0, 7, 0, 6, 2, 8, 4, 1, 5, 9, 3, 10, 0, 0, 0] **Solution : 3** Fourmi : 651 N_Cycle : 138 Coût_C : 147200.0 Coût_S : 6 Mapping : [15, 24, 1, 27, 16, 23, 9, 18, 17, 3, 4, 5, 7, 30, 29, 22, 12, 8, 21, 11, 28, 19, 25, 13, 26, 6, 14, 10, 2, 20] Positions_TSVs : [0, 0, 0, 6, 2, 0, 4, 1, 5, 0, 3, 0, 0, 0, 0, 0, 0, 0, 6, 2, 0, 4, 1, 5, 0, 3, 0, 0, 0, 0] **Solution : 4** Fourmi : 823 N_Cycle : 233 Coût_C : 190100.0 Coût_S : 1 Mapping : [19, 3, 21, 26, 27, 20, 6, 8, 16, 5, 12, 25, 7, 10, 28, 14, 30, 1, 29, 18, 22, 17, 4, 11, 9, 24, 23, 2, 13, 15] Positions_TSVs : [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] **Solution : 5** Fourmi : 941 N_Cycle : 268 Coût_C : 153600.0 Coût_S : 4 Mapping : [15, 12, 25, 11, 9, 17, 5, 28, 6, 14, 4, 30, 2, 23, 22, 27, 20, 18, 13, 19, 16, 1, 21, 7, 3, 8, 10, 26, 24, 29] Positions_TSVs : [0, 0, 0, 0, 2, 0, 4, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 4, 1, 0, 0, 3, 0, 0, 0, 0] **Solution : 6** Fourmi : 549 N_Cycle : 289 Coût_C : 124400.0 Coût_S : 11

Mapping : [17, 26, 25, 14, 24, 1, 22, 6, 8, 19, 12, 23, 7, 28, 30, 27, 5, 2, 11, 4, 3, 20, 10, 9, 21, 16, 13, 15, 18, 29] Positions_TSVs : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] **Solution : 7** Fourmi : 161 N_Cycle : 339 Coût_C : 130800.0 Coût_S : 11 Mapping : [4, 17, 26, 12, 27, 29, 28, 13, 3, 19, 20, 5, 7, 25, 23, 16, 2, 24, 30, 18, 11, 15, 10, 14, 21, 6, 9, 1, 8, 22] Positions_TSVs : [1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0] **Solution : 8** Fourmi : 526 N_Cycle : 458 Coût_C : 192600.0 Coût_S : 1 Mapping : [25, 18, 22, 3, 7, 13, 5, 6, 16, 12, 9, 21, 19, 23, 10, 28, 15, 30, 26, 20, 2, 11, 17, 8, 27, 4, 24, 29, 1, 14] Positions_TSVs : [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] **Solution : 9** Fourmi : 204 N_Cycle : 546 Coût_C : 132000.0 Coût_S : 9 Mapping : [13, 17, 14, 19, 8, 29, 23, 20, 15, 7, 28, 1, 12, 4, 24, 5, 11, 30, 9, 6, 22, 3, 21, 16, 18, 27, 2, 10, 26, 25] Positions_TSVs : [0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0] **Solution : 10** Fourmi : 232 N_Cycle : 551 Coût_C : 140000.0 Coût_S : 7 Mapping : [21, 4, 9, 2, 8, 29, 22, 25, 3, 20, 7, 16, 28, 19, 11, 6, 10, 26, 24, 12, 17, 27, 23, 15, 5, 18, 1, 14, 30, 13] Positions_TSVs : [1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1] **Solution : 11** Fourmi : 109 N_Cycle : 572 Coût_C : 153300.0 Coût_S : 4 Mapping : [1, 13, 28, 10, 14, 8, 2, 12, 6, 23, 25, 5, 17, 29, 26, 3, 19, 7, 22, 20, 21, 11, 16, 9, 18, 27, 15, 30, 4, 24] Positions_TSVs : [0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0] **Solution : 12** Fourmi : 110 N_Cycle : 648 Coût_C : 151300.0 Coût_S : 5	Mapping : [7, 29, 19, 22, 9, 3, 20, 5, 4, 26, 13, 17, 15, 2, 6, 28, 30, 18, 8, 21, 1, 25, 12, 27, 11, 14, 10, 16, 23, 24] Positions_TSVs : [0, 7, 0, 6, 2, 8, 4, 1, 5, 9, 3, 10, 0, 11, 0, 0, 7, 0, 6, 2, 8, 4, 1, 5, 9, 3, 10, 0, 11, 0] **Solution : 7** Fourmi : 855 N_Cycle : 369 Coût_C : 138800.0 Coût_S : 7 Mapping : [25, 3, 14, 22, 4, 2, 23, 28, 29, 20, 5, 16, 21, 13, 17, 24, 10, 27, 26, 6, 1, 8, 12, 9, 19, 11, 15, 7, 18, 30] Positions_TSVs : [0, 7, 0, 6, 2, 0, 4, 1, 5, 0, 3, 0, 0, 0, 0, 0, 7, 0, 6, 2, 0, 4, 1, 5, 0, 3, 0, 0, 0, 0] **Solution : 8** Fourmi : 585 N_Cycle : 387 Coût_C : 134400.0 Coût_S : 9 Mapping : [27, 5, 13, 10, 4, 15, 29, 26, 17, 7, 24, 2, 23, 3, 25, 12, 18, 16, 30, 11, 28, 14, 1, 6, 22, 9, 19, 21, 8, 20] Positions_TSVs : [0, 7, 0, 6, 2, 8, 4, 1, 5, 9, 3, 0, 0, 0, 0, 0, 7, 0, 6, 2, 8, 4, 1, 5, 9, 3, 0, 0, 0, 0] **Solution : 9** Fourmi : 662 N_Cycle : 529 Coût_C : 115900.0 Coût_S : 13 Mapping : [25, 4, 26, 15, 23, 13, 16, 12, 28, 3, 10, 18, 21, 7, 30, 2, 24, 17, 8, 29, 11, 20, 1, 27, 9, 6, 5, 19, 22, 14] Positions_TSVs : [0, 7, 12, 6, 2, 8, 4, 1, 5, 9, 3, 10, 13, 11, 0, 0, 7, 12, 6, 2, 8, 4, 1, 5, 9, 3, 10, 13, 11, 0] **Solution : 10** Fourmi : 371 N_Cycle : 643 Coût_C : 137400.0 Coût_S : 8 Mapping : [14, 15, 16, 27, 17, 18, 19, 28, 7, 3, 4, 24, 13, 9, 23, 6, 25, 12, 29, 11, 8, 20, 5, 21, 1, 2, 30, 26, 22, 10] Positions_TSVs : [0, 7, 0, 6, 2, 8, 4, 1, 5, 0, 3, 0, 0, 0, 0, 0, 7, 0, 6, 2, 8, 4, 1, 5, 0, 3, 0, 0, 0, 0] **Solution : 11** Fourmi : 530 N_Cycle : 658 Coût_C : 168900.0 Coût_S : 2 Mapping : [29, 27, 2, 14, 11, 5, 8, 26, 7, 20, 12, 23, 10, 22, 9, 15, 17, 30, 1, 4, 13, 21, 18, 6, 16, 3, 25, 19, 28, 24] Positions_TSVs : [0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] **Solution : 12** Fourmi : 394 N_Cycle : 678 Coût_C : 149500.0 Coût_S : 5

Mapping : [28, 18, 30, 8, 3, 20, 27, 26, 13, 4, 11, 2, 17, 10, 16, 1, 25, 7, 21, 19, 9, 5, 24, 22, 23, 29, 14, 6, 15, 12] Positions_TSVs : [0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1] **Solution : 13** Fourmi : 183 N_Cycle : 766 Coût_C : 159500.0 Coût_S : 3 Mapping : [23, 8, 4, 14, 27, 9, 2, 25, 5, 28, 6, 29, 22, 1, 16, 15, 11, 24, 13, 26, 10, 20, 17, 19, 30, 12, 3, 18, 21, 7] Positions_TSVs : [0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0] Fin	Mapping : [20, 3, 19, 16, 30, 7, 25, 11, 9, 26, 21, 14, 5, 22, 24, 2, 1, 12, 28, 18, 6, 17, 13, 4, 27, 8, 10, 29, 15, 23] Positions_TSVs : [0, 0, 0, 0, 2, 0, 4, 1, 5, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 4, 1, 5, 0, 3, 0, 0, 0, 0] Fin