

**République algérienne démocratique et
populaire**
**Ministère de l'enseignement supérieur et de
la recherche scientifique**
Université Saad Dahlab Blida 1



Faculté des sciences
Département informatique

**Mémoire de fin d'études pour l'obtention du
diplôme de Master en Informatique**
Option : Système informatique et réseaux

***L'algorithme des chauves-souris pour le
mapping des Applications sur Noc.***

Réalisé par :

Aba Abdenour
Amrouche Wafa

Encadré par :

Mme Boumahdi
Fatima

Année universitaire 2021-2022

Résumé

Au cours des dernières décennies, les systèmes sur puces (Soc) ont été fréquemment utilisés pour équiper des systèmes et des dispositifs complexes, ils permettent d'intégrer un ensemble de composants sur une seule puce, communiquant entre eux par une structure de communication (point à point, bus partage et cross-bar). Cependant, avec la complexité de nouvelles technologies et ses tailles, ces systèmes ont trouvé leurs limites. Un nouveau paradigme est apparu comme une solution viable aux faiblesses des interconnexions traditionnelles, connu sous le nom de réseau sur puce (Noc).

Un Noc se compose d'un ensemble de ressources (R) et de routeurs (S) sont connectés entre eux par des liens. La paire (R ; S) forme une tuile. La phase de mapping est une étape critique dans le processus de conception de réseau sur puce. En fait, un placement incorrect des composants logiciels d'une application peut réduire considérablement les performances globales du système final. L'étape de mapping est un problème NP-difficile, auquel cas l'utilisation de métaheuristiques est un bon moyen d'obtenir une solution optimale.

L'objectif principal de notre projet est de développer une nouvelle technique de mapping des IP dans une architecture réseau sur puce afin de minimiser les coûts de communication, l'énergie de communication et la latence. La nouvelle solution est basée sur une méta-heuristique appelée *l'algorithme des chauves-souris*, où nous considérons des architectures Noc homogènes (c'est-à-dire que les nœuds de calcul sont tous les mêmes) avec différentes topologies : 2D, 3D et 2D wifi.

Mot clés : Réseau sur puce (Noc), problème d'optimisation, Mapping, Algorithme de chauves-souris (Bat).

Abstract

In recent decades, chip-based systems (Soc) have been frequently used to equip complex systems and devices, allowing a set of components to be integrated into a single chip, communicating with each other through a communication structure (point to point, bus sharing and crossbar). However, with the complexity of new technologies and their size, these systems have found their limits. A new paradigm emerged as a viable solution to the weaknesses of traditional interconnections, known as network on chip (NOC).

A Noc consists of a set of resources (R) and router (S) are connected together by links. The pair (R; S) forms a tile. The mapping phase is a critical step in the smart grid design process. In fact, incorrect placement of software components in an application can significantly reduce the overall performance of the final system. The mapping step is a NP-difficult problem, in which case the use of metaheuristics is a good way to obtain an optimal solution.

The main objective of our project is to develop a new IP mapping technique in a network-on-chip architecture in order to minimize communication costs, communication energy and latency. The new solution is based on a meta-heuristic called the bat algorithm, where we consider homogeneous Noc architectures (that is, the calculation nodes are all the same) with different topologies: 2D, 3D and 2DWIFI.

Keywords: Network on chip (Noc), optimization problem, Mapping, Bat algorithm (bat Algorithm).

المخلص

في العقود الأخيرة، تم استخدام الأنظمة القائمة على الشرائح بشكل متكرر لتجهيز الأنظمة والأجهزة المعقدة، مما يسمح بدمج مجموعة من المكونات في شريحة واحدة، والتواصل مع بعضها البعض من خلال بنية اتصال. ومع ذلك، مع تعقيد التقنيات الجديدة وحجمها، وجدت هذه الأنظمة حدودها. ظهر نموذج جديد كحل قابل للتطبيق لنقاط الضعف في الترابط التقليدي، المعروف باسم الشبكة على الرقاقة.

يتكون Noc من مجموعة من الموارد والموجه متصلان ببعضهما البعض عن طريق الروابط. الزوج (الموارد، الموجه) يشكل بلاطة. تعد مرحلة رسم الخرائط خطوة حاسمة في عملية تصميم الشبكة الذكية. في الواقع، يمكن أن يؤدي التنسيب غير الصحيح لمكونات البرنامج في أحد التطبيقات إلى تقليل الأداء العام للنظام النهائي بشكل كبير. تعد خطوة رسم الخرائط مشكلة صعبة، وفي هذه الحالة يكون استخدام الخصائص الوصفية طريقة جيدة للحصول على الحل الأمثل.

يتمثل الهدف الرئيسي لمشروعنا في تطوير تقنية جديدة لرسم خرائط في بنية شبكة على شريحة لتقليل تكاليف الاتصالات وطاقة الاتصال وزمن الانتقال. يعتمد الحل الجديد على يسمى خوارزمية، حيث نعتبر معماريات المتجانسة (أي أن عقد الحساب كلها متشابهة) مع طوبولوجيا مختلفة.

الكلمات المفتاحية: الشبكة على الرقاقة، مشكلة التحسين، الخرائط، خوارزمية الخفافيش.

Remerciement

Nous remercions en premier lieu Dieu le tout puissant de nous avoir donné le courage, la patience et la force de réaliser ce projet de fin d'étude.

Nous adressons nos sincères remerciements à notre promotrice Mme Boumahdi pour sa patience, sa disponibilité et surtout son soutien moral et son encouragement, qui ont contribué à alimenter notre réflexion.

Nous tenons à remercier très chaleureusement toute l'équipe pédagogique du département d'informatique de l'université Saad Dahleb (USDH), qui ont assurés notre formation tout au long des (05) années d'études, et nous ont transmis leur savoir sans réserve.

Nous tenons à remercier également Ms Boughrara qui nous a accordé de son temps et nous a conseillé.

Nous présentons nos respects et nos sincères remerciements aux membres du jury qui nous ont fait l'honneur d'avoir accepté de faire partie du jury et de nous avoir consacré de leur temps précieux.

Enfin, nous tenons à témoigner toute nos gratitudes à toutes les personnes ayant participé de près ou de loin et plus particulièrement à nos familles à la réalisation de ce projet.

Dédicaces

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

À mes parents qui m'a toujours offert le bonheur.

*À mon support dans la vie celui qui ont été
toujours à mes côtés*

Mes sœurs Ikram , safa & Hana

*À ceux avec qui j'ai passé les meilleurs moments de
mon master :*

*Wissam , Hana , Wiam , Ferdaous , Riyadh et
Lynda*

*À tous mes enseignants de département
informatique , particulièrement
Ms.ould khaoua et Mme.Zahra .*

Wafa

Dédicaces

A mes chers parents.

A mes sœurs Imane & Mélissa.

A mon frère Mounir.

*A toute ma famille, particulièrement Ma
grand-mère.*

A tous mes amis.

Abdenour

Table des matières

<i>Introduction Générale</i>	<i>1</i>
Chapitre 1 : Réseaux sur puce (Noc)	5
1.1 Introduction.....	6
1.2 Système sur puce.....	6
1.3 Réseaux sur puce.....	6
1.4 Architecture en couches.....	7
1.5 Les principaux composants d'un réseau sur	9
1.5.1 IP (Intellectual Properties).....	9
1.5.2 Liens.....	9
1.5.3 Routeurs	10
1.5.4 Interfaces réseaux	11
1.6 Modes de commutation	12
1.6.1 Commutation par Circuit.....	13
1.6.2 Commutation par paquet.....	13
1.7 Topologies et ses Classification	15
1.7.1 Régulière & Irrégulière	16
1.7.2 Directe & indirecte	17
1.8 Topologie émergente de Noc.....	21
1.8.1 Réseau sue puce wifi (W-Noc).....	21
1.8.2 Topologie 3D.....	22
1.9 Algorithmes de Routage.....	23
1.10 Les paramètres de performance	24
1.11 Conclusion	25
Chapitre 2 : Travaux Connexes	27
2.1 Introduction.....	28
2.2 Les phase de conception d'un Noc.....	28
2.3 Problème de Mapping.....	29
2.3.1 Mapping Dynamique	30
2.3.2 Mapping Statique	30
2.3.3 Les déficit de Mapping	30
2.4 Travaux liers	31

2.5	Conclusion	36
Chapitre 3 : Conception & Implémentation		38
3.1	Introduction.....	39
3.2	Contexte de projet	39
3.2.1	Problème d’optimisation.....	39
3.2.2	Méta-heuristique & Heuristique	41
3.3	Architecture de la solution proposée.....	42
3.3.1	Module 1 : Applications.....	44
3.3.2	Module 2 : Topologies	49
3.3.3	Présentation de F1 & F2	57
3.3.4	Algorithme des chauves-souris	58
3.4	Conclusion	66
Chapitre 4 : Tests & Résultats.....		67
4.1	Introduction.....	68
4.2	Outils d’implémentation utilisés	68
4.3	La génération des Benchmark : TGFF	68
4.4	Les Benchmark utilisés	68
4.4.1	PIP.....	69
4.4.2	MDW.....	69
4.4.3	VODP.....	70
4.4.4	MPEG4	70
4.4.5	Benchmark Aléatoire	71
4.5	Présentation des interfaces.....	72
4.6	Tests et résultats	76
4.6.1	Etude paramétrique	76
4.6.2	Etude comparative entre 2D, 2D wifi et 3D	79
4.6.3	La comparaison avec la littérature	82
4.7	Conclusion	91
Conclusion Générale.....		92
Bibliographies.....		94

Tables des figures

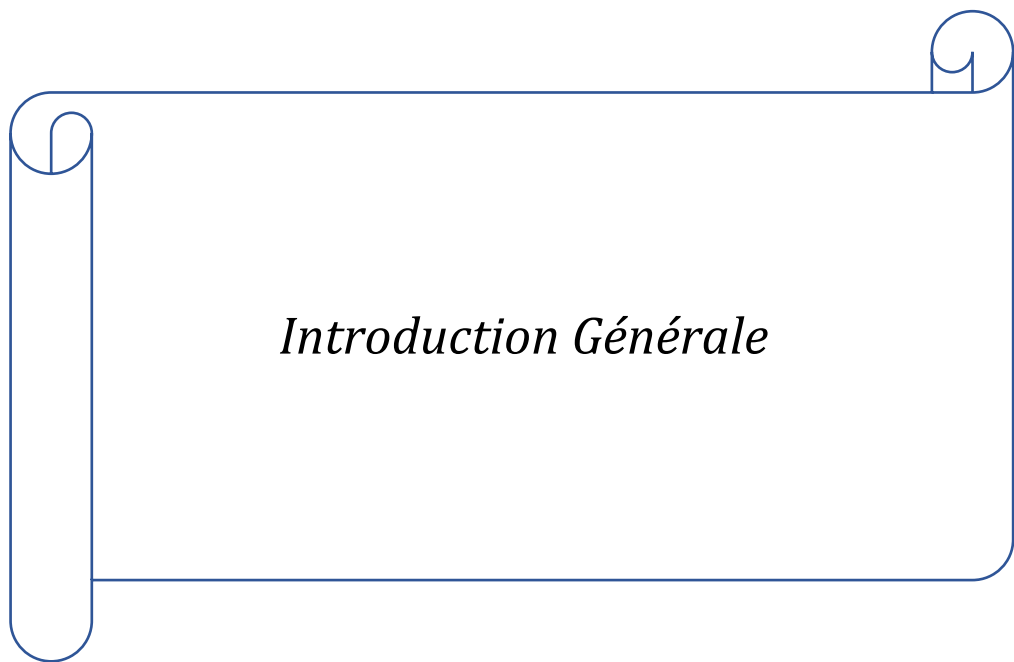
FIGURE 2-1: COUCHES OSI.(<i>CHARIET, 2014</i>)	8
FIGURE 2-2: COMPOSANTS DE RESEAU SUR PUCE.(<i>BOUGUETTAYA, 2017</i>).....	9
FIGURE 2-3: ARCHITECTURE D'UN ROUTEUR.(<i>DJABALLAH, 2011</i>).....	10
FIGURE 2-4: ARCHITECTURE D'UN ROUTEUR AVEC LES POSTIONS DE BUFFER.(<i>BOUGUETTAYA ET AL., 2016</i>)	11
FIGURE 2-5: ARCHITECTURE NA.(<i>DJABALLAH, 2011</i>)	12
FIGURE 2-6: TYPES DE MODE COMMUTATION.(<i>ISIKA ET AL., 2013</i>).....	13
FIGURE 2-7: STORE AND FORWARD.(<i>DELORME, 2007</i>)	14
FIGURE 2-8: TECHNIQUE VCT.(<i>DELORME, 2007</i>).....	15
FIGURE 2-9: TECHNIQUE WORMHOLE.(<i>DELORME, 2007</i>).....	15
FIGURE 2-10: TOPOLOGIE REGULIERE.(<i>TATAS ET AL., 2012</i>).....	17
FIGURE 2-11: TOPOLOGIE IRREGULIERE.(<i>TATAS ET AL., 2012</i>).....	17
FIGURE 2-12 : TOPOLOGIE 2D MESH.(<i>TATAS ET AL., 2012</i>)	19
FIGURE 2-13: TOPOLOGIE TORE.(<i>COTA ET AL, 2012</i>)	19
FIGURE 2-14: TOPOLOGIE ANNEAU.(<i>TATAS ET AL., 2012</i>).....	20
FIGURE 2-15: ARBRE ELARGIE .(<i>TOUATI, 2012</i>)	20
FIGURE 2-16: ARBRE ELARGI EN PAPILLON.(<i>TOUATI, 2012</i>).....	21
FIGURE 2-17: EMPILEMENT DE PLUSIEURS PUCES PAR WIRE BONDING.(<i>SHEIBANYRD & PETROT, 2014</i>)	22
FIGURE 2-18: TOPOLOGIE 3D AVEC LIEN TSV.(<i>MICHAEL ET AL, 2018</i>).....	23
FIGURE 2-19: LES TOURS ASSOCIES A CHAQUE ALGORITHME.(<i>CHARIET,</i> <i>2014</i>).....	24
FIGURE 2-1: ORGANIGRAMME DES ETAPES DE CONCEPTION DE NOC.(<i>LEMAIRE, 2006</i>)	29
FIGURE 2-2: TYPES MAPPING.....	30
FIGURE 3-1: CLASSIFICATION DES METHODES D'OPTIMISATION.	40

FIGURE 3-2:SCHEMA DE TRAVAIL	43
FIGURE 3-3:GRAPHE DES TACHES	44
FIGURE 3-4: STRUCTURE XML	46
FIGURE 3-5 : SCHÉMA DTD	46
FIGURE 3-6:DIAGRAMME DES CLASSES	48
FIGURE 3-7:DIAGRAMME DES CLASSES DES TOPOLOGIE	49
FIGURE 3-8:REPRESENTATION DE LA CLASSE 2D	50
FIGURE 3-9:PSEUDO CODE DE LA FONCTION GENERER ()	51
FIGURE 3-10:PSEUDO CODE DE LA FONCTION GENERER () 2DWIFI	52
FIGURE 3-11: REPRESENTATION DE CAS IMPAIRE * IMPAIRE	53
FIGURE 3-12:REPRESENTATION DE CAS PAIR*IMPAIR	53
FIGURE 3-13:CAS IMPAIR*PAIR.....	54
FIGURE 3-14:CAS PAIR*PAIR.....	ERROR! BOOKMARK NOT DEFINED.
FIGURE 3-15:REPRESENTATION DE LA STRUCTURE 3D	56
FIGURE 3-16:PSEUDO CODE DE FONCTION GENERER 3 D ()	57
FIGURE 3-17:REPRESENTATION DE PROBLEME DE MAPPING	ERROR! BOOKMARK NOT DEFINED.
FIGURE 3-18: L'ORGANIGRAMME DE L'ALGORITHME DE CHAUVES- SOURIS.(SAPPA & DORNAIKA, 2019).....	59
FIGURE 3-19: FORMULATION DE SOLUTION.....	60
FIGURE 3-20:PSEUDO CODE DE LA FONCTION GENERER UNE POPULATION	61
FIGURE 3-21:CHOISIR LA SOLUTION GLOBALE	65
FIGURE 3-22:EXEMPLE DE FICHIER XML DE LA SOLUTION FINALE.....	66
FIGURE 4-1:APPLICATION PIP.(TOUBALINE & AL, 2017)	69
FIGURE 4-2:APPLICATION MDW.(TOUBALINE & AL, 2017)	69
FIGURE 4-3:APPLICATION VODP.(TOUBALINE & AL, 2017)	70
FIGURE 4-4:APPLICATION MPEG4.(TOUBALINE & AL, 2017)	71
FIGURE 4-5: INTERFACE D'ACCUEIL	ERROR! BOOKMARK NOT DEFINED.
FIGURE 4-6:CHOIX DE BENCHMARK	73
FIGURE 4-7:AFFICHAGE DE GRAPHE	73
FIGURE 4-8:CHOIX DE TOPOLOGIE	74
FIGURE 4-9:PARAMETRES BAT	75
FIGURE 4-10:AFFICHAGE RESULTAT	76

FIGURE 4-11: HISTORIQUE.	ERROR! BOOKMARK NOT DEFINED.
FIGURE 4-12: HISTOGRAMME REPRESENTE LA VARIANCE DE PARAMETRES ENTRE LES BENCHMARKS	79
FIGURE 4-13: COMPARAISON GRAPHIQUE DE TEST 4.....	ERROR! BOOKMARK NOT DEFINED.
FIGURE 4-14: REPRESENTATION GRAPHIQUE DE TEST 5.....	82
FIGURE 4-15: COMPARAISON BAT AVEC GABC SUR 2D.....	84
FIGURE 4-16: COMPARAISON GRAPHIQUE DE BAT ...AVEC ABC SUR 2D.	85
FIGURE 4-17: COMPARAISON BAT AVEC BFO SUR 2D.....	ERROR! BOOKMARK NOT DEFINED.
FIGURE 4-18: COMPARAISON GRAPHIQUE ENTRE BAT ET GABC SUR 3D.....	87
FIGURE 4-19: COMPARAISON ENTRE BAT ET ABC SUR 3D.....	88
FIGURE 4-20: COMPARAISON ENTRE BAT ET BFO SUE 3D.....	89

Listes des Tableaux

TABLEAU 1:COMPARAISON ENTRE LES ARTICLES	36
TABLEAU 2:BENCHMARKS ALEATOIRE.....	71
TABLEAU 3:RESULTAT DES TESTS DE POPULATION & NOMBRE ITERATION.	78
TABLEAU 4:RESULTAT D'ETUDE PARAMETRIQUE.	79
TABLEAU 5:RESULTAT DE TEST1	80
TABLEAU 6:RESULTAT DE TEST 2	80
TABLEAU 7:RESULTAT DE TEST 3	81
TABLEAU 8:RESULTAT TEST 4.....	81
TABLEAU 9:RESULTAT TEST 5	82
TABLEAU 10:COMPARAISON ENTRE BAT ET GABC SUR 2D.....	83
TABLEAU 11:COMPARAISON BAT AVEC ABC SUR 2D.....	84
TABLEAU 12: COMPARAISON BAT AVEC BFO SUR 2D.....	85
TABLEAU 13:COMPARAISON ENTRE BAT ET GABC SUR 3D.....	86
TABLEAU 14:COMPARAISON ENTRE BAT ET ABC SUR 3D.....	87
TABLEAU 15: COMPARAISON ENTRE BAT ET BFO SUR 3D.....	88
TABLEAU 16:COMPARAISON ENTRE RESULTATS DE BAT ET LES METHODES DE LITTERATEUR.....	90



Typiquement, Un système sur puce, regroupe plusieurs composants complexes et hétérogènes tels que des processeurs programmables, des mémoires, des interfaces d'entrée/sortie, du matériel personnalisé, des périphériques, des blocs d'interface IP (propriété intellectuelle) externes, Parmi les problèmes majeurs durant la conception des Soc c'est le choix de la structure d'interconnexion entre ces éléments.(Khan et al., 2018)

Depuis 1990, la structure d'interconnexion utilisée dans les Soc est la communication à bus partage (Zerioul,2015), où tous les composants sont liés à un seul support de transport qui ne permet qu'une seule communication à la fois gérée par un arbitre. Lorsque le nombre de cœurs participants est supérieur à dix, le système de bus aura un problème de goulot d'étranglement de performances en raison de sa limitation de bande passante et ne répond pas aux besoins des nouvelles technologies où il commence à bloquer le trafic. La structure des bus commence à être étroite et elle sera moins utilisée. Des autres structures ont été également proposée afin de résoudre ces faiblesses comme : l'interconnexion entièrement cross bar et interconnexion point à point, mais elles présentent également des problèmes tels que la flexibilité et la sécurité. Pour surmonter ces défis, les concepteurs ont mis au point un nouveau concept appelé Network-on-Chip, qui s'inspire des réseaux informatiques standard.

Aujourd'hui, la majorité des applications et appareil repose sur la notion de Noc, car elle est un facteur crucial. Par rapport aux autres architectures, cette structure offre une très grande bande passante, Une diversité des chemins, le parallélisme des communications, la flexibilité, l'extensibilité et la réutilisation facile des IPs. (Bouguettaya et al., 2016)

Problématique :

Lors de la conception des réseaux sur puce il faut prendre en compte plusieurs des paramètres, tel que le choix de topologie, le choix de la technologie d'intégration et le placement des composants communicants (mapping),Ce dernier est un domaine

important qui intéresse les chercheurs et qui fait l'objet de notre travail de recherches au cours de ce mémoire.

Ce problème consiste à trouver un emplacement pour les IPs sur les tuiles d'une architecture Noc. Ce dernier est connu comme un problème NP difficile. Ceci est encore plus difficile lorsqu'il y a plusieurs facteurs souvent contradictoires à prendre en compte comme : la communication vs l'équilibrage de charge, la latence vs l'énergie de consommation, ces décisions du Mapping peuvent avoir une influence considérable sur la performance du système final.

Objectif :

Le but de notre travail est de trouver une solution qui maximise les performances des réseaux sur puce dans un cadre mono-objectifs.

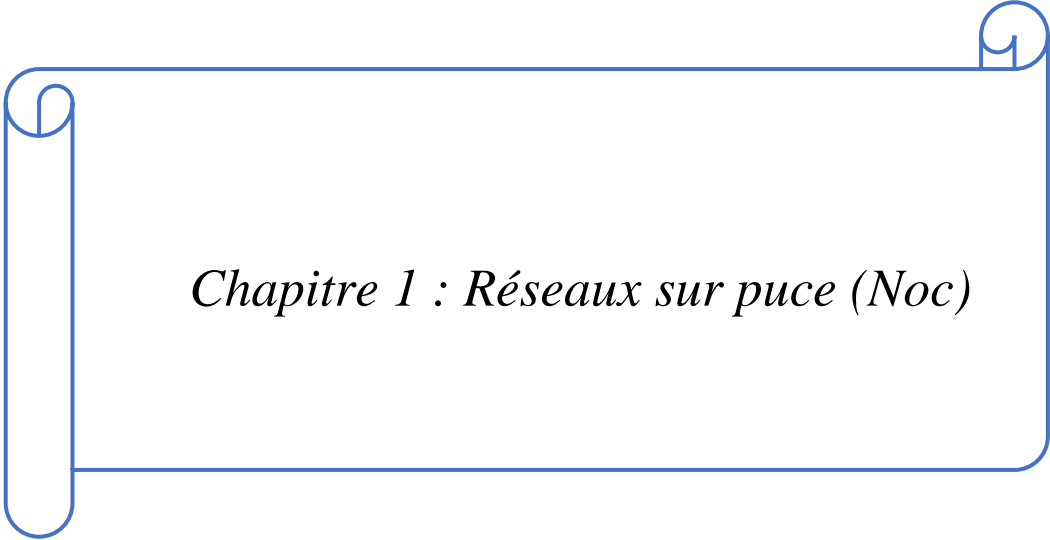
Le problème de mapping est un problème d'optimisation difficile à résoudre et à trouver sa solution optimale, les métaheuristiques sont donc de bons candidats pour résoudre ce problème, nous proposons un algorithme appelé algorithme de chaves de souris, qui permet un mapping statique d'applications parallèles sur une architecture homogène de Noc afin de minimiser le coût de communication.

Plan de mémoire :

Notre mémoire est constitué de 4 chapitres :

- Le chapitre 1 introduit le principe du réseau sur puce (Noc) et donne quelques généralisations, Nous allons commencer par une définition de Noc et ses composants, ainsi que les principaux paramètres qui jouent un rôle important dans l'évaluation de leur performance, telles que la topologie, les modèles de commutation et les algorithmes de routage, enfin, les paramètres de performance.
- Dans le chapitre 2, il sera intéressant de discuter et de définir le problème de mapping et ses types, puis nous présenterons quelques travaux connexes et comparerons les méthodes utilisées pour obtenir une vue d'ensemble de notre travail.

- Dans le chapitre 3, nous allons présenter notre phase conception et formulation de problématique, ainsi que l'adaptation de l'algorithme de chaves souris sur notre problème.
- Dans le chapitre 4, nous allons présenter tous les outils et les Framework utilisée pour la réalisation de ce travail, et les différents tests et résultats obtenus.



Chapitre 1 : Réseaux sur puce (Noc)

1.1 Introduction

Les systèmes embarqués sont aujourd'hui utilisés dans la plupart des systèmes informatiques et des appareils électroniques : téléphones portables, voitures, téléviseurs, avions, appareils médicaux, qui sont équipés de microcontrôleurs, de logiciels et de systèmes d'exploitation en temps réel, qui sont basés sur des systèmes sur puce (Soc).

Du succès des Soc et de leur évolution rapide et de plus en plus complexe en taille, ainsi que les rôles multiples des éléments présents sur une même puce, les exigences de communication et d'interconnexion des blocs IP (Intellectual Property) à l'intérieur de la puce deviennent de plus en plus importantes et constituent une partie fondamentale lors de la conception de tels systèmes, et plusieurs facteurs, tels que la réduction du temps de mise sur le marché, la flexibilité et les besoins de bande passante, nécessitent une approche de conception appropriée. Par conséquent, les chercheurs ont adopté un nouveau concept, le réseau sur puce, pour répondre à la complexité croissante et aux besoins de communication des futurs systèmes sur puce. Ces réseaux sont conçus à l'image des réseaux informatiques traditionnels, mais avec certaines normes, qui sont générées par l'environnement de ces réseaux.

Dans ce chapitre nous allons définir les réseaux sur puce, leur composant, architecture et aussi les algorithmes de routage et les principales topologies, mais avant ça, nous avons présenté c'est quoi les systèmes sur puce pour mieux comprendre le fonctionnement de ce concept.

1.2 Système sur puce

Un système sur puce (System on Chip en anglais, Soc) peut intégrer sur une seule et même puce des processeurs, des mémoires, des réseaux d'interconnexions et des circuits spécifiques. Il existe différents types de systèmes d'interconnexion utilisables dans un Soc dont le point-à-point, le bus partagé ou hiérarchique, le cross-bar et le réseau sur puce. (Claasen, 2006)

1.3 Réseaux sur puce

La technologie réseau sur puce (*network on chip en anglais, Noc*) est une approche relativement nouvelle de la signalisation qui permet non seulement des interconnexions plus efficaces, mais également des processus de conception et de

vérification plus efficaces pour les systèmes sur puce moderne. Elle est une approche de la signalisation qui correspond aux besoins du signal à divers protocoles de communication d'une manière qui réduit la complexité de l'interconnexion de la puce. Les signaux lents ou à faible bande passante peuvent être multiplexés sur une seule ligne avec d'autres signaux, tandis que seuls les signaux à la vitesse la plus élevée et à la bande passante la plus élevée communiquent directement sur des chemins parallèles encombrants (Tatas et al., 2014).

1.4 Architecture en couches

Le réseau sur puce est un réseau de communication à commutation de paquets provenant du domaine informatique parallèle. La communication globale sur puce est décomposée en couches similaires au modèle de référence OSI vu dans les réseaux informatiques (Figure 1.1). Le modèle ISO/OSI ne définit pas exactement comment un système doit être construit, mais sert plutôt de guide conceptuel (Tatas et al., 2014). Plus précisément, en permettant à chaque couche de cacher sa propre complexité aux couches supérieures et de ne communiquer qu'avec les couches adjacentes (Tatas et al., 2014). Dans ce modèle, la fonction de réseau est divisée en sept couches différentes, de bas en haut, la couche physique, la couche liaison de données, la couche réseau, la couche transport, la couche session, la couche présentation et la couche application :

La couche application : C'est dans cette couche que se trouvent les ressources (IPs). (Tatas et al., 2014)

La couche présentation : Comme les ressources IP peuvent avoir différentes représentations de données, il doit y avoir certaines opérations de conversion entre elle. Ces opérations sont localisées dans la couche présentation (Tatas et al., 2014).

La couche session : En utilisant la couche transport, la couche session sera responsable de l'établissement de connexions entre les ressources. Elle adapte les protocoles de l'IP et du NoC (Tatas et al., 2014).

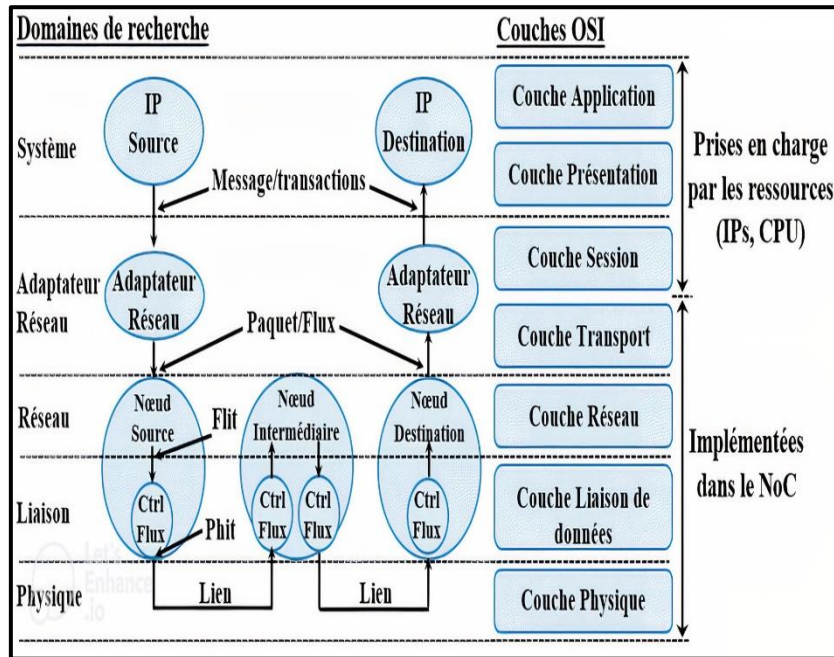


Figure 1-1: Couches OSI.(Chariet, 2014)

La couche transport : Cette couche est chargée de la transformation des messages en paquets, et inversement. Elle assure le contrôle de flux de bout en bout afin d'éviter que des données ne soient envoyées vers une destination qui n'a plus de place disponible pour les recevoir. De plus elle gère l'adaptation de protocole entre le NoC et les IP via l'adaptateur réseau(Tatas et al., 2014).

La couche réseau : La couche réseau gère les problèmes liés à la topologie et au routage qui en résulte schème(Chariet, 2014).

Couche de liaison ou commutation : Utilise la couche physique pour implémenter le mécanisme de transmission de données à travers le réseau (échange de données entre les différents routeurs avec détection et correction d'erreur de transmission)(Tatas et al., 2014).

Couche physique : Définit la structure physique et les protocoles nécessaires (nature d'inter connexion et largeur en nombre de bits des mots de données) pour établir la communication au niveau des liens entre les différents routeurs (Tatas et al., 2014).

Cette classification peut être résumés comme suit : (Tatas et al., 2014)

- ✓ Les couches doivent échanger un minimum d'informations à travers les interfaces.
- ✓ Il doit y avoir suffisamment de couches pour que des fonctions non liées ne soient pas insérées la même couche.
- ✓ Chaque couche doit avoir des limites avec sa couche supérieure et inférieure uniquement.

1.5 Les principaux composants d'un réseau sur

Un réseau sur puce est constitué de quatre principaux composants (Figure 1.2), Le premier et le plus important sont les IPs, Le deuxième liens qui relient physiquement les nœuds et mettent réellement en œuvre la communication, Le troisième est le routeur qui permet d'acheminer les données à partir d'une source vers une destination, le dernier composant est l'interface réseau qui fait la connexion logique entre les cœurs IP et le réseau, puisque chaque IP peut avoir un protocole d'interface distinct par rapport au réseau (Cota et al., 2012).

1.5.1 IP (Intellectual Properties)

Les IP sont n'importe quels composants tels qu'un microprocesseur, un circuit intégré spécifique à une application (ASIC), une mémoire, ou une combinaison de composants reliés entre eux. Ces IP sont reliées à des routeurs réseau via une interface réseau (Bouguettaya, 2017).

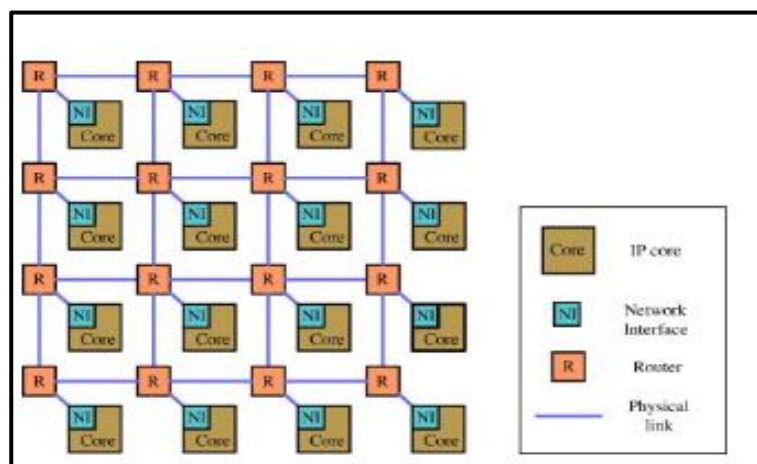


Figure 2-2: Composants de réseau sur puce. (Bouguettaya, 2017)

1.5.2 Liens

Les liens résument les connexions entre l'interface réseau et le routeur, et entre le routeur lui-même. Ils fournissent une bande passante pour la communication entre la source et la destination et peuvent être unidirectionnels ou bidirectionnels. (Cota

et al., 2012) Un lien peut être constitué de plusieurs canaux physiques ou logiques pour réduire les taux de congestion lors des communications(Leroy, 2007).

1.5.3 Routeurs

Le rôle des routeurs est d'aiguiller les données qu'il reçoit vers leurs destinataires, son architecture dépend de la topologie du Noc(figure1.3). Généralement, Il est constitué d'un ensemble de files d'attente (buffer Fifo) à chaque port, un cross-bar (matrice de commutation) qui relie les ports d'entrée avec ceux de sortie, une unité de routage et d'arbitrage(Bouguettaya, 2017) .

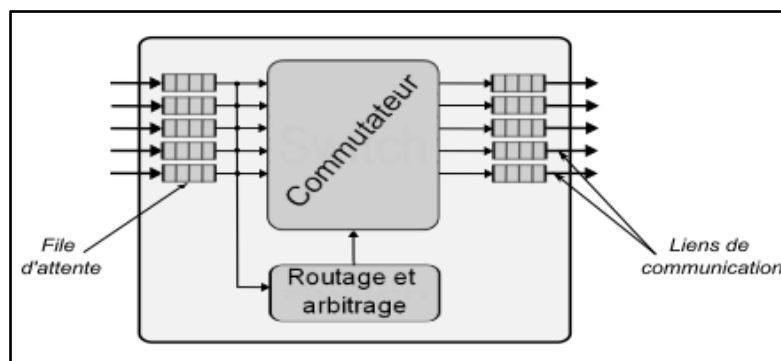


Figure 2-3:Architecture d'un routeur.(Djaballah, 2011)

- **File d'attente :** Utilisée pour le stockage temporaire des données transmises, sa taille est très importante car elle affecte directement le délai de routage, la perte de paquets, la consommation d'énergie et la surface, selon le mode de commutation choisi, les files d'attente peuvent contenir plusieurs paquets, un seul paquet, ou une partie d'un paquet. Concernant leur largeur, elle ne peut être inférieure d'une unité de contrôle de flux, Ils peuvent être placés en entrée, en sortie, ou encore en sortie virtuelle (*figure 1.4*)(Palesi & Daneshtalab, 2014) .

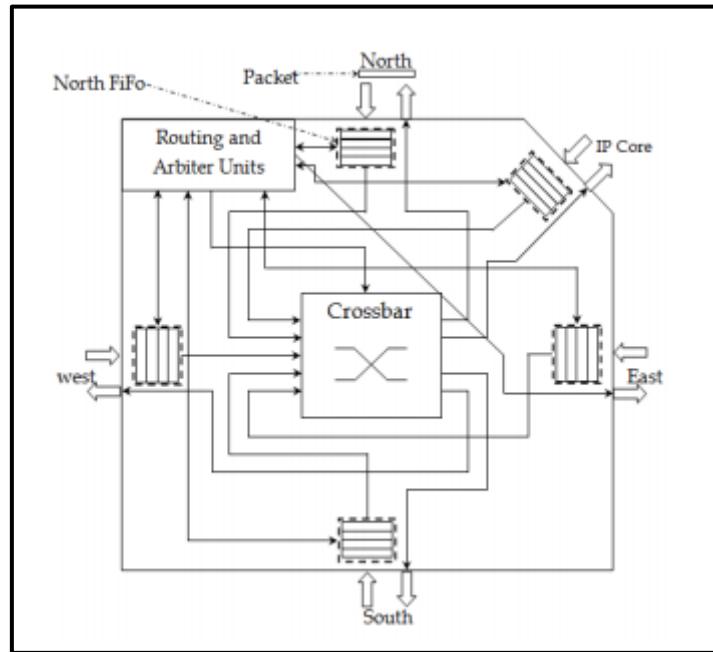


Figure 2-4: Architecture d'un routeur avec les positions de Buffer.(Bouguettaya et al., 2016)

- **Matrice de commutation :** est un commutateur ne bloque pas qui active des connexions entre l'entrée et la sortie d'un routeur. Il est utilisé pour envoyer le paquet du tampon d'entrée vers le port de sortie approprié en fonction de l'adresse contenue dans l'en-tête du paquet(Bouguettaya et al., 2016) .
- **Unité de routage :** permet de décoder les unités d'information envoyées par les messages entrants. Il est basé sur l'algorithme de routage et la destination du message. Sa fonction principale est de sélectionner le port de sortie le plus approprié pour la transmission des messages(Bouguettaya et al., 2016).
- **Unité d'arbitrage :** est utilisée pour déterminer quel paquet aura la priorité lorsque plusieurs paquets provenant de différentes sources sont en concurrence pour la transmission sur la même interconnexion. Par conséquent, lorsqu'un paquet se déplace de la source à la destination, il traverse plusieurs liaisons et passe par un ou plusieurs routeurs du réseau NoC(Bouguettaya et al., 2016).

1.5.4 Interfaces réseaux

Le quatrième composant de construction NoC est l'interface réseau (NI) ou l'adaptateur réseau (NA). Il fait la connexion logique entre les cœurs IP et le réseau. Ce composant est important car il permet la séparation entre le calcul et la

communication. Cela permet la réutilisation à la fois de l'infrastructure centrale et de l'infrastructure de communication indépendamment l'une de l'autre (Bjerregaard & Mahadevan, 2006).

L'adaptateur peut être divisé en deux parties : une partie frontale (front end) une partie arrière (back end). (Figure 1.5) Le frontal implémente généralement un protocole de communication point à point standardisé. La partie arrière gère le protocole réseau (assemble et désassemble le paquet, réorganise les tampons, met en œuvre des protocoles de synchronisation, aide le routeur en termes de stockage, ect) (Cota et al., 2012).

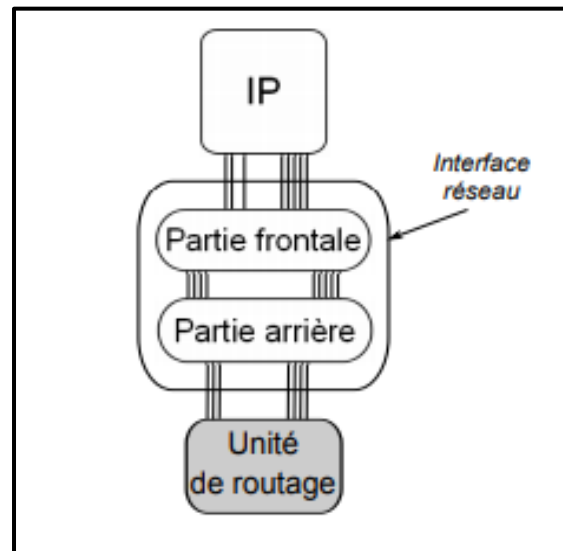


Figure 2-5: Architecture NA. (Djaballah, 2011)

1.6 Modes de commutation

Les techniques de commutation déterminent le moment et la manière dont les données sont transmises du nœud source au nœud cible (destinataire). Il existe deux principaux types (figure 1.6) de commutation dans le NoC : la commutation par circuit et la commutation par paquet (Isiaka et al., 2013).

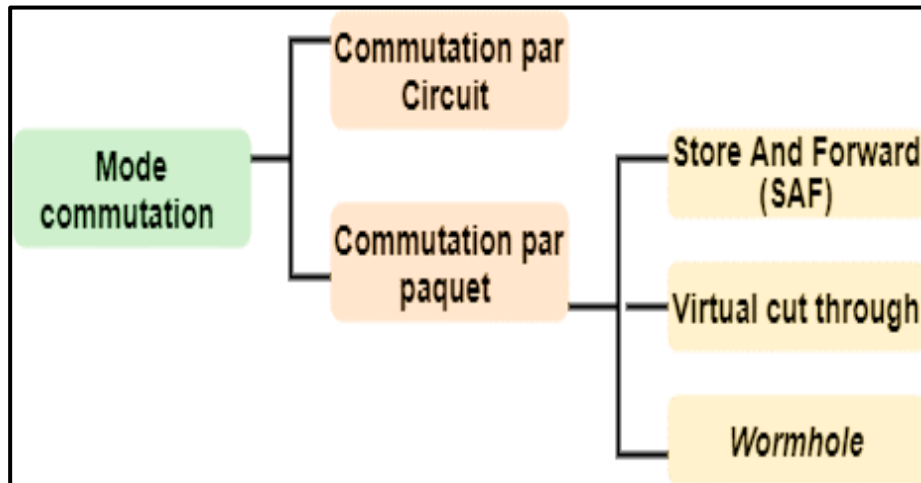


Figure 2-6:types de mode commutation.(isiaka et al., 2013)

1.6.1 Commutation par Circuit

La commutation de circuit (*circuit switching : en anglais*) est basée sur l'établissement d'un chemin physique réservé, composé de routeurs et de liens, entre la source et la destination avant la transmission des données. Elle peut être utilisée aussi pour améliorer l'évolutivité du réseau.

Bien que la commutation de circuits offre des transferts à faible latence en raison de l'utilisation complète de la bande passante des liaisons et l'envoi des données sans les mettre en mémoire tampon, elle gaspille les liaisons établies lorsqu'il n'y a pas de transmission de données, ce qui entraîne des problèmes d'évolutivité(isiaka et al., 2013).

1.6.2 Commutation par paquet

Ce mode divise un message en paquets, chaque paquet contient deux parties : une partie appelée l'en-tête pour le contrôle et la deuxième partie qui est l'information transportée. Les routeurs analysent et appliquent des modifications sur les en-têtes des paquets qui les traversent sur leurs ports, et guident le flux de données vers le port de sortie ciblé par les en-têtes de paquets.

D'autre part, cette technique implique que le paquet entier doit être stocké dans le routeur avant de décider d'acheminer les données vers le port de sortie, pour lequel une stratégie de commutation est requise. Il définit comment les ressources réseau doivent être allouées et libérées une fois le transfert terminé. Trois principales

stratégies de commutation sont distinguées : Store And Forward (SAF), Wormhole (WH) et Virtual Cut Through (VCT)(Palesi & Daneshtalab, 2014).

- Store and Forward: en commutation SAF, les paquets sont stockés en mémoire tampon FIFO avant d'être transmis à la prochaine routeur (Figure 1.7). L'inconvénient principale de cette commutation est la nécessité d'utiliser des tampons de grande taille, ce qui engendre en conséquence une grande surface de silicium en plus de la latence engendrée par l'attente de la réception de la totalité du paquet(Palesi & Daneshtalab, 2014).

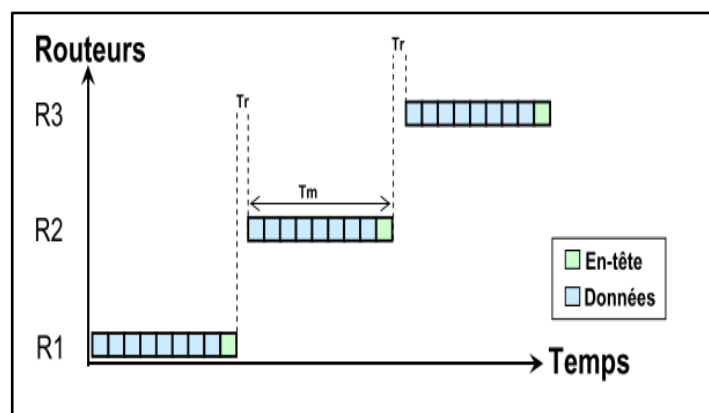


Figure 2-7: Store and Forward.(Delorme, 2007)

- Virtual Cut Through : Ce mode de commutation réduit la latence des modes (SAF) ainsi que les risques de blocage. Un routeur peut envoyer un paquet au routeur suivant avant qu'il ne le possède entièrement (figure 1.8), à condition que ce dernier garantisse qu'il dispose d'un espace mémoire suffisant pour stocker l'intégralité du paquet. Dans le pire des cas, où l'on retombe en mode SAF s'il y a contention. L'inconvénient de ce mode est que celui-ci requiert le même espace de stockage que le SAF(TOUATI, 2012).

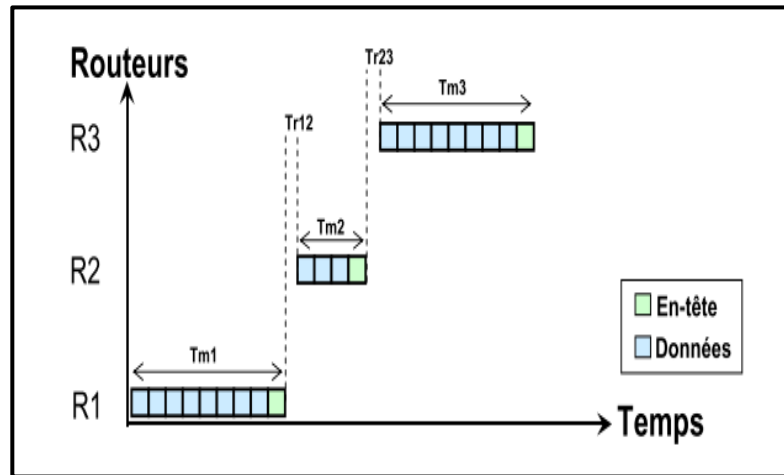


Figure 2-8:Téchnique VCT.(Delorme, 2007)

- Wormhole: Dans cette stratégie, le paquet est découpé en plusieurs flits et envoyé un par un au routeur suivant,(figure 1.9) sans que ce dernier ne puisse mémoriser sa totalité, contrairement à la stratégie SAF (BELKACEMI, 2020) . Par conséquent, l'espace tampon requis par un routeur donné et son délai de transmission sont réduits à la fois. En revanche, si le premier flit (appelé flit d'en-tête) est bloqué dans le routeur, les flits qui le suivent (appelés flit de données) appartenant au même paquet bloquent à leur tour les liens intermédiaires. Cette condition est appelée blocage en tête de ligne (HOL) et constitue le principal inconvénient de ce type d'échange(BELKACEMI, 2020).

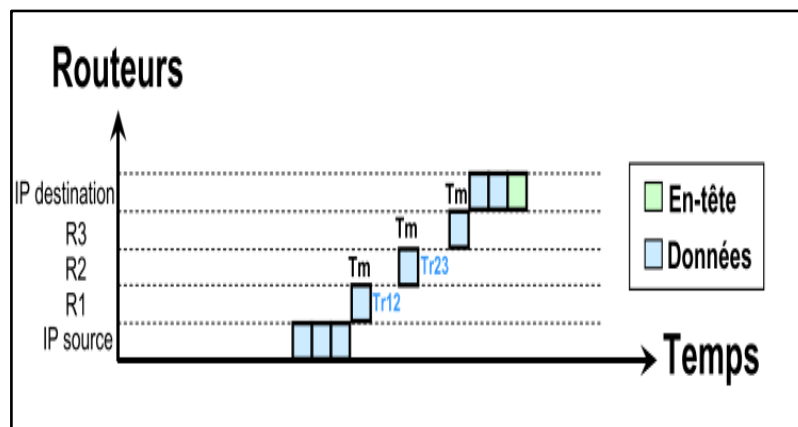


Figure 2-9:technique Wormhole.(Delorme, 2007)

1.7 Topologies et ses Classification

La topologie d'un réseau sur puce définit comment les différents composants sont interconnectés, Elle est souvent modélisée sous la forme d'un graphe $G(N,C)$ dans lequel N représente l'ensemble des routeurs et C l'ensembles des liens de

communication sont respectivement associés aux sommets et aux arrêtes de graphe(Cota et al., 2012).

Une topologie peut classifier selon plusieurs critères :

1.7.1 Régulière & Irrégulière

Les topologies Noc peut être classifier au premier lieu selon leur régularité. D'où un Noc est caractérisé soit comme régulier, soit comme irrégulier (Fehmi & Chatmen, 2016). Ces deux topologies présentent des avantages peut être exploitable par le domaine d'application cible. Une *topologie régulière* (figure 1.10) repose sur une distribution homogène des routeurs, elle convient principalement aux architectures à base de maillage et de tore. De plus, les topologies régulières sont réutilisables et imposent un minimal effort de reconception, dans le cas où elles seraient utilisées pour différentes applications/architectures(Tatas et al., 2014). Malgré ces avantages, l'utilisation de topologies régulières n'est pas fortement acceptable pour les produits commerciaux car elles imposent un certain nombre de lacunes. L'utilisation non optimale du réseau d'interconnexion c'est la raison principale de ces limitations, ce qui conduit à une augmentation des délais et de consommation d'énergie.

Une *topologie irrégulière* (figure 1.11), ou personnalisées a été également proposée afin de pallier ces inconvénients. Étant donné que ces topologies sont principalement orientées application, elles sont utilisées généralement aux Soc constitués de cœurs hétérogènes. De plus, la conception de cette topologie exige des algorithmes de routage plus avancés qui compte tenu de la non-uniformité du réseau sous-jacent. Par exemple, supposons une architecture Noc où chacun des routeurs doit être relié à un nombre différent de nœuds. Dans le cas où une topologie régulière est utilisée, certains des routeurs resteront non pleinement utilisés, ce qui entraînera à son tour des retards, une puissance et des surfaces importants(Tatas et al., 2014). D'autre part, il est possible de concevoir une architecture Noc hétérogène, où chaque routeur possède un nombre de ports différent. Une telle personnalisation des composants matériels se traduit par des performances supérieures, par rapport au cas où une topologie régulière est employée(KAMECHE, 2013). Cependant, l'hétérogénéité introduite impose un effort de conception supplémentaire à la fois pour les routeurs, ainsi que pour l'ensemble du Noc(Tatas et al., 2014).

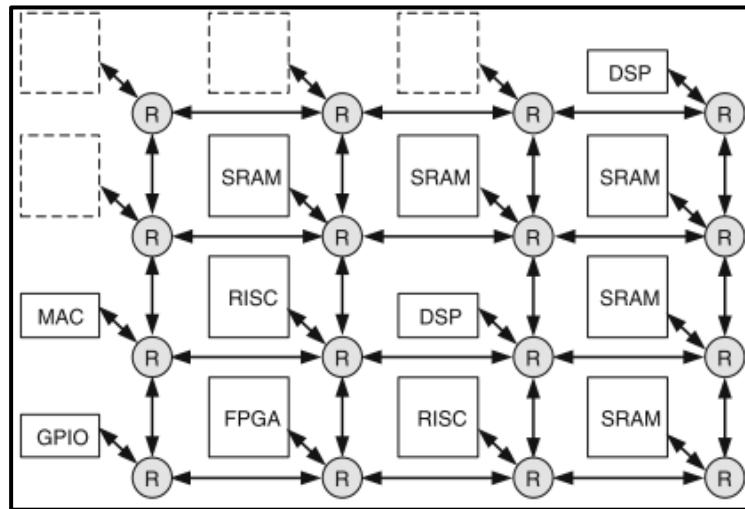


Figure 2-10: Topologie Régulière. (Tatas et al., 2014)

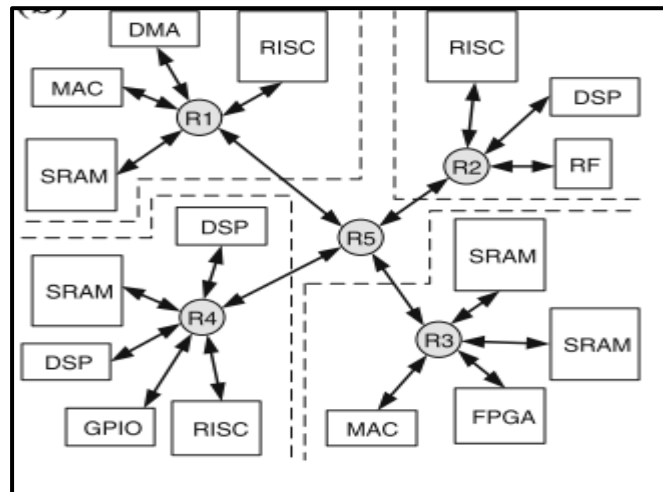


Figure 2-11: Topologie Irrégulière. (Tatas et al., 2014)

1.7.2 Directe & indirecte

Une topologie peut être également classé selon la manière dont les nœud sont connecte, trois grandes classes sont distingué par (Cota et al., 2012):

- Directe : chaque routeur est connecté à au moins un élément communicant (IP).
- Indirecte : les routeurs peuvent être connectés seulement à d'autres routeurs.
- Hybride : l'interconnexion des routeurs est une combinaison des réseaux directes et indirecte.

Topologie directe : chaque nœud est directement connecté à un nombre fixe de nœuds voisins et un message entre deux nœuds passe par un ou plusieurs nœuds intermédiaires (Rahman,2009). Seuls les routeurs sont impliqués dans la

communication dans une topologie directe et la communication est basée sur l'algorithme de routage mis en œuvre par les routeurs. La plupart des implémentations Noc sont basées sur des dispositions orthogonales des routeurs dans une topologie directe. Dans cette disposition, les nœuds sont répartis dans un espace à n dimensions et le paquet se déplace dans une dimension à la fois. Ces dispositions sont celles qui présentent le meilleur compromis entre le coût et les performances, et présentent également une bonne évolutivité(Cota et al., 2012).

Les topologies directes les plus courantes sont MeSH 2D (maillage à deux dimensions), Tore, Anneau :

Topologie de maillage à deux dimensions (MeSH 2D) : représenté sur la Figure 1.12, est la topologie la plus simple et la plus utilisée pour les Noc. Il se constitue d'un maillage $M \times N$ de commutateurs interconnectant des nœuds, ces nœuds peuvent être des cœurs de traitement, des mémoires, etc. Chaque commutateur (ou routeur), à l'exception de ceux situés sur les côtés, est connecté à quatre commutateurs voisins et à un nœud(Rahman,2009).. Donc le nombre de commutateurs est égal au nombre de nœuds. Des canaux de communication sont utilisés afin de réaliser le chemin de communication physique entre nœuds et commutateurs (ou entre commutateurs), dont chacun se compose de deux liaisons unidirectionnelles entre soit un commutateur et un nœud, soit deux commutateurs(Cota et al., 2012). La topologie maillée 2D est simple de mise en œuvre et facilement implantable sur une technologie silicium car tous les liens ont la même longueur et impose donc une régularité inhérente qui simplifie la conception physique. Au plus, elle est plus facile à la prédation des exigences de surface pour les topologies maillées, car elle permet d'augmenter d'une façon presque linéaire avec l'augmentation du nombre de nœuds. Mais malgré ces avantages, l'utilisation de la topologie maillée présente également certains inconvénients. La pléthore de routeurs incorporés dans cette topologie conduit généralement à des régions encombrées sur le périphérique(Tatas et al., 2014).

Topologie Tore : C'est une topologie dérivée d'une structure 2D, avec la propriété de replier le bord extérieur sur lui-même (Figure 1.13). En tant que tel, il offre une bande passante légèrement supérieure à un maillage 2D(Delorme, 2007).

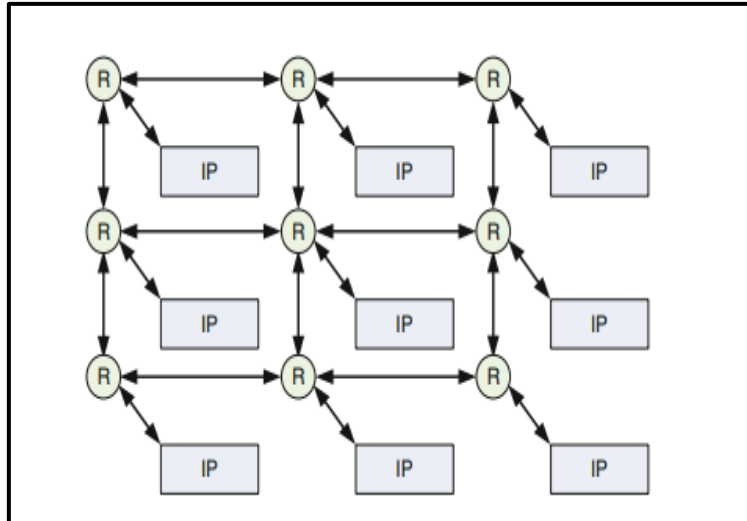


Figure 2-12 :Topologie 2D MeSH.(Tatas et al., 2014)

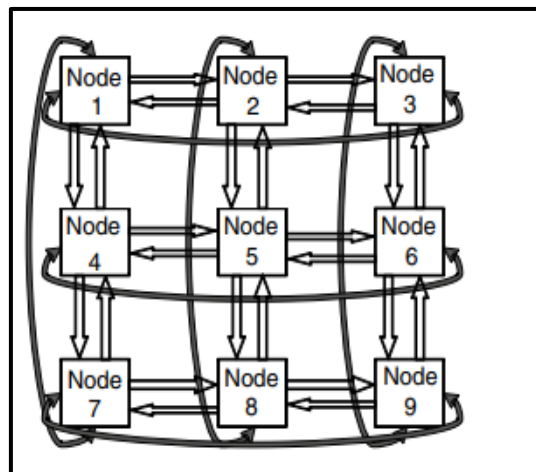


Figure 2-13: Topologie Tore.(Cota et al., 2012)

Topologie en anneau : C'est est une configuration simple, telle qu'elle est illustrée à la Figure 1.14, consiste en un anneau de huit nœuds, liés entre eux par 12 liaisons bidirectionnelles(Djaballah, 2011). Ces liaisons doté une communication à deux sauts entre n'importe quelle paire de nœuds dans l'anneau, lequel permet d'utiliser un algorithme simple pour effectuer un routage rapide mais efficace par le plus court chemin. Cette topologie suppose que chaque nœud est associé à un nœud et à un commutateur, alors que pour accomplir la communication nécessaire entre deux couples des nœuds, aux plus deux sauts sont nécessaires à l'intérieur de l'unité octogonale de base(Tatas et al., 2014).

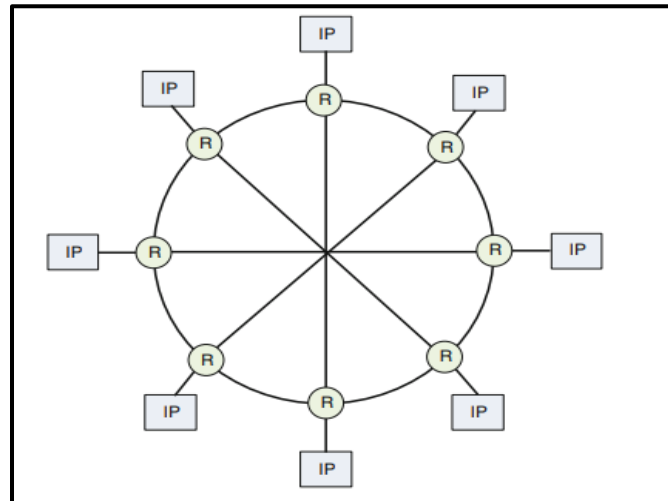


Figure 2-14: Topologie Anneau. (Tatas et al., 2014)

Dans la **topologie indirecte**, tous les routeurs ne sont pas connectés à des unités de traitement comme dans le modèle direct. Au lieu de cela, certains routeurs ne sont utilisés que pour propager les messages à travers le réseau, tandis que d'autres routeurs sont connectés à la logique et seuls ceux-ci peuvent être la source et/ou la cible d'un message. Les plus connus Papillon (Butterfly flat-tree) et Arbre élargi (Cota et al., 2012).

Arbre élargi : est un réseau indirect (le nombre de nœuds est indépendant du nombre de cœurs IP) (Figure 1.15), où les nœuds représentent les routeurs et les feuilles sont des cœurs connectés au réseau. Un routeur au-dessus d'une feuille est appelé son ancêtre, et les feuilles en dessous de son ancêtre sont appelées ses descendants, respectivement. Dans cette topologie, chaque nœud a plusieurs ancêtres, ce qui signifie qu'il existe de nombreux chemins alternatifs entre les nœuds (Fehmi & Chatmen, 2016).

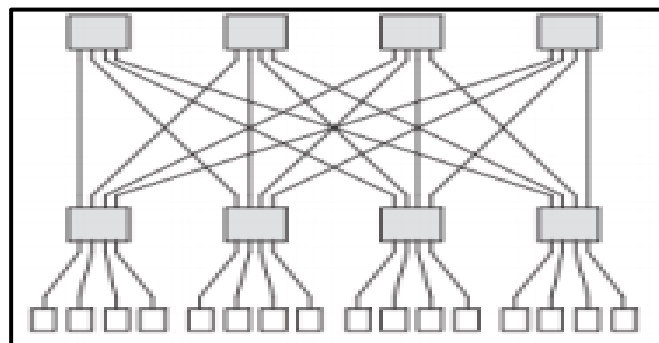


Figure 2-15: Arbre élargi. (TOUATI, 2012)

La topologie papillon (Butterfly flat-tree) : cette topologie conduit à l'augmentation de la complexité d'interconnexion et le coût en surface en raison de sa hauteur d'arbre plus élevée (figure 1.16). Dans cette topologie chaque nœud dispose de deux ancêtres et de quatre nœuds descendants (Fehmi & Chatmen, 2016). Pour pointer les nœuds, des coordonnées (L, P) sont données à chaque nœud où L indique le niveau du nœud, et P indique sa position à l'intérieur de ce niveau. L'adresse de niveau le plus bas est zéro et les adresses des nœuds sont définies de 0 à (N-1). Il y a $N / 4$ nœuds au premier niveau et $N / 2^j + 1$ nœuds pour le j-ème niveau. Le nombre de commutateurs à chaque niveau est divisé par 2 (Fehmi & Chatmen, 2016).

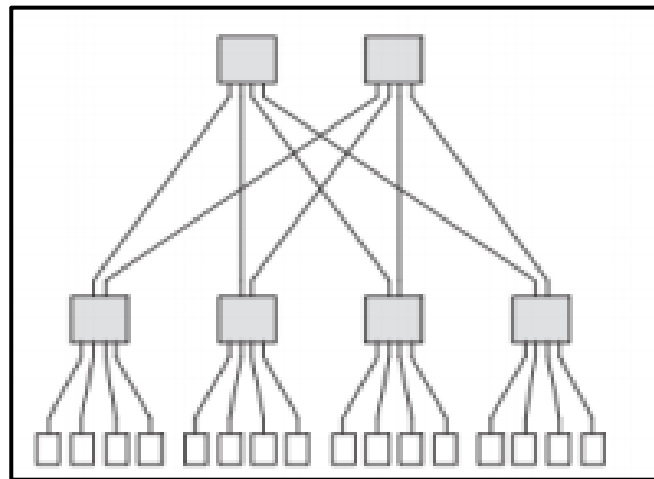


Figure 2-16: Arbre élargi en papillon. (TOUATI, 2012)

1.8 Topologie émergente de Noc

La topologie maillée 2D est la plus utilisée dans les Noc en raison de sa simplicité, mais cette topologie ne peut pas répondre aux exigences d'un système sur puce trop complexe. L'intégration du wifi dans le Noc, ou l'architecture 3D, est une solution efficace au problème complexe de l'interconnexion des Soc à grande échelle.

1.8.1 Réseau sur puce wifi (W-Noc)

Un réseau sur puce wifi (W-noc) est un nouveau concept basé sur les réseaux sur puce simple. Elle utilise deux types d'interconnexion de communication entre les cœurs : une liaison filaire et autre sans fil. Le W-Noc se compose des mêmes composants que le Noc simple, la différence est en routeurs, Deux types de routeurs sont existants : Le routeur basique a la même microarchitecture que le routeur

conventionnel, et le routeur hybride qui est une combinaison de routeur basique et unité de communication sans fil (Wireless communication unit)(Yin et al., 2012) .

1.8.2 Topologie 3D

Le réseau sur puce en 3D est une amélioration de la topologie 2D. L'intégration 3D fournit des communications verticales à travers des liens verticaux courts (TSV), ainsi que des communications horizontales. L'avantage principal de cette intégration est la réduction considérable de la longueur des liens d'interconnexions et donc, elle diminue le chemin parcouru pour le transfert de données(Toubaline, 2018).

1.8.2.1 Les liens TSV

Au début des années 1950(Al Attar, 2012), une proposition a été appliquée aux diapositives électroniques pour permettre la communication verticale. La première tentative consistait à empiler une puce sur une autre et à utiliser un câble à l'extérieur de la puce (Figure 1.17). Cependant, la grande surface occupée par ces liens a obligé les chercheurs à envisager un autre type d'interconnexion connu sous le nom de vias traversants en silicium, abrégé en TSV(Al Attar, 2012).

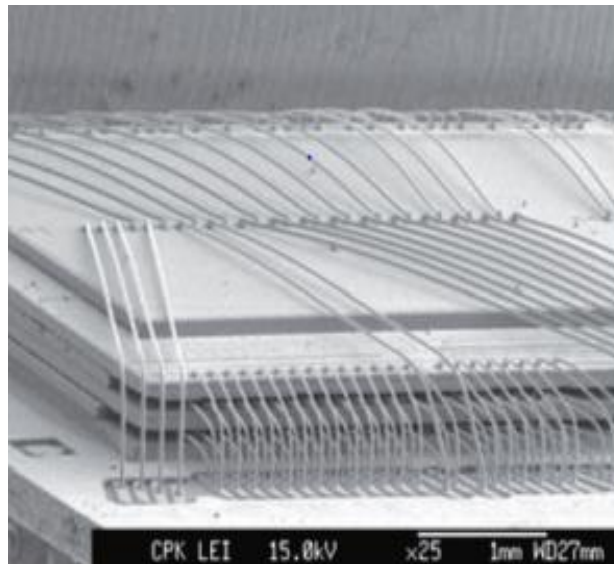


Figure 2-17:Empilement de plusieurs puces Par wire bonding.(Sheibanyrd & Pétrot, 2011)

La figure 1.18 illustre une topologie 3D avec des liens TSV :

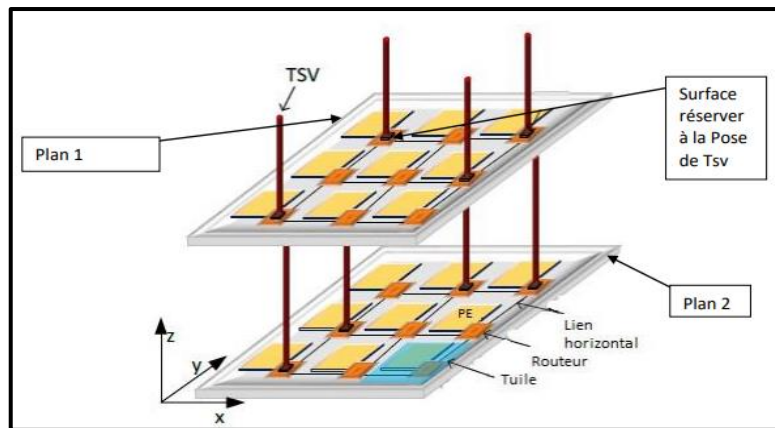


Figure 2-18: Topologie 3D avec lien TSV.(Michael et al, 2018)

1.9 Algorithmes de Routage

Après avoir déterminé la topologie du réseau, que nous avons préalablement définies comme une organisation physique d'un réseau de nœuds, nous devons choisir un algorithme de routage. Les algorithmes de routage sont chargés de décider du chemin qu'un message doit emprunter afin d'être acheminé efficacement de sa source à sa destination. Le choix de l'algorithme de routage devient très important pour améliorer les performances du réseau, son but est de répartir le trafic de la même manière entre les chemins fournis par la topologie du réseau, améliorant ainsi le délai et le débit du réseau(Bouguettaya, 2017).

Un routage peut être déterministe ou adaptatif :

Routage déterministe

Pour le routage déterministe, le chemin emprunté par un paquet entre la source et la destination est unique et prédéfini ,un exemple d'algorithme de routage déterministe est l'algorithme XY appliqué dans un réseau à topologie maillée 2D , dans ce cas où le paquet se déplace dans la direction x (axe des abscisses) puis dans la direction y vers la destination, un seul tour du paquet est autorisé le long du chemin de routage(Agrawal et al., 2009)(Saravankumar et al., 2013).

Routage adaptative

Pour le routage adaptatif, plusieurs chemins peuvent être envisagés lors de la transmission du paquet, et le chemin choisi est celui qui montre le moins d'encombrement possible, contrairement aux algorithmes déterministes, dans ce cas, le paquet peut réaliser plusieurs tours avant d'atteindre sa destination (figure 1.19), certains algorithmes adaptatifs comme (west first turns, north last turns, negative first turns,, Odd-even turn model OE) ne permettent pas certains tours afin d'éviter les deadlock (Pétrot, 2009).

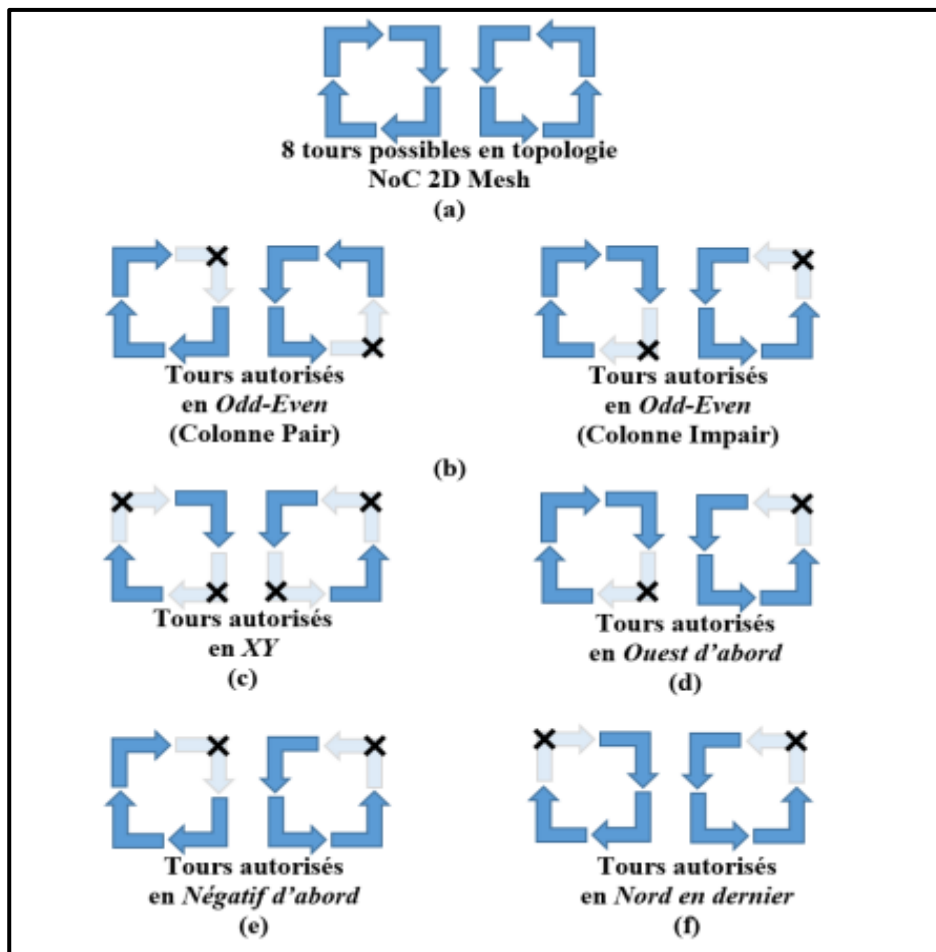


Figure 1-19:les tours associés à chaque algorithme.(Chariet, 2014)

1.10 Les paramètres de performance

Les performances d'un réseau sur puce sont évaluées par quelques critères, nous citons : la bande passante, le débit et la latence :

La bande passante : définit au taux maximal de propagation des données une fois qu'un message est dans le réseau. L'unité de mesure de la bande passante est le bit

par seconde (bps) et prend généralement en compte l'ensemble du paquet, y compris les bits de l'en-tête, de la charge utile et de la queue(Cota et al., 2012).

Le débit : est défini comme le trafic maximal accepté par le réseau, c'est-à-dire la quantité maximale d'informations délivrées par unité de temps. La mesure du débit correspond aux messages par seconde ou aux messages par cycle d'horloge(Cota et al., 2012). On peut avoir un débit normalisé (indépendamment de la taille des messages et du réseau) en le divisant par la taille des messages et par la taille du réseau. Par conséquent, l'unité du débit normalisé est le bit par nœud par cycle d'horloge (ou par seconde)(Cota et al., 2012).

La latence : est le temps total entre le début de la transmission d'un message (ou d'un paquet) et sa réception complète au nœud cible. La latence est mesurée en unités de temps et principalement utilisée comme base de comparaison entre différents choix de conception. Dans ce cas, la latence peut également être exprimée en termes de cycles d'horloge du simulateur. Normalement, la latence d'un seul paquet n'est pas significative et on utilise la latence moyenne pour évaluer les performances du réseau. En revanche, lorsque certains messages présentent une latence bien supérieure à la moyenne, cela peut être important. Par conséquent, l'écart type peut également être une mesure intéressante.(Cota et al., 2012)

1.11 Conclusion

Dans ce chapitre, nous avons présenté le concept de réseaux sur puce et ses caractéristiques telles que l'architecture et les composants principaux, les différentes topologies, les modes de commutation et les algorithmes de routage, parmi lesquels nous avons montré comment choisir le meilleur chemin pour envoyer des données du nœud émetteur au nœud récepteur, à la fin, nous avons présenté certains des simulateurs les plus répondus, à partir desquels nous pouvons mieux comprendre comment fonctionne cette notion.

En effet, le réseau sur puce est un paradigme inspiré des réseaux informatiques. Il consiste à intégrer l'ensemble d'un réseau de communication dans un système sur puce afin d'interconnecter ses différents composants. Afin d'améliorer leur performance et réduire la consommation d'énergie et latence, Les concepteurs et les chercheurs proposent des méthodes de mapping des tâches des applications sur les tuiles des Noc.

Dans le chapitre suivant, nous allons présenter quelque des ces méthodes et explique c'est quoi un problèmes de mapping.



Chapitre 2 : Travaux Connexes

2.1 Introduction

Le réseau sur puce (Noc) est un paradigme de communication sur puce qui évolue vers une meilleure alternative à l'intégration d'un grand nombre de cœurs sur un seul système sur puce (Soc). Le flux de conception Noc contient de nombreux problèmes de différents domaines, tels que la mise en réseau, la conception embarquée et l'architecture informatique. Le mapping des applications est l'un de ces problèmes, qui est bien étudié dans la littérature, mais est généralement considéré comme un problème de minimisation de l'énergie de communication, et il existe plusieurs approches pour résoudre ce type de problème.

Dans ce chapitre, nous allons présenter quelques travaux connexes, dans chacun desquels nous allons mettre l'accent sur les architectures de réseau sur puce, la méthode choisie pour résoudre le problème et dans quel but ils ont choisi cette approche, mais d'abord, nous allons définir ce qu'est un problème de mapping et ses types.

2.2 Les phases de conception d'un Noc

Concevoir un Noc efficace qui répond aux besoins des utilisateurs est à la fois difficile et complexe. Afin de concevoir un Noc, il faut passer par plusieurs couches allant de la couche de modélisation à la couche d'implémentation physique.

Les phases de conception d'un Noc peuvent être résumées comme suit (Pop & Kumar, 2004) :

- ❖ **Modélisation du Traffic (description des applications)** : C'est la première étape dans la description générique d'un NOC.
- ❖ **Détermination de la topologie Noc** : Il s'agit d'une architecture générique spécifiquement destinée aux applications.
- ❖ **Assignation des tâches aux nœuds de l'application** : attribuer les tâches aux processeurs.
- ❖ **Ordonnement des tâches** : Déterminez l'ordre dans lequel les tâches sont exécutées.
- ❖ **Placement des tâches sur une topologie** : affecter les processeurs aux tuiles de l'architecture.

- ❖ **Allocation des chemins de routage et réservation des ressources** : Le choix de bon chemin entre la source et la destination, car cela a un impact important sur les performances du Noc.
- ❖ **Vérification des performances** : Validez les performances du Noc et comparez-les aux performances souhaitées par le concepteur.
- ❖ **Développement** : développer des modèles de synthèse et de simulation.

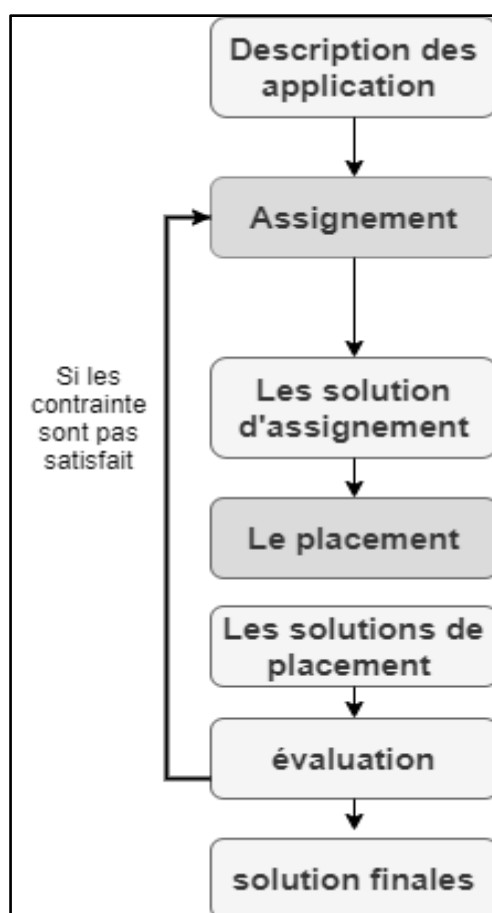


Figure 2-1: Organigramme des étapes de conception de Noc.(Lemaire, 2006)

Nous nous intéressons à ce mémoire par la phase de placement des tâches (Le mapping) que nous présenterons dans la section suivante.

2.3 Problème de Mapping

Le problème de mapping d'application est NP-difficile. Selon le moment où les tâches sont attribuées à IP pour le traitement. Les techniques de mapping peuvent être divisées en mapping dynamique et statique(Figure 2.2)(Sahu & Chattopadhyay, 2013).

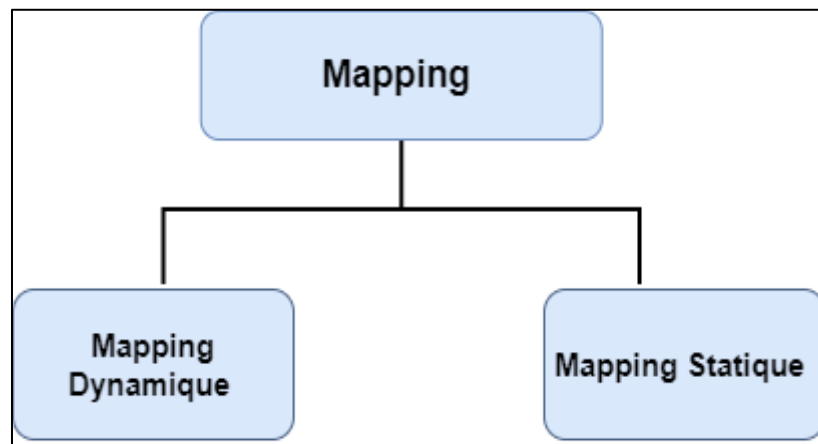


Figure 2-2:types mapping.

2.3.1 Mapping Dynamique

L'affectation et la planification des tâches sont réalisées lors de l'exécution de l'application. Le mapping dynamique essaie toujours de détecter les goulots d'étranglement des performances et de répartir la charge de travail entre les processeurs. Étant donné que le mapping dépend de la charge actuelle du processeur, cela devrait donner une meilleure solution. Cependant, la surcharge de calcul de l'algorithme de mapping augmente la latence d'exécution et la consommation d'énergie de l'application(Sahu & Chattopadhyay, 2013).

2.3.2 Mapping Statique

Généralement le mapping des tâches est effectué hors ligne, avant que l'application ne soit lancée. Pour une application donnée et une infrastructure de communication cible, le mapping statique essaie toujours de définir le meilleur placement des tâches au moment de la conception. Comme le mapping est terminé avant l'exécution, l'algorithme de mapping n'est exécuté qu'une seule fois. Pour Noc, le mapping statique est généralement recommandé, car une surcharge de communication excessive dans le mapping dynamique affecte considérablement les performances du système, augmentant le retard global du système(Sahu & Chattopadhyay, 2013).

2.3.3 Les déficit de Mapping

La complexité des fonctions intégrées dans le Soc impose diverses contraintes à la mise en œuvre de ce dernier. Ces contraintes peuvent apparaître à différents niveaux(KAMECHE, 2013) :

- **Au niveau applicatif** : le principal défi consiste à exploiter le parallélisme des tâches et à prendre en compte la communication concurrente dans le modèle de calcul.
- **Au niveau architecture** : plusieurs topologies sont explorées afin de choisir celle qui répond le mieux aux exigences et aux besoins de l'application ciblée. Ceci peut être complexe et très long à effectuer.
- **Au niveau communications** : l'application doit être placée sur les composants de l'architecture de telle sorte que les coûts soient minimisés. Cette procédure est délicate car un mauvais placement peut engendrer une violation des contraintes imposées.

2.4 Travaux liés

Dans cette section nous allons présenter quelques travaux connexes pour mieux comprendre le problème de mapping et voire les différentes méthodes existant dans la littérature.

❖ Article 1

D'après (Sahu et al., 2015), Des stratégies ont été proposées pour le mapping sur les Noc basé sur la topologie Butterfly-Fat-Tree (BFT) (figure 2.2) en utilisant une technique métaheuristique qui s'appelle *partical swarm optimisation* (PSO) unique et PSO discrète augmentée multiple. Cette approche se base sur ces principales caractéristiques :

1. Les PSO ont été présentée pour le mapping des applications sur le BFT afin de minimiser le coût global de communication.
2. Ils ont également utilisé une PSO en plusieurs étapes. Les meilleures informations locales et globales de la i -ème étape ont été transmises aux particules dans $(i + 1)$ étape. Ceci assure une convergence plus rapide et une meilleure qualité de solution pour les étapes successives.
3. Quelle que soit l'étape de l'OSP, la génération initiale de la population n'est pas totalement aléatoire. Un bon nombre de particules ont été créées à l'aide d'une heuristique. Cela permis à cette technique d'explorer beaucoup mieux les régions prometteuses de l'espace de recherche.

4. Les valeurs métriques de coût de communication des solutions de mapping de cette approche ont été comparées à l'approche existante pour le mapping BFT. Il montre une bonne amélioration de la qualité de la solution.
5. Comparaison des performances dynamiques (en termes de latence du réseau) et la consommation d'énergie ont également été effectués.

Il peut être noté à partir des résultats obtenus que cette stratégie de PSO augmentée à plusieurs étapes montre de meilleurs résultats que la stratégie de mapping PSO à une étape et KL_BFT (figure 2.3). Pour les Noc ayant un plus grand nombre de cœurs, cette stratégie PSO augmentée en plusieurs étapes produit les meilleures solutions. La comparaison des solutions produites établit les techniques de cartographie sur le Noc basé sur BFT pour être un concurrent sérieux des stratégies de cartographie précédemment disponibles.

❖ Article 2

Dans (Çeløkk, 2012), le problème de mapping d'application de flux de conception Noc est abordé d'un point de vue réseau, et l'impact du mapping sur les performances du réseau, c'est-à-dire le taux de perte généré par la mémoire tampon, est analysé. En supposant un modèle de trafic autosimilaire pour le réseau sur puce, des expressions pour le taux de perte de paquets et le nombre moyen de paquets dans la mémoire tampon sont obtenues.

Ils ont constaté que les cœurs de mapping qui ne tiennent compte que de la consommation d'énergie ont un impact dégradant significatif sur les performances du réseau en termes de taux de perte pour les systèmes de réseau sur puce. Les résultats montrent que les solutions de mapping avec une consommation d'énergie de communication acceptable peuvent varier considérablement lorsque les taux de perte observés sur les tampons sont considérés sous l'hypothèse de flux autosimilaire. Même si la consommation d'énergie est optimale, si la solution de mapping exploite un ou plusieurs tampons dans un état congestionné, cela dégradera les performances d'exécution du réseau sur puce. Par conséquent, après avoir analysé cet impact, ils ont déterminé la nécessité de résoudre le problème de

mapping d'application d'une manière qui évite également la congestion de la mémoire tampon tout en minimisant l'énergie de communication.

❖ Article 3

Dans (Murali & Micheli, 2004), les auteurs ont présenté NMAP qui est un algorithme rapide pour le mapping des cœurs sur une architecture Noc maillée sous des contraintes de bande passante, minimisant le délai de communication moyen. Ils ont conçu des systèmes monolithiques complexes dans lesquels les cœurs de traitement génèrent et consomment des quantités variables de données, mettant les liens de communication au bord de la congestion. Les applications typiques sont dans le domaine du traitement multimédia. Une architecture de réseau sur puce (Noc) basée sur le maillage est proposée, dans laquelle ils explorent l'allocation de cœur aux intersections de maillage afin que le trafic sur les liaisons respecte les contraintes de bande passante. Le routage déterministe à chemin unique entre les cœurs impose des exigences élevées en matière de bande passante sur la liaison. En répartissant le trafic entre les cœurs sur plusieurs chemins, les besoins en bande passante peuvent être considérablement réduits. L'algorithme NMAP fait le routage à chemin minimum unique et pour le routage à trafic partagé. L'algorithme est appliqué à une conception DSP de référence et le Noc résultant est construit et simulé au niveau précis du cycle dans System à l'aide de macros de la bibliothèque `xpipes`. En outre, des expériences avec six applications de traitement vidéo montrent des économies significatives en termes de bande passante et de coût de communication pour l'algorithme NMAP par rapport aux algorithmes existants.

❖ Article 4

Cet article (Wang et al., 2016) étudie le problème de mapping IP sous les contraintes de bande passante et de latence qui mappe un ensemble donné de cœurs IP sur les tuiles d'une architecture de réseau sur puce (Noc) basée sur le maillage afin de minimiser la consommation d'énergie de communications.

Un algorithme a été proposé pour solver ce problème, C'est un algorithme efficace basé sur un modèle (TEM) qui génère des résultats de mapping de haute qualité avec un temps d'exécution réduit. Il est conçu sur la base de deux modèles : les graphiques avec des sommets étroitement couplés et ceux avec des sommets distribués. Dans le 1^{er} cas, TEM mappe d'abord les nœuds chauds avec leurs quatre

voisins les plus significatifs, après quoi les cœurs IP restants sont mappés par ordre décroissant en fonction des poids des bords. Dans le 2^{-ème} cas, les sommets distribués sont mappés selon un schéma de partition de graphe, d'où TEM divise le Noc en régions et le CTG en blocs ; puis il mappe les cœurs IP à l'intérieur de chaque bloc d'une manière diviser pour mieux régner.

Les tests sur les benchmarks multimédias et aléatoires montrent que TEM génère des résultats de mapping de haute qualité avec de faibles temps d'exécution et permet d'économiser en moyenne 15 % d'énergie par rapport à MOCA.

❖ Article 5

Les architectures Noc sont considérées comme les systèmes d'interconnexion de nouvelle génération pour les systèmes multiprocesseurs sur puce. La sélection de l'architecture du réseau et la cartographie des nœuds IP sur la topologie Noc sont deux sujets de recherche importants.

Dans cet article (Bononi et al., 2017), Ils ont comparé quatre architectures Noc importantes : Ring et MeSH, le Spidergon STNoC et le Cross-bar sans tampon en utilisant des modèles de trafic Mpeg4 théoriques et réalistes. Le mapping PE/SE a été calculé par une version personnalisée de l'outil de partitionnement SCOTCH, tandis que le mapping Cross-bar est trivialement optimal. Ainsi que le débit maximal de livraison de réponse a été mesuré sous différents modèles de trafic homogènes et différentes tailles de réseau. Les résultats indiquent que dans de nombreux scénarios de trafic uniformes, le Noc Cross-bar surpasse les autres architectures en raison de sa cartographie optimale et de ses propriétés de distance moyenne optimales. Spidergon STNoC montre une meilleure évolutivité (exigences de tampon réduites) que MeSH tout en maintenant des performances en ligne avec le MeSH lui-même. Dans un scénario de trafic Mpeg4 réel, la barre transversale montre un comportement très évident sous un trafic complexe et non uniforme. Le manque de mémoire tampon le rend moins efficace en termes de débit. Bien qu'il existe des différences significatives dans la mise en mémoire tampon du Noc à 12 nœuds, les topologies Spidergon et Ring fonctionnent de manière similaire. De plus, Spidergon et Ring surpassent légèrement les architectures MeSh et Cross-bar, avec un bon équilibre du trafic des canaux. Les performances du maillage sont limitées par la forme rectangulaire et les propriétés de l'algorithme de

routage : la sélection de chemin plus large accordée par la topologie du maillage ne constitue pas un avantage si l'algorithme de routage ne peut pas en tirer profit. Les petites différences de performances mesurées dans les scénarios MPEG suggèrent que dans ce cas, le choix du Noc à utiliser doit être principalement basé sur des facteurs tels que la surface et les exigences de tampon. Ils recommandent une analyse approfondie dans les travaux futurs de la pénalité de performance introduite par les répéteurs qui peuvent être insérés sur des liens trop longs pour respecter la synchronicité du système, tout en considérant des graphes de tâches ou plus pertinents pour les applications de streaming réalistes ou les modèles de trafic plus théoriques tels que des arbres et des forêts à routage unique/multiple.

Mémoire 1

Cette mémoire intitulée sur « Mapping dans les réseaux sur puce 3D » (MESSAOUADI, 2019) se concentre sur le développement d'une méthode pour mapper les IP d'une application sur un réseau sur puce afin de minimiser les coûts de communication. Ils sont basés sur la topologie 3D la plus largement utilisée. Permet de répondre aux exigences et aux faiblesses des architectes 2D, notamment pour les applications avec un très nombre des tâches et de grande taille. La technique présentée dans ce travail est une méta-heuristique, appelée BFO, qui appartient à l'une des approches de résolution de problèmes d'optimisation, qui sont basées sur l'optimisation de butinage dans le monde bactérien.

Tableau 1: Comparaison entre les articles

Article	Architecture Noc	Problème	Objective	Méthode
(Murali & Micheli, 2004)	2D mesh	Mapping sur les cœurs noc	Minimiser latence et bande passante	Algorithme NMPP
(Çeløk, 2012)	/	Augmenter la performance de noc	Minimiser l'énergie	Conception de memoire tampon
(Sahu et al, 2015)	BTF	Mapping sur les cœurs noc	Minimiser énergie	PSO
(Wang et al., 2016)	2D mesh	Mapping sous la contrainte de latence et bande passante	Minimiser energy de communication	TEM
(Bononi et al., 2017)	Ring/mesh/corsbar /Spidergon	Comparaison entre des architectures noc	Trouve meilleure performance	/
(MESSAOUADI, 2019)	3D »	Mapping des applications sur la topologie 3D	Minimiser le coût de communication	BFO.

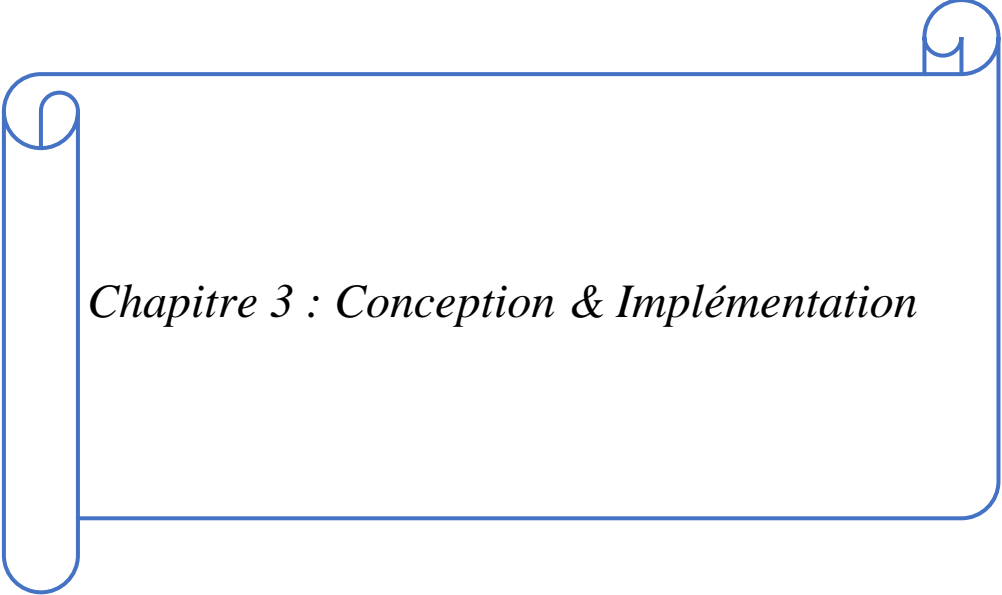
2.5 Conclusion

Dans ce chapitre, nous avons présenté le concept de mapping et leurs types, puis nous avons étudié les travaux connexes qui traitent les méthodes de mapping sur les Noc.

En effet, les Nocs sont les cadres de communication sur puce pour les systèmes multi/multicœurs car ils surpassent les bus traditionnels en termes d'évolutivité, de parallélisme et d'efficacité énergétique. Le mapping des IPs des applications sur

leur tuile est l'étape la plus importante de la construction et de la conception de différents architectes.

Dans le chapitre suivant, Nous allons présenter notre solution pour résoudre le problème de mapping afin d'améliorer les performances de noc.



Chapitre 3 : Conception & Implémentation

3.1 Introduction

L'objectif principal de notre travail est le mapping sur les réseaux sur puce, Ce problème peut être classifié parmi les problèmes d'optimisation, car il consiste à minimiser le coût de communication, Nous avons proposé de résoudre ce problème avec l'un des algorithmes méta-heuristique, l'algorithme de chauves-souris (Bat algorithme).

Ce chapitre présente la partie de conception de notre mémoire, nous allons donc d'abord définir les problèmes d'optimisation et les méta- heuristique, ensuite nous allons présenter l'algorithme proposé et son adaptation sur notre problème, ainsi la modélisation de l'algorithme, et à la fin nous concluons le chapitre par une conclusion.

3.2 Contexte de projet

Les problèmes d'optimisation parmi les concepts les plus reconnus au domaine informatique, ils servent à minimiser ou maximiser une fonction qui s'appelle la fonction d'objectif, Mais certains de ces derniers sont impossibles à résoudre d'une façon exacte au temps raisonnable, Pour cette raison, les chercheurs utilisent souvent des méthodes approchées, qui sont les heuristiques ou les méta- heuristiques, Malgré cela ils ne donnent pas des résultats exacts et optimaux, mais ils nous aident à trouver des solutions approximatives.

3.2.1 Problème d'optimisation

Un problème d'optimisation se définit comme la recherche du minimum ou du maximum (donc de l'optimum) d'une fonction donnée par rapport à tous les paramètres concernés. (Luis & Moncayo, 2002) Par exemple, dans notre problématique de mapping des composants logiciels ou des tâches d'une application, nous cherchons à minimiser le coût de communication en termes d'énergie et de latence.

La définition du problème d'optimisation est souvent complétée par la donnée des contraintes : tous les paramètres (ou variables de décision) de la solution proposée doivent respecter ces contraintes, sinon la solution n'est pas réalisable. (Luis & Moncayo, 2002) Dans notre cas, pour mapper une application qui est représentée par un graphe des tâches, il faut vérifier que la taille de ce graphe est inférieure ou égale à la taille de Noc.

Les méthodes d'optimisation se diversifient selon les nombres des contraintes et le problème en plusieurs classes, Nous citons dans la *figure 3.1* les plus importants et proches de notre domaine de recherche.

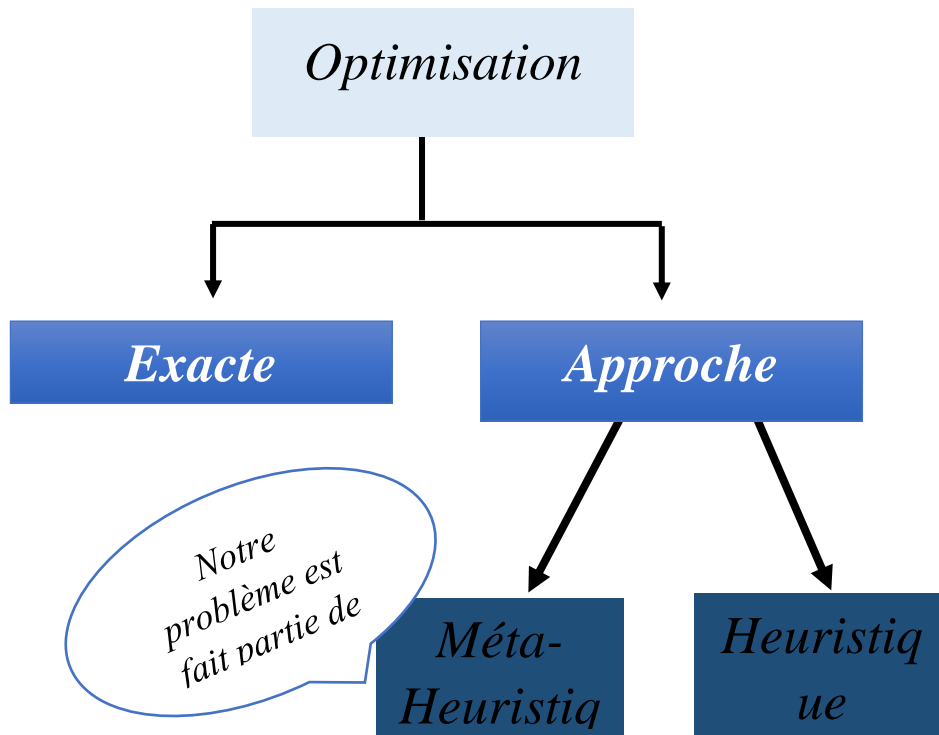


Figure 3-1: Classification des méthodes d'optimisation.

Ajouter des détails sur le graphe

Les méthodes exactes : L'intérêt de ces méthodes est qu'elles garantissent la meilleure solution au problème traité. Ces méthodes énumèrent toutes les solutions possibles et choisissent la meilleure d'entre elles. En effet, ils permettent de parcourir tout l'espace de recherche pour s'assurer d'obtenir toutes les solutions possibles meilleures que la solution optimale trouvée lors de la recherche. Cependant, si le problème est important, la méthode exacte prend trop de temps et coûte souvent trop cher en termes de ressources requises (MEHAMDIA & TIGRE, 2018).

Les méthodes approchées : Pour pallier les problèmes liés aux méthodes exactes, à savoir un temps et un coût excessifs, des méthodes approchées sont apparues. Ces méthodes sont plus

pratiques pour résoudre des problèmes difficiles, et à grande tailles, visant à trouver des solutions dans un délai raisonnable. Ils ont donné de bons résultats dans un temps acceptable. Ces méthodes se divisent en deux catégories (MEHAMDIA & TIGRE, 2018): Les heuristique et les méta-heuristique.

3.2.2 Méta-heuristique & Heuristique

Les heuristiques sont des solutions développées pour résoudre un problème spécifique. Ils sont performants et peuvent trouver efficacement des valeurs optimales dans un petit espace de recherche de manière précise, mais cela nécessite beaucoup d'expertise et d'expérience. Quant aux métaheuristiques, ce sont des méthodes générales qui peuvent être appliquées à un ensemble de problèmes. Ces méthodes sont souvent des algorithmes inespérés de la nature (physique, évolution biologique, éthologie...) et elles sont applicables à tout problème d'optimisation.(Karima, 2003) Dans ce projet, nous proposons d'utiliser une technique basée sur **l'algorithme de chauves-souris (Bat)**, inspiré du comportement des chauves-souris.

Le nombre d'objectifs d'optimisation permet de distinguer les problèmes mono-objectifs des problèmes à objectifs multiples. Résoudre un problème d'optimisation mono-objectif consiste à trouver une solution qui optimise une seule fonction objective. Cependant, les problèmes de type multi-objectifs cherchent à optimiser plusieurs objectifs simultanément (Karima, 2003). Notre recherche vise à minimiser l'énergie de communication, la latence et le coût de communication dans le cas des architectures homogène où tous les PE fonctionne de manière identique, Pour cela ces objective peuvent être résumé dans une fonction d'objectif unique qui minimise le nombre de sauts entre la source et la destination, où plus la distance est petite, plus le coût global de communication est faible. Par conséquent, nous pouvons conclure que notre étude s'inscrit dans le cadre de l'optimisation mono objectif.

3.3 Architecture de la solution proposée

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

1. Les Modules :

- ✓ **Module1** : les applications qui sont sous forme des graphes des tâches.
- ✓ **Module 2** : les architectures Noc qui constitue un ensemble des PE relie entre eux par des liens.
- ✓ **Module 3** : l'algorithme de chauves-souris et son adaptation sur notre problème.

2. Les fonctions :

- ✓ **F1 (Lire un fichier xml)** : une fonction permet de lire un fichier xml qui décrit une application.
- ✓ **F2 :(Choix une topologie) : une** fonction permet de choisir une topologie Noc parmi les topologies proposées.
- ✓ **F3(Sauvegarder le résultat final) : permet** de sauvegarder la solution finale obtenue par le bat.
- ✓ **F4(Ajouter le résultat à l'historique) : ajouter** la solution finale dans le fichier historique.

La **figure 3.2** illustre un schéma qui détaille notre architecture.

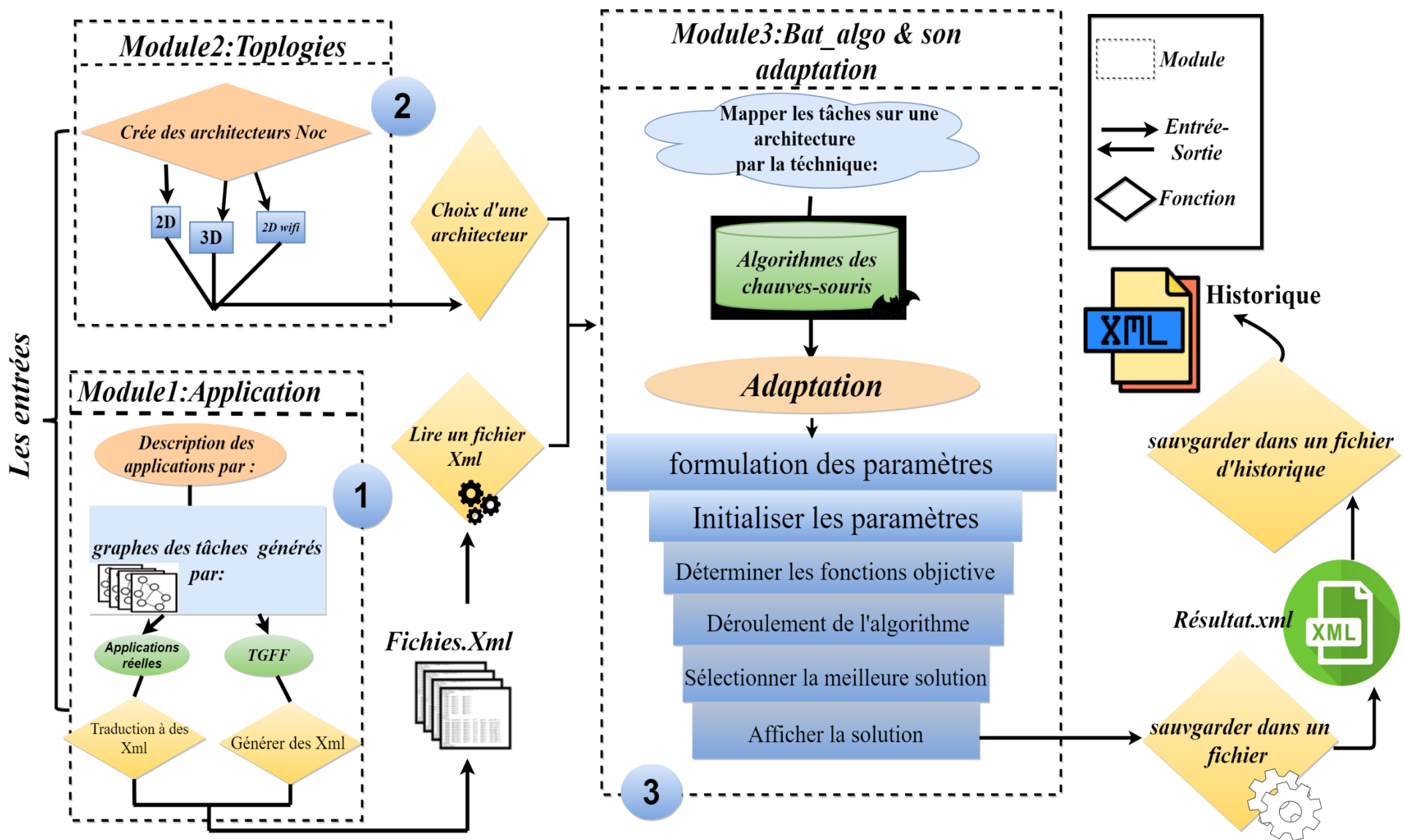


Figure 3-2: Schéma de travail.

3.3.1 Module 1 : Applications

Le **Module 1** sur le schéma précédant décrit les premiers entrés de notre système. Pratiquement une application ou un logiciel représenté par un graphe des tâches.

Un graphe des tâches ou graphe d'application $G (T, C)$ est un graphe orienté (*Figure 3.3*), dont les nœuds ($t_i \in T$) représentent les tâches caractérisé par des numéros (ID), et les arcs ($c_i \in C$) représentent les liens entre les différentes tâches, ils sont caractérisés par un coût de communication qui désigne le nombre de paquets envoyée d'une source à une destination.

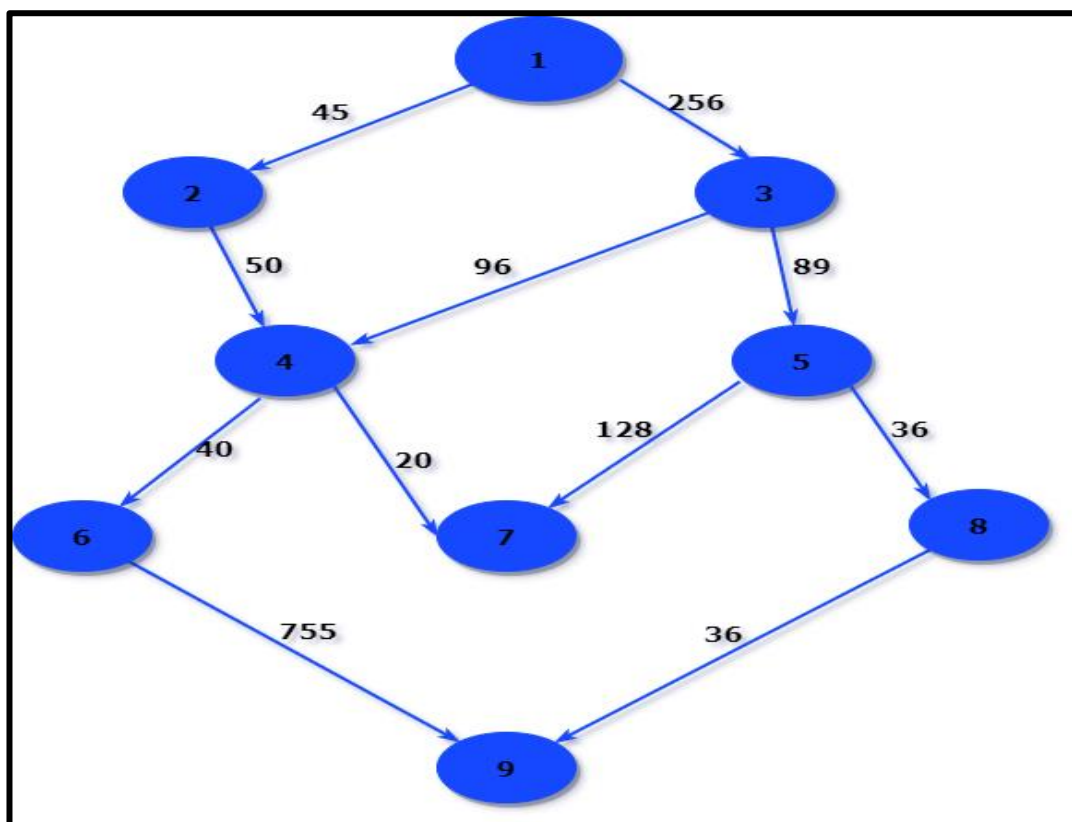


Figure 3-3: Graphe des tâches.

Nous avons utilisé deux types de graphe des tâches, des graphes générées aléatoirement par l'outil TGFF qui permet de générer des fichiers xml automatiquement, et des graphes des applications réelles existant aux littératures (VODP, PIP, MDW, MPEG4).

Pour simplifier l'extraction des données, Nous avons traduit les fichiers générés par le TGFF à notre structure personnalisée, également pour les graphes des applications réelles. Le fichier est structuré sur des balises comme suit :

Tableau 2:Les noms des balises.

Balise	Description
Application	Décrit le nom de l'application.
Layer	Désigne sur quel niveau situe la tâche.
Tâche	Décrit le numéro de tâche.
Débit	Définit la quantité des données envoyées par second (nous avons l'initialise à zéro).
Énergie	Définit l'énergie consommé par la tâche (nous avons l'initialise à zéro).
Connexion	Représente la liste des voisins de la tâche.
Data size	Représente le coût de communication entre deux voisins.

La figure 3.4 et 3.5 montre respectivement la structure de fichier xml et sa format DTD (Document Type Definition) :

```

<?xml version="1.0"?>
- <application name="VOPD">
  - <layer name="layer1">
    - <tache id="tache1">
      <debit>0</debit>
      <energie>0</energie>
    - <connction>
      - <tachedist id="tache2 ">
        <debit>0</debit>
        <energie>0</energie>
        <datasize>70</datasize>
      </tachedist>
    </connction>
  </tache>
</layer>

```

Figure 3-4: Structure XML.

```

<!ELEMENT application (layer+)>
<!ATTLIST application
  name CDATA #REQUIRED>
<!ELEMENT layer (tache+)>
<!ATTLIST layer
  name CDATA #REQUIRED>
<!ELEMENT tache (débit, Energie, connexion)>
<!ATTLIST tache
  id CDATA #REQUIRED>
<!ELEMENT debit (#PCDATA)>
<!ELEMENT Energie (#PCDATA)>
<!ELEMENT connexion (tachedist+)>
<!ELEMENT tache-dist (debit, énergie, data-size)>
<!ATTLIST tache-dist
  id CDATA #REQUIRED>
<!ELEMENT data-size (#PCDATA)

```

Figure 3-5 : Schéma DTD.

Pour la phase de l'implémentation nous avons modélisé le graphe des tâches par la class Application, la tâche par la class Tâches et les arcs par la class Connecte.

Class _ tâche (string ID, Integer numéro-PE, Arrayliste<class-connect> list des voisin).

- String ID : un numéro attribué à chaque tâche pour l'identifier.
- Integer Numéro PE : décrit la position de tâche sur le Noc.
- Arraylist<class_connect> list voisin : Une liste contient tout le voisin communique directement avec cette tâche.

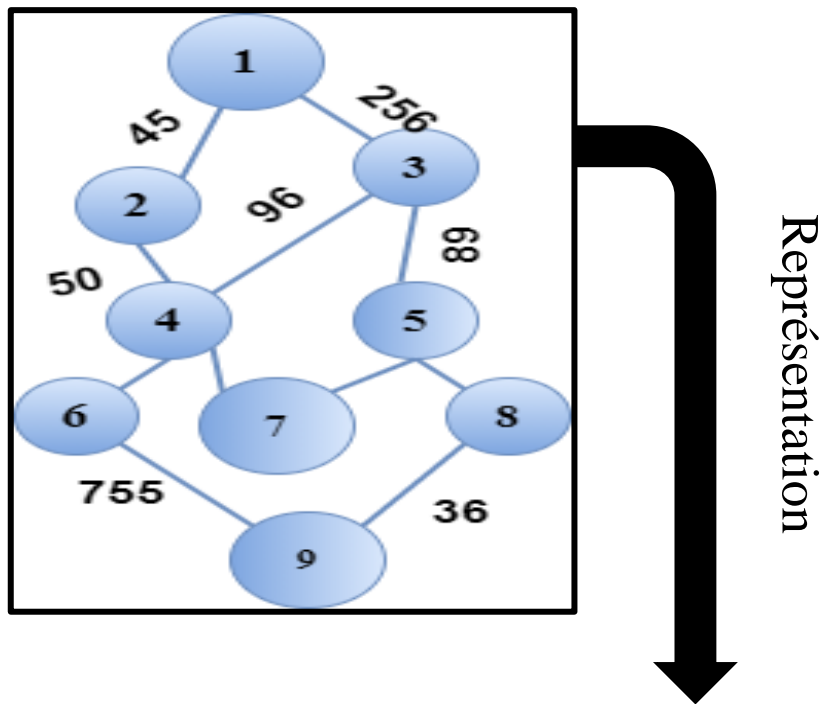
Class_connect (*Class_tâche* source, *Class* tâche destination, *Integer* number bit).

- *Class_tâche* source : représente une tâche source, le point de départ d'envoi des données.
- *Class* tâche destination : Représente une tâche destination qui reçoit les données.
- *Integer* Numbers bit : la quantité des données envoyée d'une source à une destination.

Class_Application (String Nom application, Integer number layer, Number des tâches, Arraylist<arraylist<tâche >> Architecture)

- String Nom application : décrit le nom de l'application.
- Integer number layer : représente le nombre des niveaux (couches) de notre graphe.
- Number des tâches : décrit le nombre total des tâches de notre graphes des tâches.
- Arraylist<arraylist<tâche >> Architecture : Représente la structure de notre application.

La relation entre ces classes est illustrée sur le diagramme de class (figure 3.6) :



Représentation

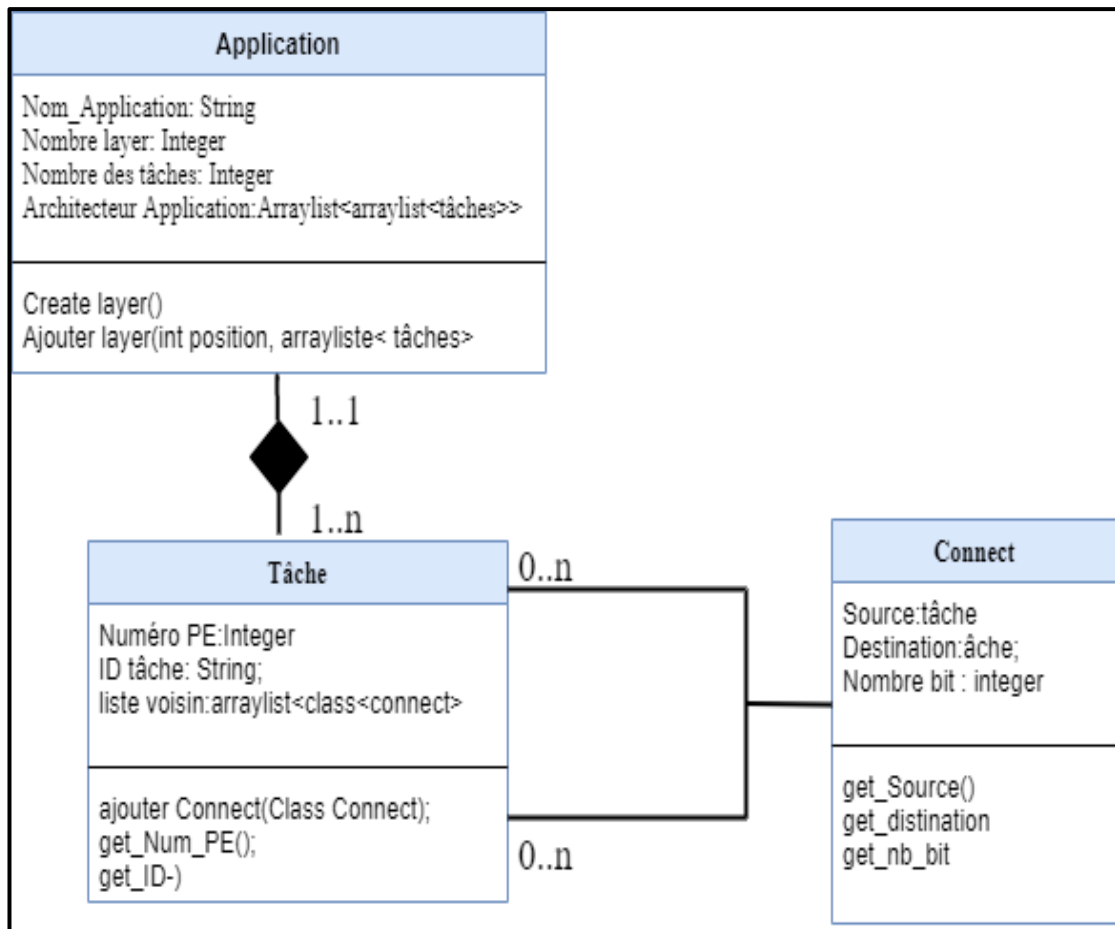


Figure 3-6: Diagramme des classes.

3.3.2 Module 2 : Topologies

Le **Module 2** représente le 2-ème entrée qui définit la topologie ou l'architecture Noc, nous avons utilisé :2D MeSH simple, 2D MeSH wifi et 3D avec des liens (TSV). Nous avons considéré dans ce travail que les architectures sont homogènes (les nœuds de calcul sont tous les mêmes).

Nous avons modélisé chaque topologie par une classe qui hérite son comportement basic de la classe *Topologie*, La figure 3.7 représente le diagramme de class qui définit la relation entre ces classes :

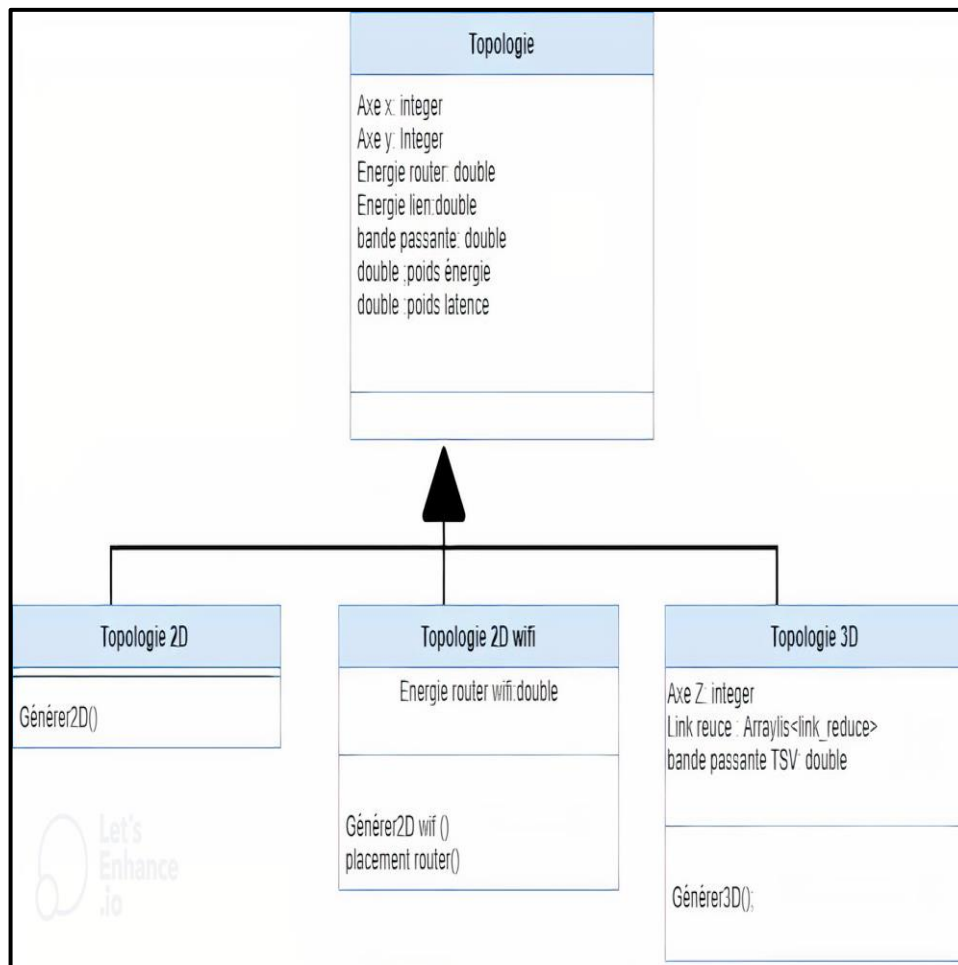


Figure 3-7:diagramme des classes des Topologie.

3.3.2.1 Représentation de Topologie 2D simple

L'architecture avec 2 dimensions se définit par 2 axes, Axe X horizontale et vertical Y, chaque axe comporte un nombre des PE, donc sa taille est $X*Y$, pour la partie de

l'implémentation nous avons la présenter par une class java avec des attribue et des fonctions.

Architecture_2D (Integer Arc X, Integer Arc Y, double Energie Router, Double Energie lien double Bande passante entre PE, Double poids énergie, double poids latence, Nombre tâche Array-list< Class tâche>).

- Integer Arc X, Integer Arc Y : représente respectivement le nombre des PE dans l'axe X et le nombre des PE dans l'axe Y.
- Double Energie Router : l'énergie consommé par router
- Double Energie lien : énergie consommée par lien
- Double Bande passante entre PE : la capacité d'envoi les données dans un PE.
- Double poids énergie : un coefficient attribué à l'énergie.
- Double poids latence : un coefficient attribué à la latence.
- Nombre tâche Array-list< Class tâche> : nombre des tâches dans un PE.

La **fonction générer** c'est la fonction principale de cette class elle nous permettre de crée notre architecture sous forme une liste chaine avec une taille X*Y d'où chaque case représente un PE, qui contient une liste chaine d'un ensemble des tâches (au début la liste se crée vide sans affecter les tâches au PE) (Figure 3.8)

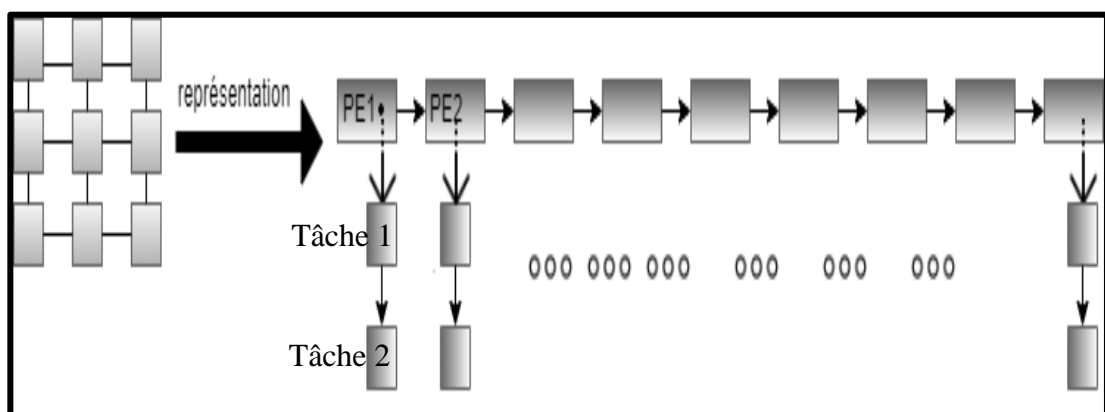


Figure 3-8: Représentation de la classe 2D.

La figure 3.9 ci-dessous représente un pseudo code de la fonction Générer ()


```
Générer ()  
Input : indice début ; Indice fin ;  
Out put: Array list< class_tâche> ;  
Tanquant (indice début<= indice fin) {  
1.Créer la liste des Tâche ;  
2.Ajouter la liste des tâches à la liste de noc ;  
3.Incrémenter début ;}  
Fin
```

Figure 3-9:Pseudo code de la fonction générer ().

3.3.2.2 Représentation Architecture 2D wifi

L'architecture 2D wifi rassemble à l'architecture 2D simple, en plus d'autre caractéristique. Dont la communication entre les PE se fait avec 2 types d'interconnexion : interconnexion sans fils, et interconnexion filaire. La taille de Noc est divisée en quatre (4) sous réseau, dans chaque sous réseau un routeur wifi se placée au milieu. L'avantage de cette topologie par rapport au topologie 2D simple réside dans sa capacité de réduire le nombre de saut entre la source et la destination, d'où la communication dans le même sous réseau est assurée par les liens et entre les sous réseaux est réalisée à l'aide des routeurs wifi.

Pour la phase d'implémentation nous avons la représenter par une class :

Class 2D_Wifi (Integer Arc X, Integer Arc Y, double Energie Router, Double Energie lien double Bande passante entre PE, Double poids énergie, double poids latence, Nombre tâche Array-list< Class tâche>, Double énergie Router wifi).

Cette classe a deux principales fonctions, ils sont illustrés sur le diagramme des class (figure 3.10). La fonction **générer ()** permet de crée l'architecture et la fonction **placement wifi ()** permet de placer les routeurs wifi, Voici un pseudo code :

```
Générer () ;  
Input : Début, fin ;  
Out put: Array-list< class Tâche >  
Tanquant (Début<fin) {  
1.Créer la liste des Tâche ;  
2.Ajouter la liste des tâches à la liste de noc ;  
3.Incrémenter début ;}  
Placement wifi () ;  
}  
Fin.
```

Figure 3-10:Pseudo code de la fonction générer () 2Dwifi.

La **fonction Placement wifi ()** décide le bon placement des routeurs Wifi, la taille de Noc varie d'une architecture à autre selon le nombre des PE dans l'axe X et Y, Nous distinguons alors quatre principaux cas :

- ✓ Axe x et y pair,
- ✓ Axe x et y impair,
- ✓ Axe x impair et y pair ;
- ✓ Axe x pair et y impair ;

Nous allons les explique dans la section suivante.

- **Cas impair * impair :**

Un routeur se place au milieu pour chaque sous-réseau, dans ce cas le nombre d'axes X et Y sont impaires, il faut donc compter la première fois la ligne médiane avec le sous-réseau 1 et la deuxième fois pour le sous-réseau 2 (sens horizontal), De même dans le sens vertical nous comptons la lignes une fois pour le sous-réseau 1 et une fois pour le sous-réseau 3. (Figure 3.11)

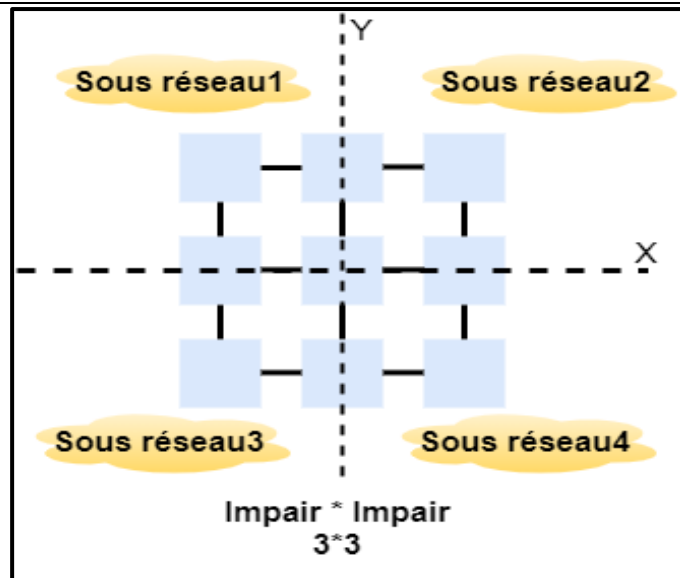


Figure 3-11: Représentation de Cas impaire * impaire.

- **Cas pair * Impair**

Dans ce cas l'axe X est pair et l'axe des Y impaire, l'axe X se divise en 2 sans aucun problème. Pour l'axe des Y nous appliquons la même règle que le premier cas, nous prenons la ligne de centre une fois pour le réseau 1 et autre pour le réseau 3 etc. (figure 3.12)

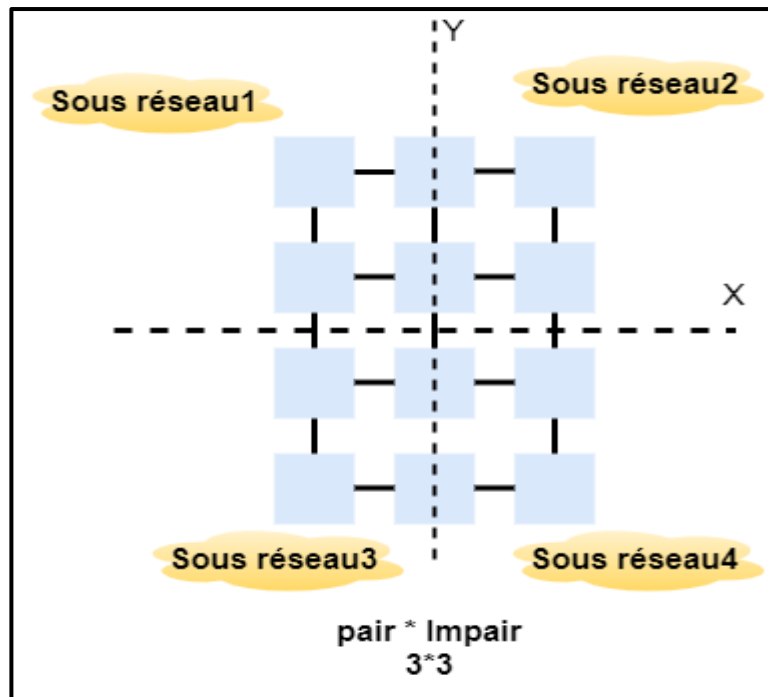


Figure 3-12: Représentation de Cas pair*impair.

- **Cas impair*pair**

Dans ce cas de l'axe X est impaire et l'axe des Y pair, Nous divisons l'axe Y en 2 sans aucun problème, et l'axe des X sera traité comme le cas précédant, Nous prenons la ligne de centre une fois pour le réseau 1 et autre pour le réseau 3. (Figure 3.13)

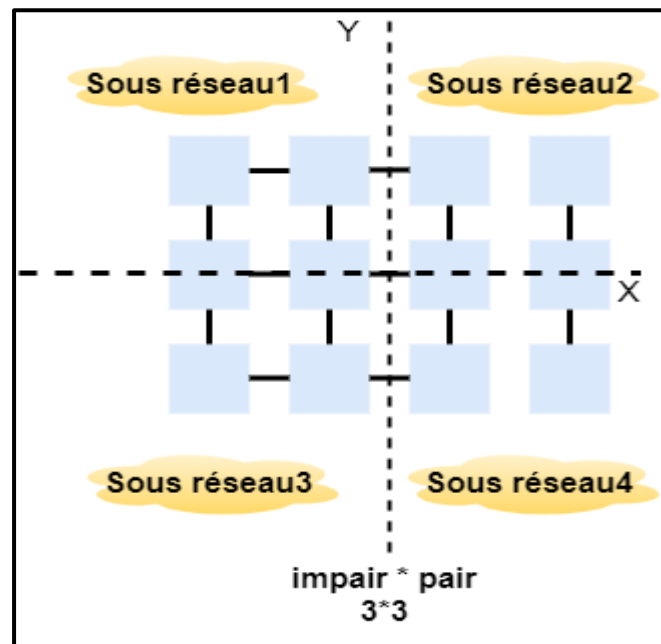


Figure 3-13: Cas impair*pair.

- **Cas pair *pair**

Dans ce cas les axe sont égaux donc il suffit juste divise en deux et placer un routeur au milieu de chaque sous réseau. (Figure3.14)

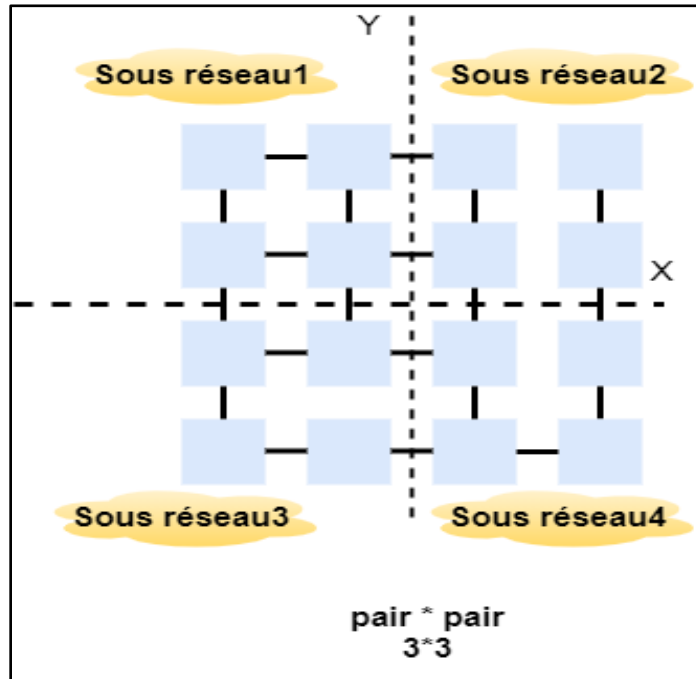


Figure 3-14: Cas pair*pair.

3.3.2.3 Représentation de l'architectures 3D

L'architecture 3 dimension se définit par 3 axes X , Y , Z , elle constitué d'un ensemble des architecture 2D répartie en des couches. Dans le cas ou $Z = 0$ elle se voie comme une architecture 2D.

La communication entre les PE de même couche est assurée par les liens directs, et la communication entre les couches est assuré par des liens vertical (TSV). Nous avons modélisé sa structure par une liste chaine d'où chaque case représente une couche qui contient une liste d'une architecture 2D. (figure 3.15)

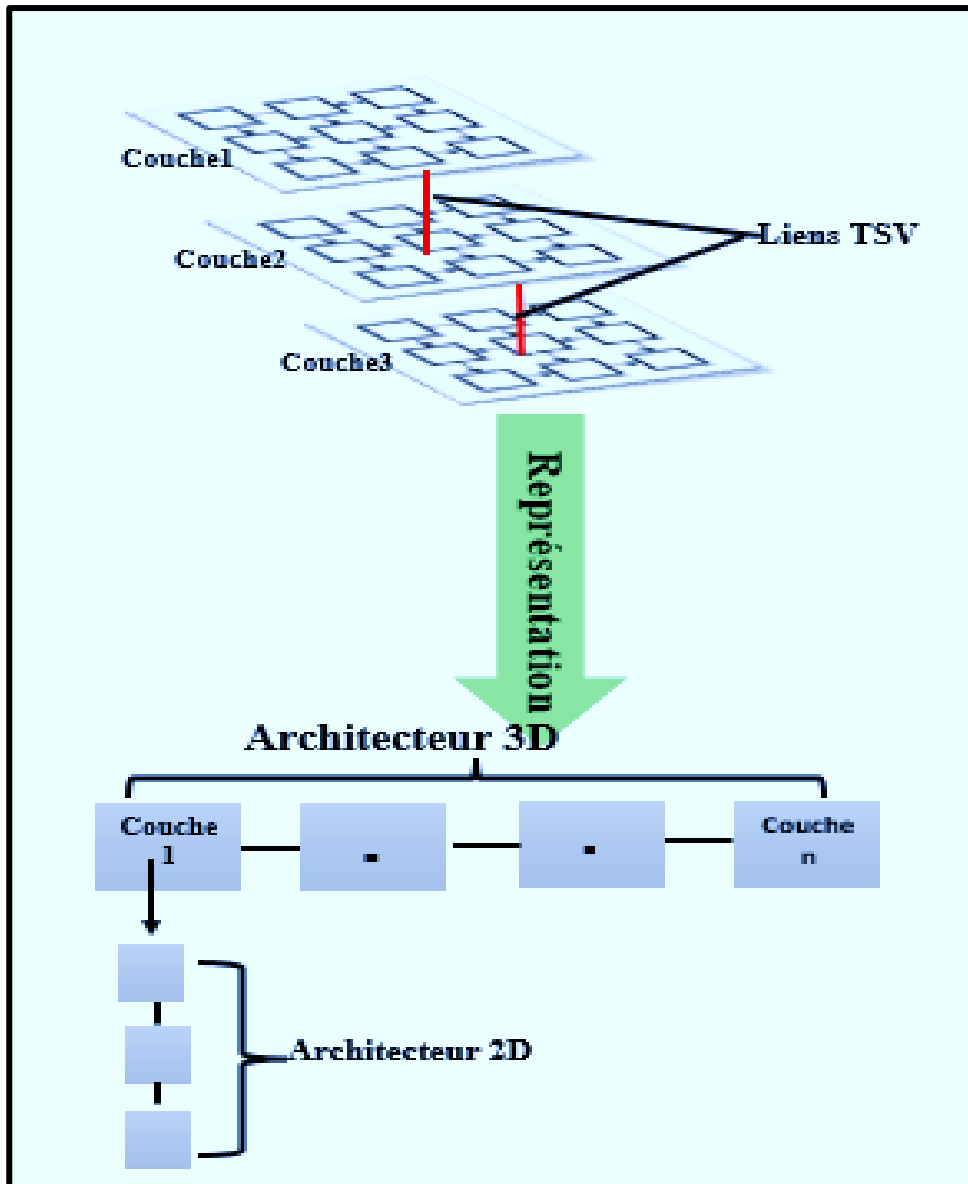


Figure 3-15: Représentation de la structure 3D.

Pour la phase d'implémentation nous avons représenté cette topologie par une classe :

Classe 3D (Integer Arc X, Integer Arc Y, Integer Arc Z, double Energie Router, Double Energie lien double Bande passante entre PE, Double énergie lien TSV).

Cette classe comporte une fonction principale qu'est la fonction **générer ()**, pour crée sa structure. (Figure3.16)

```
Générer3D () ;  
Input : Début, Z ;  
Out put: Array-list< class tâche>  
Tanquant (Début<Z) {  
1.Generer 2D () ;}  
{  
2.Ajouter la liste d'architecture 2D a la liste 3D ;  
}  
  
Fin.
```

Figure 3-16:Pseudo code de fonction Générer 3 D ().

3.3.3 Présentation de F1 & F2

La fonction 1 et la fonction 2 fournit les entrées de la phase de mapping. Le mapping consiste à mapper un nombre des tâches d'une application ou les composants d'un logiciel sur une architecture Noc, (figure 3.17) cette phase se fait comme suit :

1. Fonction1 (F1) : permet à l'utilisateur de lire un fichier Xml, ce fichier décrit une application et son comportement.
2. Fonction 2 (F2) : permet à l'utilisateur de Choisir une architecture parmi les architectures proposées et définit ses paramètres.
3. Lancer l'algorithme de chauves-souris pour choisir le bon placement des tâches.

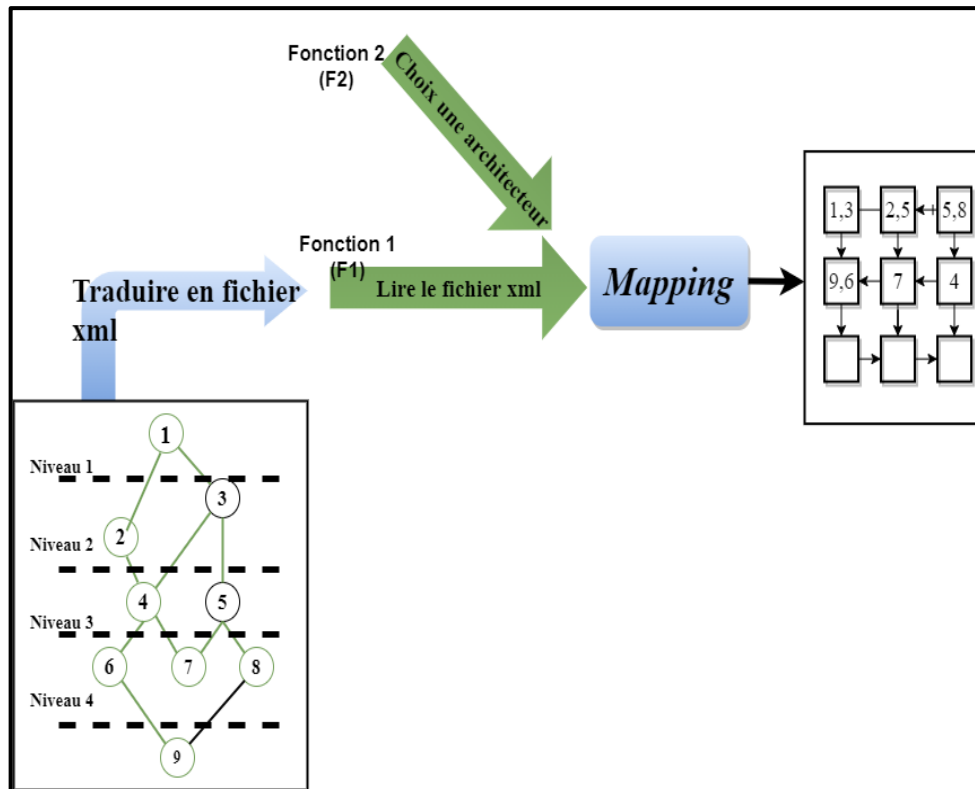


Figure 3-17:Présentation de problème de mapping.

Nous allons présenter l’algorithme de chauves-souris dans la section suivante, pour mieux comprendre son adaptation dans le module 3.

3.3.4 Algorithme des chauves-souris

L'algorithme est basé sur le comportement d'écholocation des microbots, qui utilisent un type de sonar appelé écholocation, avec des fréquences d'impulsions d'émission et d'intensité variables, pour détecter les proies, éviter les obstacles et localiser les crevasses de repos dans l'obscurité. L'idéalisation de l'écholocation des microchiroptères est la suivante (Yang, 2010) :

- ✓ Les chauves-souris utilisent l'écholocation pour détecter la distance et faire la distinction entre la nourriture, les proies et les barrières de fond(Yang, 2010).
- ✓ Chaque chauve-souris virtuelle vole au hasard avec une vitesse v_i à la position (solution) x_i avec une fréquence fixe f_{min} , une longueur d'onde λ variable et une intensité sonore A_0 pour rechercher une proie. Au fur et à mesure qu'il cherche et trouve sa proie, il change la longueur d'onde (ou la fréquence) de ses impulsions émises et ajuste le taux d'émission d'impulsions r , en fonction de la proximité de la cible(Yang, 2010).

- ✓ l'intensité sonore variera d'une valeur A_0 (initialement grande et positive) à une valeur minimale constante A_{\min} (Yang, 2010).

La figure 3.14 présente l'organigramme de l'algorithme de chauves-souris dans le cas générale :

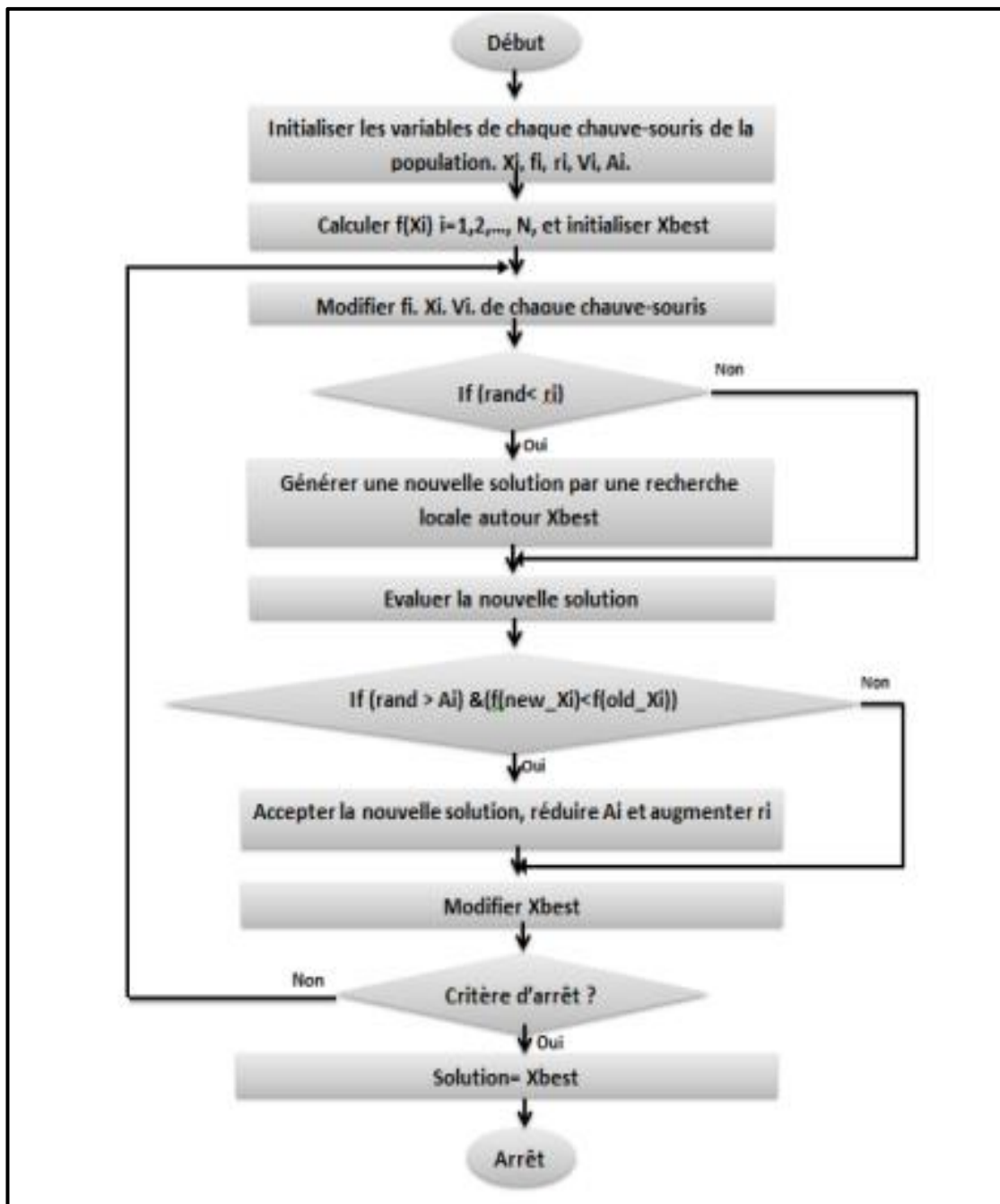


Figure 3-18: L'organigramme de l'algorithme de chauves-souris. (Sappa & Dornaika, 2019)

Ce module présente l'algorithme de chauves-souris et ses principales étapes, Ces étapes peuvent être résumées comme suit :

3.3.4.1 Formalisation de l'algorithme

Les chauves-souris déplacent par des groupes (Swarm) afin de trouver leurs proies, Un groupe représente une population initiale, d'où chaque chauve-souris définit une solution dans notre espace de recherche (chauve-souris = une solution de mapping), et une proie représente la solution optimale. (Figure 3.19)

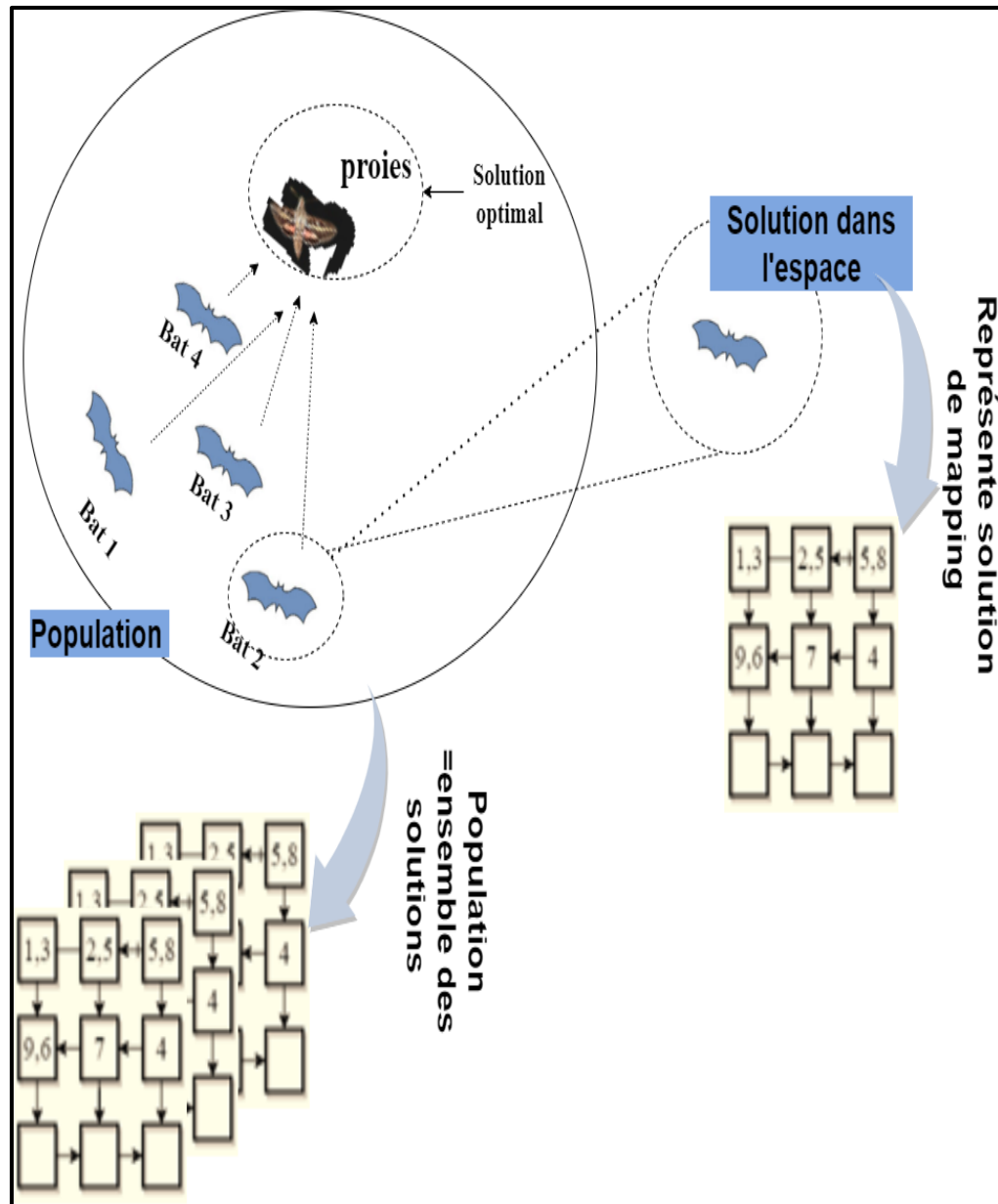


Figure 3-19: Formulation de solution.

3.3.4.2 Initialisation des paramètres

L'algorithme des chauves-souris se définit par la population initiale, la position x , la vitesse v , la fréquence f , la pulsion r et le volume A , ainsi la fonction d'objective

$f(x)$, nous allons expliquer ses paramètres et comment nous avons les implémenter sur notre travail.

1. **Population initiale** : C'est un nombre n des chaves souris à des positions différentes, chaque chaves souris représente une solution dans l'espace générée aléatoirement. Nous avons décrit un individu de la population par une liste vide, ensuite nous avons affecter à chaque case un ensemble des tâches dans un PE avec la fonction **Random** en respectant la capacite de ce PE, cette capacite est un pourcentage choisi par l'utilisateur.

La fonction **Random** donne des valeurs entre 0 et 1, donc nous avons la multipliera par la taille de réseau sur puce pour nous donne des positions dans l'intervalle que nous besoin.

La **figure 3.20** présente un pseudo code de la génération d'une population :

```
Fonction Générer _ Population ()  
Input : La taille de population  
Out put : les listes des architectures  
Taquant ( $i \leq$  taille population) {  
1. Générer une architecture Noc (refaire)  
2. affecter les tâches à chaque PE, en respectant la capacite.  
Incrémenter  $i$ .  
}
```

Figure 3-20:Pseudo code de la fonction Générer une population.

2. **Vitesse** : Chaque chauve-souris de la population se définit par une position à un instant t , pour qu'il changer cette position, il nécessite une vitesse V pour voler dans l'espace, le changement de la position X_i à la position X_{i+1} nous donne une nouvelle solution, c'est-à-dire une nouvelle liste (architecture noc) avec des positions des tâches différent.

Une vitesse s'attribue à chaque position, pour cela nous avons représenté cette valeur par une liste chaîne de même type que la position (au premier lieu nous avons l'initialiser à 0)

- 3. La fréquence, pulsion r et volume A :** les chauves-souris volent dans l'espace avec des vitesse v à des positions x en basant de son des proies, ils ont besoin donc d'une fréquence f , en variant la longueur d'onde et le volume A de recherche de proie. Elles peuvent ajuster automatiquement la longueur d'onde (ou la fréquence) ainsi que le taux d'émission de pulsation r , selon la proximité de leurs proies. La relation entre la fréquence et la vitesse se donne par la relation : $\lambda = \frac{v}{f}$

Nous affectons à chaque chauves souris de la population une valeur de f dans un intervalle $[f_{\min}, f_{\max}]$, le volume $A \in [A_{\min}, A_{\max}]$, et $r \in [0, 1]$.

- 4. La fonction d'objective :** l'évaluation de chaque chauve-souris afin de trouver des nouvelles solutions s'effectue à l'aide d'une fonction d'objective, nous avons formulé pour chacune topologie sa fonction de calcul de nombre de saut.

- **Pour la topologie 2D**

Totale Nombre de saut

$$= \sum_{i=0}^{n=\text{taille de noc}} \text{Poid de communication} * \text{Distance Manhattan.}$$

D'où :

Le poids de communication : représente la quantité des données envoyés d'une source à un destination.

La distance Manhattan = $|X_{\text{source}} - X_{\text{destination}}| + |Y_{\text{source}} - Y_{\text{destination}}|$

- **Pour la topologie 3D :**

La fonction d'objective se calcule avec la même manière que 2D, sauf que nous ajoutons le lien vertical donc la distance devient :

La-distance-Manhattan = $|X_{\text{source}} - X_{\text{destination}}| + |Y_{\text{source}} - Y_{\text{destination}}| + |Z_{\text{source}} - Z_{\text{destination}}|$.

- **Pour la topologie 2D wifi**

Dans cette architecture la distance se calcule avec deux méthodes :

1. Dans le même sous réseau :

La distance est la même que l'architecture 2D.

2. Dans deux sous réseau différent :

Nous calculons la distance entre la source et le routeur wifi par la distance Manhattan, de même la distance entre le routeur et la destination, et la distance entre les deux routeurs wifi se calcule par la distance euclidienne. Sachant que :

La distance euclidienne = $\sqrt{(X_{r-s} - X_{r-dis})^2 + (Y_{r-s} - Y_{r-dis})^2}$.

X_{r-s}, Y_{r-s} :représente les axe X et Y de routeur source.

X_{r-dis}, Y_{r-dis} :représente les axe X et Y de routeur destination.

Après le calcul des deux destinations nous cherchons le minimum, Pour choisir la bonne méthode.

Et les distances globales dans ces architectures peuvent être résume comme suit :

Distance Globale = distances filaires + distances wifi

3.3.4.3 Déroulement de l'algorithme

Selon le pseudo code après l'initialisation des paramètres, Nous cherchons une meilleure solution parmi les individus de la population, le choix de cette solution se fait par l'évaluation de la population i , d'où l'individu avec la minimum fonction d'objective sera sélectionné comme la bonne location courante X^* .

Ensuite, Nous crions des nouvelles solutions par des changements de la position, la vitesse et la fréquence, cela se fait par ces 3 formule :

$$f_i = f_{min} + (f_{max} - f_{min})\beta$$

$$V_i = V_i + (x_i - X^*)f_i$$

$$X_i = X_i + V_i$$

D'où β est une valeur dans l'intervalle $[0,1]$.

Une nouvelle solution sera générée à partir la location courante X^* en utilisant la formule :

$$X_{nouvelle} = X_{precedante} + \varepsilon A^t$$

D'où ε est un nombre aléatoire dans $[-1, 1]$, cette formule sera appliquée après la vérification que rand est supérieur à r_i pulsation, nous ferons une comparaison avec l'individu courant si $\text{rand} < A_i$ et $f(x) < f(X^*)$ nous gardons cette solution et ferons des mises à jour sur la valeur de l'intensité et la pulsion, l'intensité A sera diminuée et la pulsion r sera augmenter :

$$A_i^{t+1} = \alpha A_i^t$$
$$r_i^{t+1} = r_i^0 (\exp(-\gamma t)).$$

Dans chaque itération nous gardons une solution courante X^* , la solution globale sera sélectionnée par une comparaison entre ces derniers. (Figure 3.21)

La solution globale s'affiche sur l'interface de logicielle sous forme un tableau, ensuite l'utilisateur a le droit de la sauvegarder sous forme un fichier xml. Ce fichier est structuré par des balises et comporte toutes les informations sur la structure de la solution finale : la taille d'architecture, Capacité de chaque PE, la quantité des données entre les PE, le totale d'énergie et temps d'exécution consommé. L'utilisateur a le droit aussi d'ajouter les informations importantes de la solution finale sur un fichier d'historique de type xml, ce fichier contient tous les résultats générés par notre application.

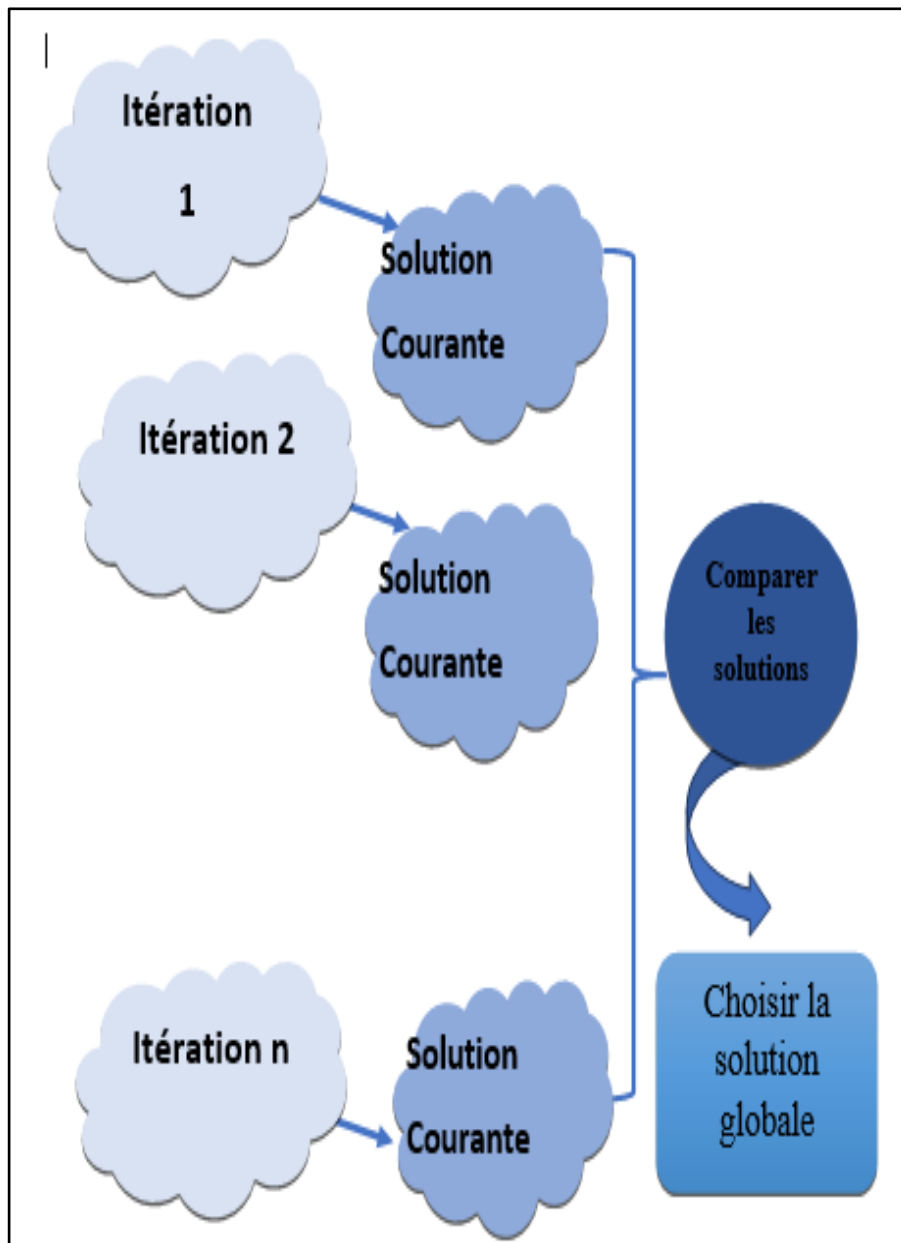


Figure 3-21: Choisir la solution Globale.

La figure ci-dessous représente le format de fichier Xml de la solution finale :

```

<?xml version="1.0" encoding="UTF-8"?>
<Test>
  - <Noc_archiectuer nom_architecteur="noc_without_wifi" id_architecteur="1">
    <Axe_x>2</Axe_x>
    <Axe_y>4</Axe_y>
    <max_tache_pe>1.0</max_tache_pe>
    <Energy_in_router>0.1</Energy_in_router>
    <Energy_in_link>0.1</Energy_in_link>
    <energie_excution>0.1</energie_excution>
    <poids_energie>1.0</poids_energie>
  </Noc_archiectuer>
  - <solution total_energy="243.2" coust="640.0">
    - <tache id="tache2">
      <numero_pe>8</numero_pe>
      <debit>0.0</debit>
      <energie>0.0</energie>
    - <connection>
      - <tache_destination id="tache1">
        <debit>0.0</debit>
        <energie>0.0</energie>
        <datasize>128.0</datasize>
      </tache_destination>
      - <tache_destination id="tache3">
        <debit>0.0</debit>
        <energie>0.0</energie>
        <datasize>64.0</datasize>
      </tache_destination>
    </connection>
  </tache>
  
```

Figure 3-22:Exemple de fichier Xml de la solution finale.

3.4 Conclusion

Ce chapitre a été consacré à la présentation de la phase de conception et de l'implémentation de notre travail, Nous avons abordé ce chapitre par les problèmes d'optimisation et ses classifications, d'où nous avons met l'accent sur les méta-heuristique car c'est la classe qui comporte notre algorithme proposé. Ensuite, Nous avons détaillé l'algorithme de chauves-souris et la manière dont laquelle nous avons l'adapté pour résoudre le problème de mapping.

Dans le prochain chapitre, nous allons présenter les machines et les outils utilisés afin de mettre en œuvre notre solution, ainsi les benchmarks sur lesquels nous allons effectuer les tests ainsi que la discussion sur les résultats finaux.



Chapitre 4 : Tests & Résultats

4.1 Introduction

Le problème de mapping est un problème d'optimisation combinatoire NP-difficile. Théoriquement, mapper un nombre N des IPs sur M nœuds de réseau sur puce (avec $N < M$), implique $M! / (M - N)!$ arrangements IP possibles sur les nœuds (enlever et déplacer dans chapitre 2). La solution optimale d'un problème d'optimisation peut rarement être déterminée en temps polynomial. Il est alors nécessaire de développer une méthode heuristique ou métaheuristique qui fournit une solution quasi optimale (ou optimale) en un temps CPU raisonnable. Ces méthodes sont inspirées de la nature et du comportement des animaux, comme l'algorithme de chauve-souris que nous avons choisi pour ce travail. Afin d'implémenter cette technique, elle nécessite des langages et des outils, que nous aborderons dans ce chapitre, ainsi que le logiciel réalisé pour tester nos benchmarks, et enfin nous discuterons les résultats obtenus.

Les tests sur l'algorithme de chauves-souris proposés sont réalisés sur une machine équipée d'un processeur Intel Core I5, 2,53GH, une Ram de 8Go sous le système d'exploitation Windows 10 64 bits. (section)

4.2 Outils d'implémentation utilisés

Nous avons utilisé le langage Java pour implémenter notre travail, avec l'environnement NetBeans.

4.3 La génération des Benchmark : TGFF

TGFF a été initialement développé en 1998 par R.P. Dick et D.L. Rhodes pour faciliter les benchmarks aléatoires normalisés pour la recherche sur la planification et l'allocation, en général, et la recherche sur la co-synthèse matériel-logiciel, en particulier. TGFF convient à de nombreuses applications nécessitant la génération de graphes pseudo-aléatoires. K. Vallerio a ensuite mis à jour et amélioré le code et la documentation. La dernière version (3.0) étend les fonctionnalités de TGFF, fournissant un générateur de graphes aléatoires hautement configurable capable de générer plusieurs types de graphes aléatoires (Vallerio, 2008).

4.4 Les Benchmark utilisés

Les benchmarks sont des graphes de tâches utilisés pour effectuer des tests, nous avons utilisé des graphes standard (PIP, MDW, VODP, MPEG4) et cinq graphes aléatoires.

4.4.1 PIP

La fonction PIP, inventée par Greg Dockery, permet de regarder une deuxième chaîne sur un écran de télévision. La première chaîne est affichée en plein écran tandis que la seconde est affichée dans une petite fenêtre. L'application PIP se compose de huit PE communiquant via huit liens. Le nombre total de paquets échangés est de 576 (Gregory et al., 1997).

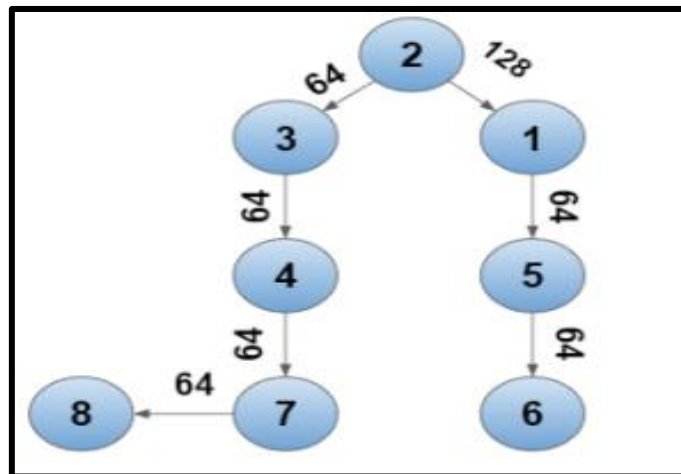


Figure 4-1: Application PIP.

4.4.2 MDW

L'application MWD se compose de 12 adresses IP communiquant via 13 liens. Le nombre total de paquets échangés est de 1 120 (Benatchba et al., 2005).

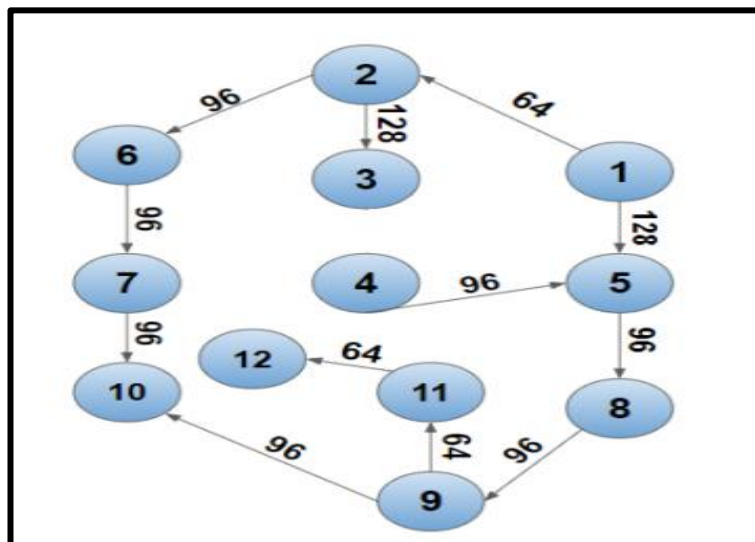


Figure 4-2: Application MDW.

4.4.3 VODP

L'application VODP se compose de 16 IP qui communiquent via 20 liens. Le nombre total de paquets envoyés est de 3 731 (Carvalho et al., 2009).

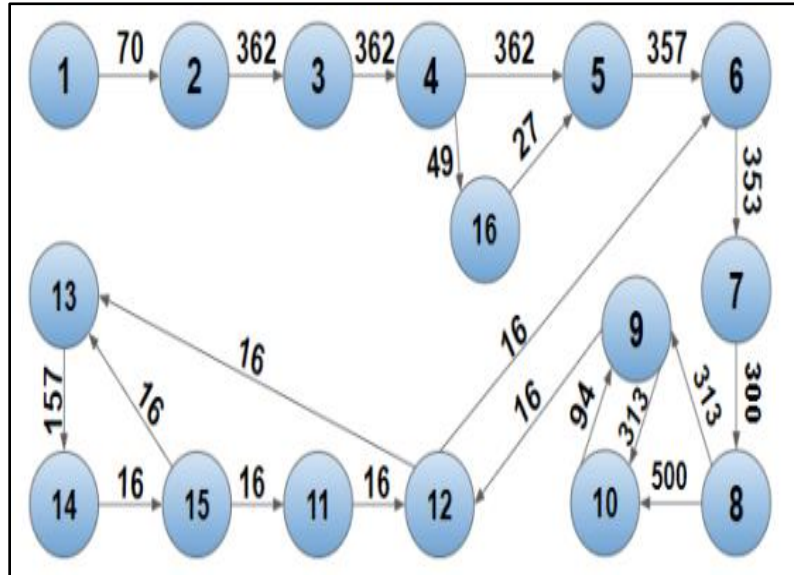


Figure 4-3: Application VODP.

4.4.4 MPEG4

Normalise les algorithmes de codage audiovisuel dans les applications multimédias. Il se compose de 12 IP communiquant via 26 liens. Le nombre total de paquets échangés est de 3 466 (Katto & Ohta, 1996).

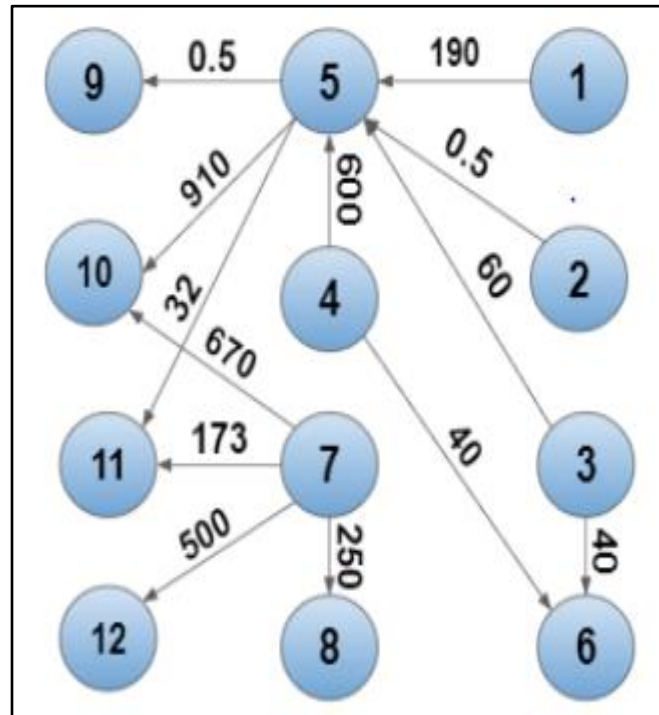


Figure 4-4: Application MPEG.

4.4.5 Benchmark Aléatoire

Nous avons utilisé également cinq graphes générés aléatoirement par TGFF, Ces graphes appelées TG6, TG7, TG8, TG9, TG10. Ils sont présentés dans le tableau 4.1 ci-dessous :

Tableau 3: Benchmarks Aléatoire.

Graphes	La taille de graphe
TG 6	50
TG7	30
TG8	27
TG9	16
TG10	12

4.5 Présentation des interfaces

1. L'interface d'accueil représente la première interface affichée après l'exécution de l'application. Le menu de gauche permet d'initialiser les entrées pour démarrer le mapping. (Figure 4.5)

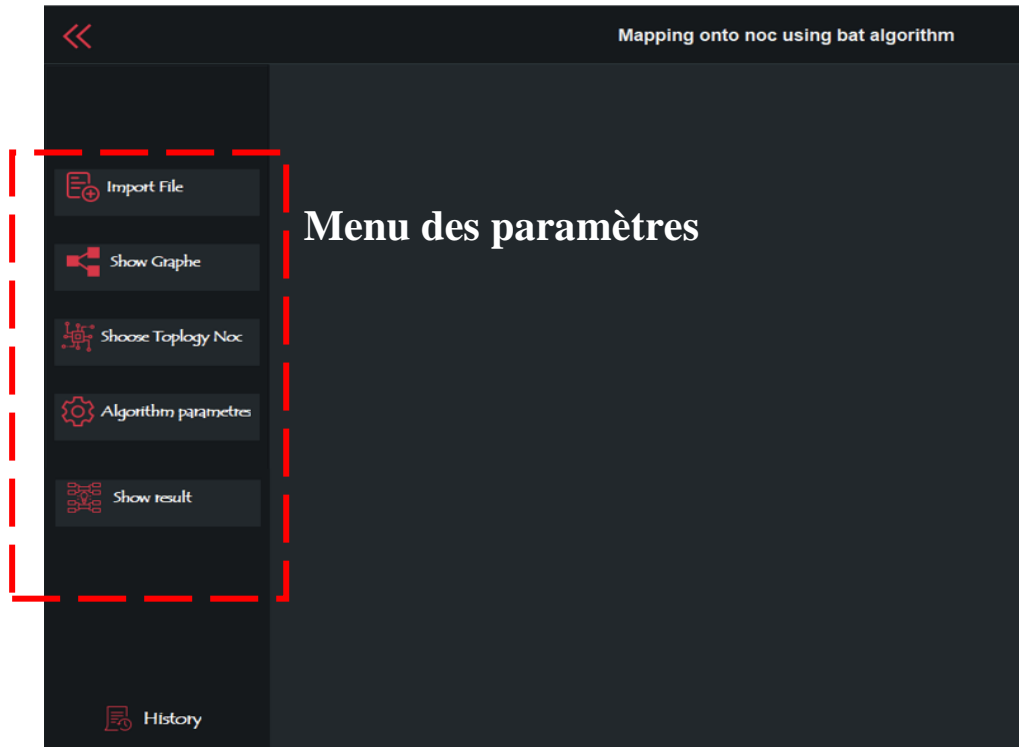


Figure 4-5:Interface d'accueil.

2. D'abord un benchmark sera sélectionné, en cliquant sur un bouton « **Import file** » l'utilisateur pourra lire le fichier de type xml qui définit son graphe d'application. (Figure 4.6)
3. Lorsqu'une application est sélectionnée, l'utilisateur clique sur le bouton « **show graphe** » afin d'afficher une représentation graphique de l'application. (Figure 4.7)
4. Après le choix de benchmark l'utilisateur sélectionne alors une architecture Noc, le Button « choose Noc » donne la main à choisir une topologie (figure4.8) et initialise les paramètres de topologie (le nombre des PE dans l'axe X, le nombre de PE dans l'axe Y et le maximum des tâches dans un PE, figure 4.9) la taille du Noc dépend de la taille du graphe d'application (GA), d'où la taille $GA \leq \text{taille NOC}$, pour cela le Button « choose Noc » sera active uniquement si l'utilisateur sélectionne un benchmark.

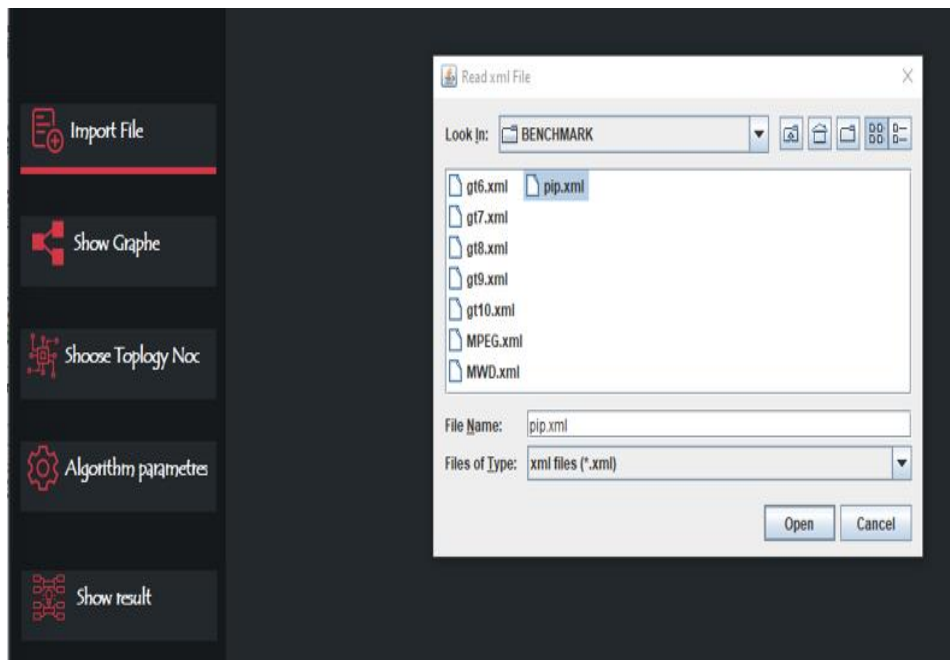


Figure 4-6: Choix de benchmark.

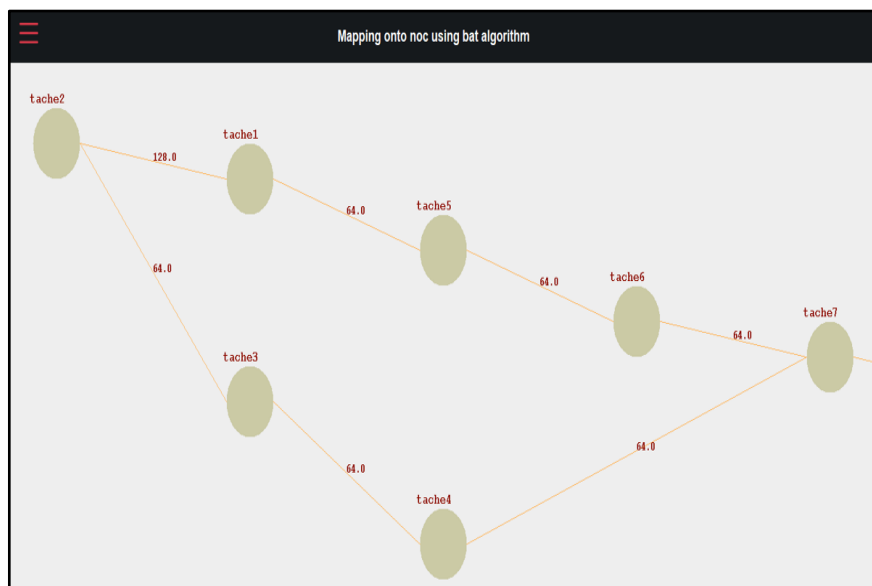


Figure 4-7: Affichage de Graphe.

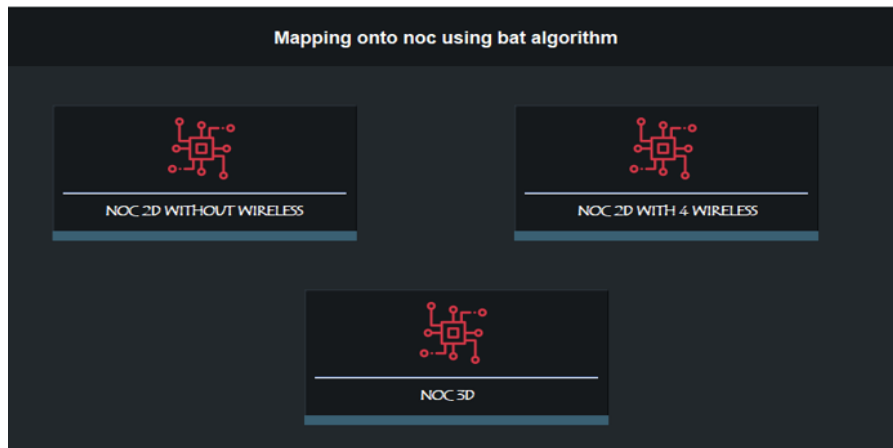


Figure 4-8: Choix de Topologie.



Figure 4-9: fixer les paramètres de topologie.

5. L'utilisateur doit ensuite définir les paramètres de l'algorithme de chauve-souris, tel que le nombre des itérations, nombre des chauves-souris, la valeur de volume et pulsion, ainsi que la valeur de beta et y. (figure 4.9)

6. Les résultats obtenus sont synthétisés dans un tableau, en cliquant sur le Button 'Show result' le tableau se remplit avec les informations de mapping (l'emplacement des tâches, leurs communications, et le coût de communication avec ce mapping). Le résultat sera automatiquement stocké dans l'historique (figure 4.11) et l'utilisateur pourra également l'enregistrer sous forme de fichier xml sur sa machine. (Figure 4.12)

Mapping tasks onto noc using bat algorithm

Bat algorithm parameters


iteration

bat loudness pulse rate

alpha y

Figure 4-10: Paramètres BAT.

History



name noc	axe x	axe y	axe z	iteration	bats	loudness	puls rate	alpha	y	name applicat...	energy	Cooust	Date
noc 2D without ...	2	4	/	250	10	1.1	0.5	0.1	5.0	pip.xml	243.2	640.0	2022-06-18

Figure 4-11: interface d'historique.

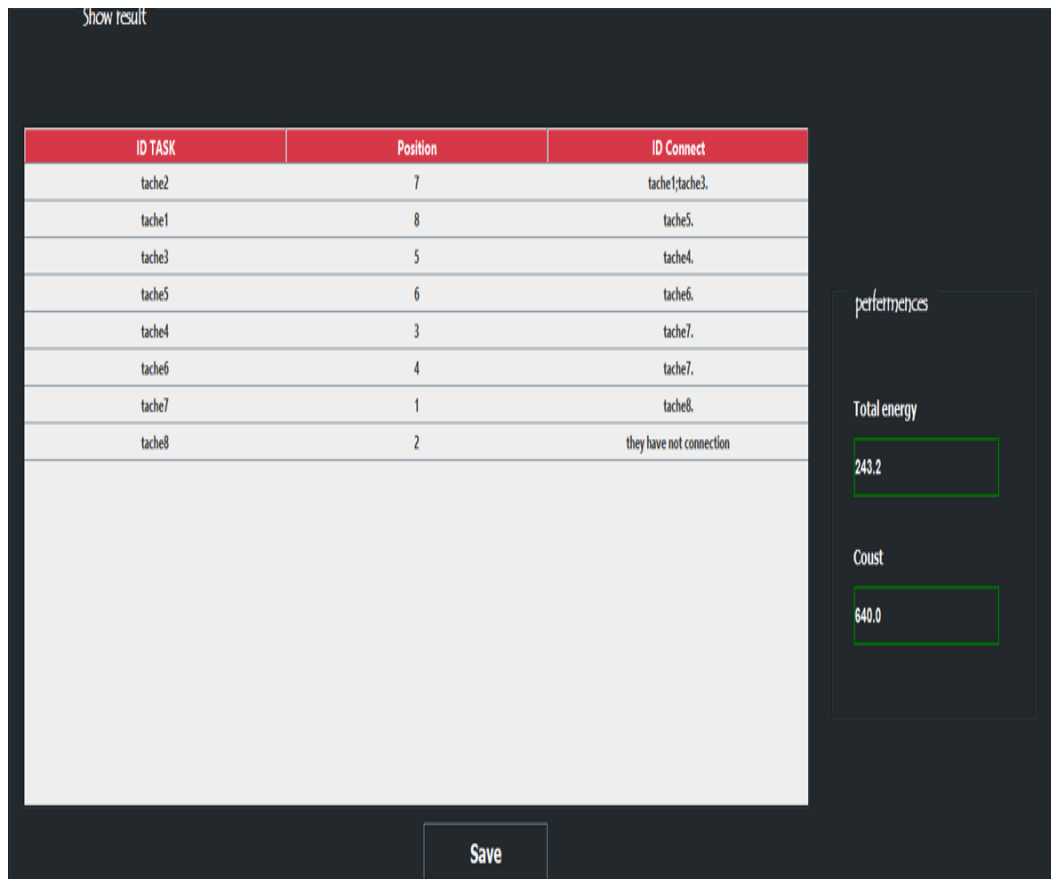


Figure 4-12:Affichage Résultat.

4.6 Tests et résultats

Nos tests sont effectués en trois parties comme suit :

4.6.1 Etude paramétrique

La performance de l'algorithme proposé est influencée par un certain nombre de paramètres, tel que la fréquence minimale et maximale, la pulsion r , le volume A , la taille de population et le nombre des itérations.

À titre de simplification, nous avons choisi la pulsion r et le volume A selon le littérateur 66 comme suit : $r_i = A_i = 0.25, \forall i=1, n$. et la fréquence f a été choisie dans cet intervalle $[0,500]$.

Nous avons effectué des tests sur les benchmarks GT1 (Générer aléatoire par TGFF), MDW, VODP et PIP afin de tester les effets de la variation du nombre des itérations et du nombre de chauves-souris dans une population initiale sur les résultats, et choisir après les paramètres à utiliser.

Nous avons fait plusieurs tests. En variant sur les valeurs de nombres d'itérations et nombres de chauves-souris sur chaque benchmark, l'architecture utilisée dans cette section est 2D simple avec des tailles différentes selon la taille de benchmarks.

Le tableau ci-dessous résume les principaux tests :

Discussion

Après la réalisation de tous ces tests nous remarquons que notre technique est influencée par la taille de la population et du nombre d'itérations, ainsi que la taille de benchmark choisi.

Plus la taille du benchmark augmente plus le nombre des itérations et la taille de population augmente, Par exemple dans le cas de benchmarks PIP une valeur des itérations 50 et nombres des chauves-souris 10 seulement suffit pour trouver des bons résultats, alors que pour les benchmarks avec des grandes tailles, il faut maximiser ces valeurs, d'où les meilleures solutions est atteignable au bout des itération entre 400 et 1600, et entre 250 et 600 pour la taille de population.

Tableau 4:Résultat des tests de Population & Nombre itération.

Benchmarks	Nombre d'itération	Nombre de bat	Coût de communication
PIP	50	5	768
		10	640
	100	15	640
		20	640
		50	640
MDW	200	200	1408
		250	1312
	1000	300	1344
		400	1312
		450	1312
VODP	400	300	4916
	1000	300	4688
	1600	1000	4497
		1400	4345
	1600	1600	4431
GT6	1000	500	284700
		200	287300
	1500	500	274800
		600	279900
	1600	600	271700

Ce Tableau résume les résultats obtenus :

Tableau 5:Résultat d'étude Paramétrique.

Benchmarks	Nombre d'itération	Taille Population
PIP	50	10
MWD	400	250
VODP	1600	1400
GT6	1600	600

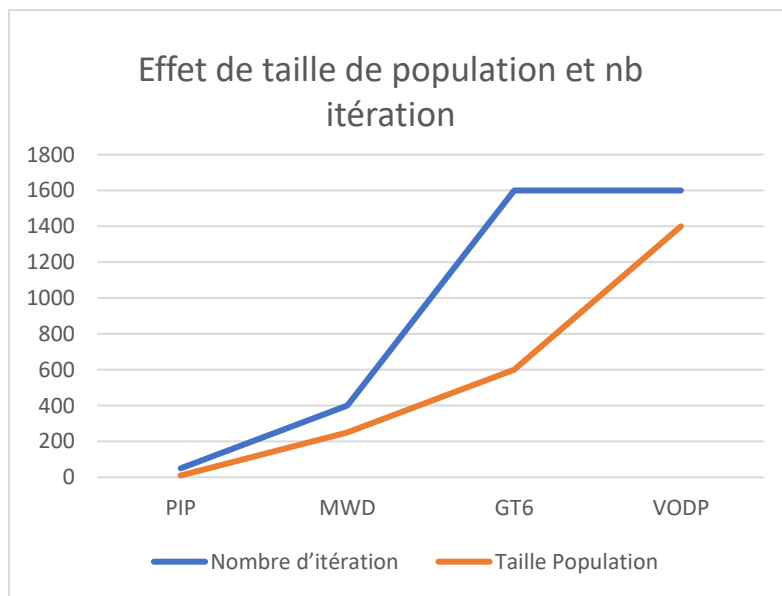


Figure 4-13: Histogramme représente la variance de paramètres entre les benchmarks

Nous constatons que le nombre des itérations et la taille de population influence d'une manière directe sur le résultat, dont un bon choix peut maximise la performance de mapping, et vice verra un mauvais choix peut réduire considérablement les performances globales du système final.

4.6.2 Etude comparative entre 2D, 2D wifi et 3D

Dans notre travail nous avons utilisé trois topologie 2D MeSH simple ,2D wifi et 3D avec des liens TSV. Afin d'étudier l'effet de choix de topologie et sa taille, nous avons réalisé des tests sur des benchmarks différent.

Les tableaux ci-dessous contiennent les meilleurs résultats obtenus tout en variant sur la topologie Noc et sa taille.

- **Test 1**

Les tests de cette section sont effectués sur le benchmark PIP

Tableau 6:Résultat de Test1

Topologies	Coût de communication
2D (2*4)	640
2D wifi (2*4)	640
3D (2*2*2)	640

Nous remarquons que les résultats obtenus sont égaux, n’y a pas une amélioration en changement de topologie.

- **Test 2**

Les tests de cette section sont effectués sur les benchmarks MDW, MPEG4 et GT10

Tableau 7:Résultat de Test 2

Benchmarks	MDW	MPEG4	TG10
2D (3*4)	1280	7353	183000
2D wifi (3*4)	1280	7350	198000
3D (3*2*2)	1180	7271	182000

Nous remarquons que les résultats se change d’un benchmark à un autre, le MDW ne donne pas une amélioration par rapport 2D à 2D wifi, tandis que le MPEG et TG10 marque une amélioration entre les deux architectures, Par contre pour la topologie 3D, nous remarquons que pour les trois benchmarks, les meilleurs résultats sont produits par cette dernière.

- **Test 3**

Les tests de cette section sont effectués sur le benchmark TG8

Tableau 8:Résultat de Test 3

Topologies	Coût de communication
2D (9*3)	84400
2D wifi (9*3)	84400
3D (3*3*3)	65600

• **Test 4**

Les tests de cette section sont effectués sur les benchmarks VODP et TG9

Tableau 9:Résultat test 4

Topologies	VODP	GT9
2D (4*4)	4345	318000
2D wifi (4*4)	4415	325000
3D (2*4*4)	4070	276000

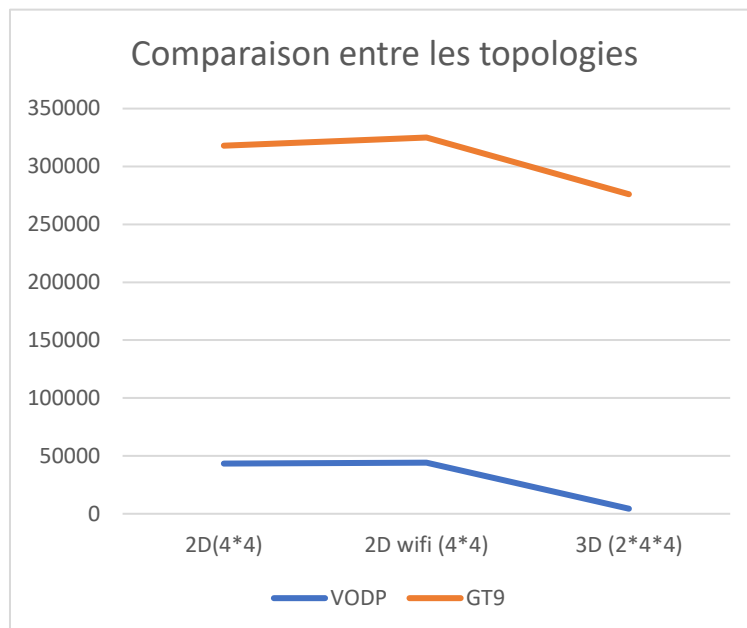


Figure 4-14:Comparaison de Test 4.

• **Test 5**

Les tests de cette section sont effectués sur le benchmark TG6

Tableau 10:Résultat Test 5

Topologies	Coût de communication
2D (8*8)	271700
2D wifi (8*8)	281900
3D (4*4*4)	201700

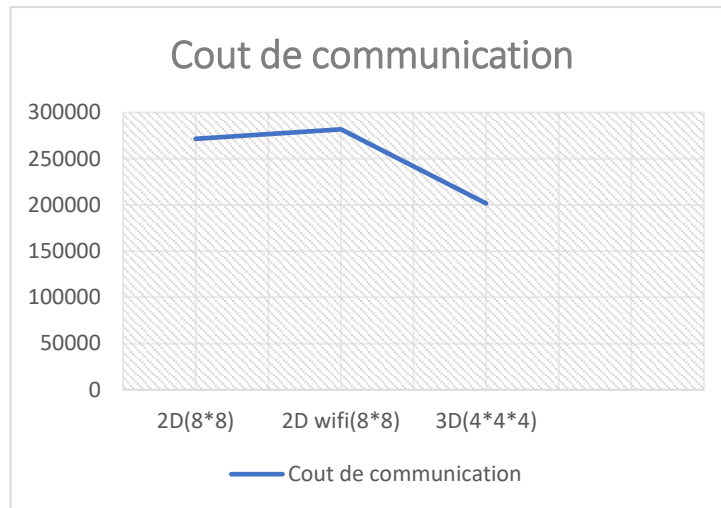


Figure 4-15: Représentation graphique de Test 5.

Pour les tests 3, 4 ,5 nous remarquons quand nous varient entre les architectures, les résultats sont améliorés, et les meilleurs résultats sont celle de l'architecture 3D.

Ces résultats permettent de déduire que l'utilisation des technologie 2D wifi et 3D prouvent son efficacité par rapport 2D simple dans le cas des applications de grande taille qui nécessite une architecture plus de 4*4, mais dans le cas d'une petite application par exemple PIP, qui besoin une architecture 2*4, un mapping sur une architecture 2D est suffisante pour avoir des bons résultats.

4.6.3 La comparaison avec la littérature

Après avoir effectué une étude comparative entre les topologies 2D, 2D wifi et 3D, nous comparons maintenant nos résultats obtenus par BAT avec la techniques ABC (Artificiel Bee Colony), GABC (Genetic- Artificiel Bee Colony) qu'ont présentés dans le projet de fin d'étude (MESSAOUDI,2015), et la technique BFO (Bacterial Foraging Optimization) basée sur l'optimisation de butinage dans le monde bactérien présenter par (MESSAOUADI, 2019), Une autre comparaison a été également effectué avec la littérature (Dageleh & Jamali, 2018)

Notre étude a été effectuée comme suit :

- ✓ Comparaison Bat avec GABC sur 2D.
- ✓ Comparaison Bat avec ABC sur 2D.
- ✓ Comparaison Bat avec BFO sur 2D
- ✓ Comparaison Bat avec GABC sur 3D.
- ✓ Comparaison Bat avec ABC sur 3D.
- ✓ Comparaison Bat avec BFO sur 3D.
- ✓ Comparaison avec des algorithmes publiés.

Nous allons présenter et discuter les résultats dans les sections suivantes.

❖ **Comparaison Bat avec GABC sur 2D :**

Tableau 11: Comparaison entre Bat et GABC sur 2D.

Benchmark	BAT	GABC
GT6	271700	176300
GT7	109400	70700
GT8	89000	60100
GT9	43600	25200
GT10	30300	21000

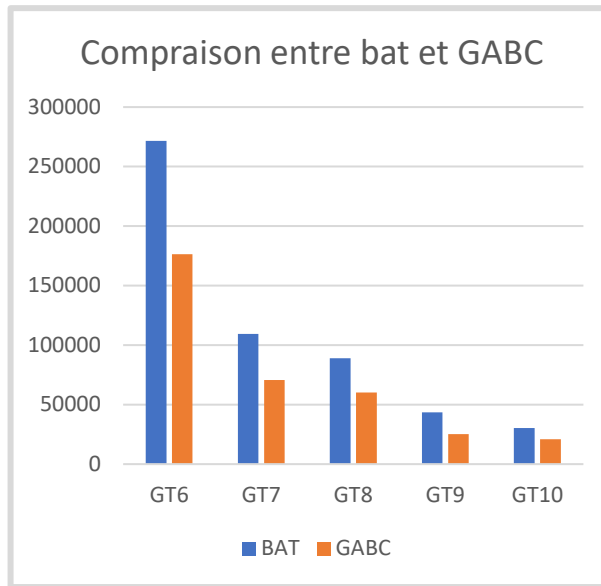


Figure 4-16: Comparaison Bat avec GABC sur 2D

D'après le graphique, nous remarquons que notre technique Bat ne donne pas de bons résultats par rapport à la technique GABC, en particulier dans les grands benchmarks, mais nous pouvons également remarquer que dans les petits benchmarks comme TG10 et TG9, ses résultats fournis sont proches de GABC.

❖ **Comparaison Bat avec ABC sur 2D :**

Tableau 12: Comparaison Bat avec ABC sur 2D.

Benchmark	BAT	ABC
GT6	271700	326400
GT7	109400	179900
GT8	89000	115100
GT9	43600	63400
GT10	30300	49300

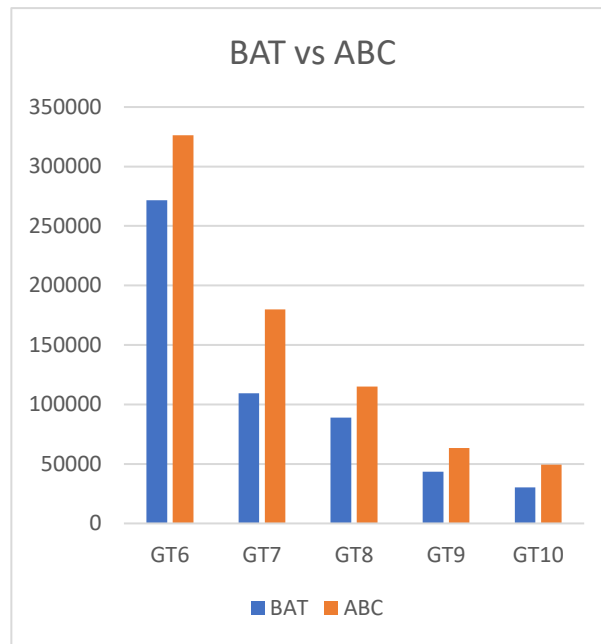


Figure 4-17: Comparaison Bat & ABC sur 2D

Nous remarquons que la technique Bat fournit de meilleurs résultats par rapport à la technique ABC dans une architecture 2D.

❖ **Comparaison Bat avec BFO sur 2D :**

Tableau 13: Comparaison Bat avec BFO sur 2D.

Benchmarks	BAT	BFO
GT6	271700	268200
GT7	109400	134900
GT8	89000	88500
GT9	43600	46700
GT10	30300	32400

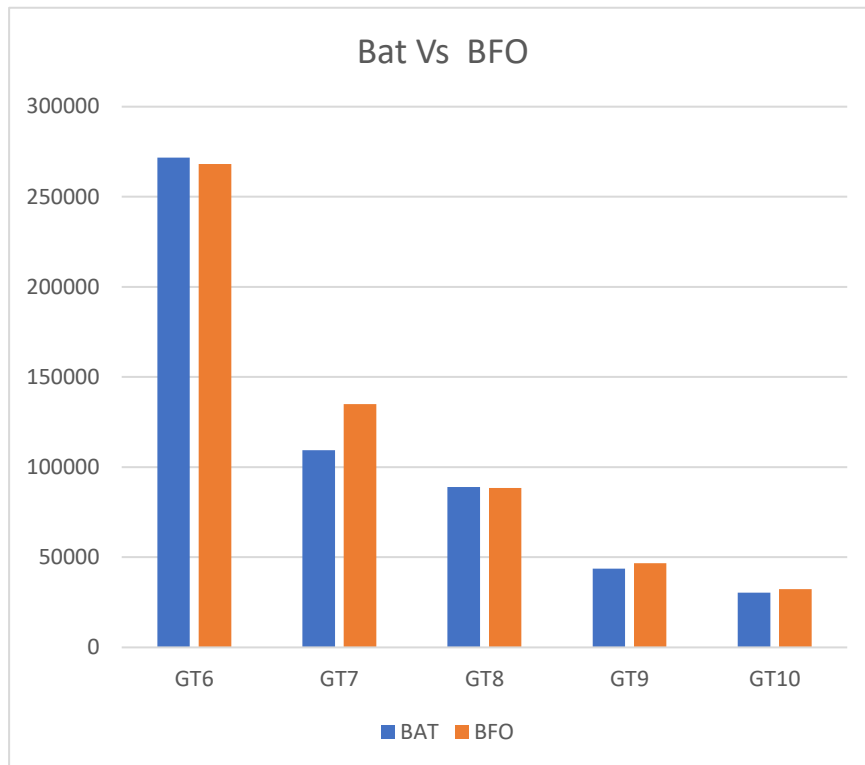


Figure 4-18: Comparaison Bat & BFO sur 2D.

Nous remarquons une variance d'amélioration des résultats entre les deux technique, le bat ne fournit pas toujours des bons résultats par rapport la technique BFO, dont il marque un meilleur retour dans le graphe GT7, et les restes graphes les résultats sont presque égaux.

❖ **Comparaison Bat avec GABC sur 3D :**

Tableau 14: Comparaison entre Bat et GABC sur 3D.

Benchmarks	BAT	GABC
GT6	201700	188200
GT7	109300	116600
GT8	61800	68100
GT9	34400	38400
GT10	18200	28600

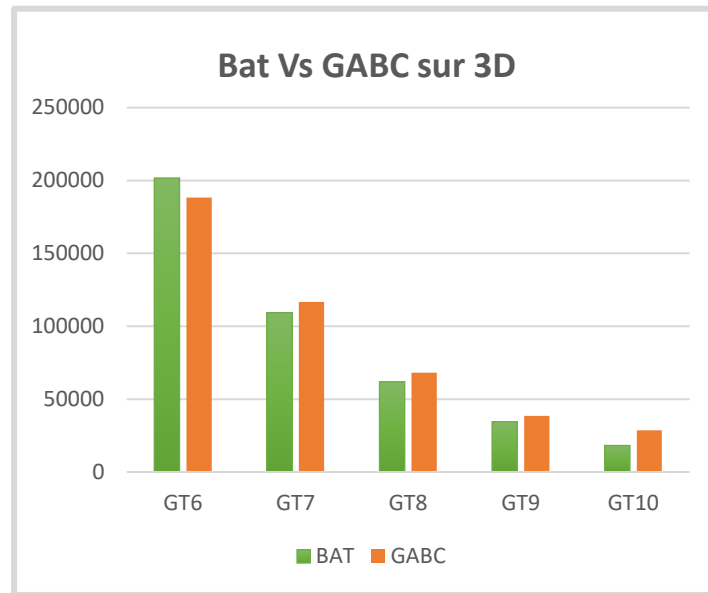


Figure 4-19: Comparaison graphique entre Bat et GABC sur 3D.

D'après le graphe, nous remarquons que le Bat améliore les résultats sur l'architecture 3D par rapport le GABC, dans la majorité des graphes sauf pour le graphe GT6, d'où la technique GABC produit un meilleur coût de communication.

❖ **Comparaison Bat avec ABC sur 3D :**

Selon le résultat au-dessous, nous remarquons également que la technique bat améliore les résultats par rapport ABC, sauf que pour le Graphe TG6.

Tableau 15: Comparaison entre Bat et ABC sur 3D.

Benchmarks	BAT	ABC
GT6	201700	199900
GT7	109300	121100
GT8	61800	73000
GT9	34400	40600
GT10	18200	31700

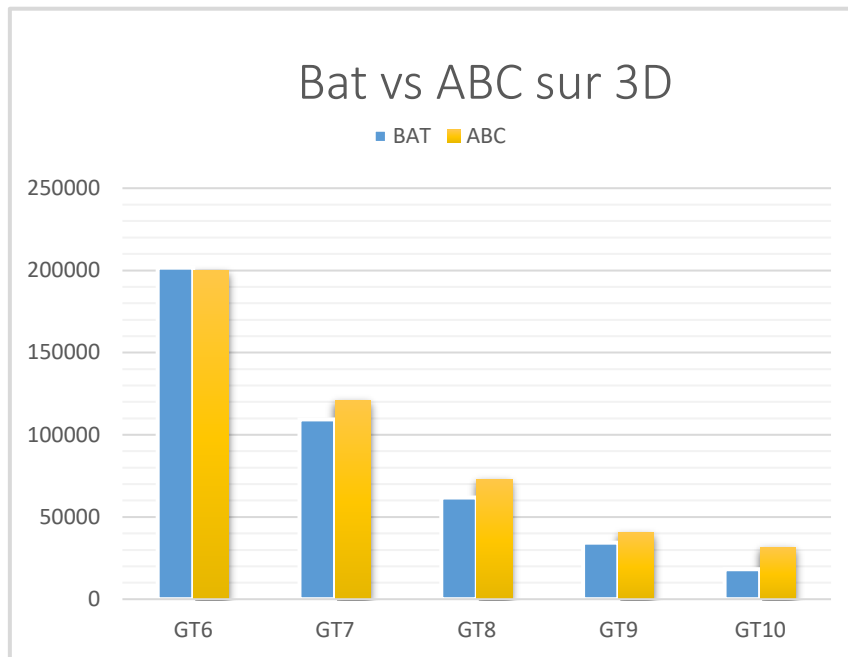


Figure 4-20: Comparaison entre Bat et ABC sur 3D.

❖ **Comparaison Bat avec BFO sur 3D.**

Tableau 16: Comparaison entre BAT et BFO sur 3D.

Benchmarks	BAT	BFO
GT6	201700	188200
GT7	109300	116600
GT8	61800	68100
GT9	34400	38400
GT10	18200	28600

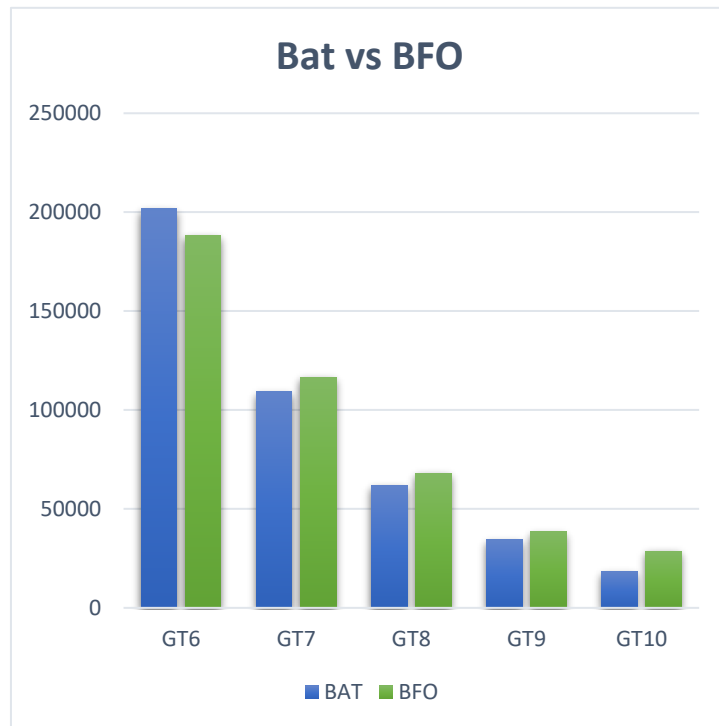


Figure 4-21: Comparaison entre Bat et BFO sur 3D.

D'après le graphique de ce test, nous remarquons aussi que le Bat fournit de bons résultats sur l'architecture 3D par rapport au BFO, dans la majorité des graphes, tandis que pour le graphique GT6, la technique BFO produit un meilleur coût de communication.

Nous pouvons constater que la méthode Bat améliore les résultats par rapport à d'autres techniques dans le cas des graphes de petite ou moyenne taille, mais dans le cas des grands graphes, elle ne fournit pas d'améliorations considérables.

❖ Comparaison avec des algorithmes publiés

Dans cette section, nous avons effectué des comparaisons entre BAT et des algorithmes publiés sur (Dageleh & Jamali, 2018), avec des architectures 3D et les benchmarks PIP, VODP, MWD.

Tableau 17: Comparaison entre résultats de Bat et les méthodes de littérature.

	VODP	PIP	MDW
NMAP3D	4119	640	1184
PSMAP3D	4119	640	1120
DPSOMAP	4119	640	1120
ILP	4103	3567	1120
TSMB	4103	640	1120
BAT	4070	640	1180

D'après les résultats de tableau, nous remarquons que le BAT, produit les mêmes résultats sur le benchmark PIP, tandis que sur VODP et MDW, n'obtient pas des bons résultats par rapport aux autres méthodes, mais il fournit des résultats approximatifs.

A la fin de notre étude et après la réalisation de tous ces tests nous avons obtenu les remarques suivantes :

Un bon choix de paramètres initiaux de l'algorithme de chauves -souris (le nombre d'itération et la taille de population) peut maximise considérablement la performance globale de Noc après le mapping.

L'utilisation de 2D wifi améliore les résultats par rapport l'architecture 2D dans le cas de graphes avec grandes tailles, et l'architecture 3D fournit les meilleurs résultats par rapport ces deux dernières topologies.

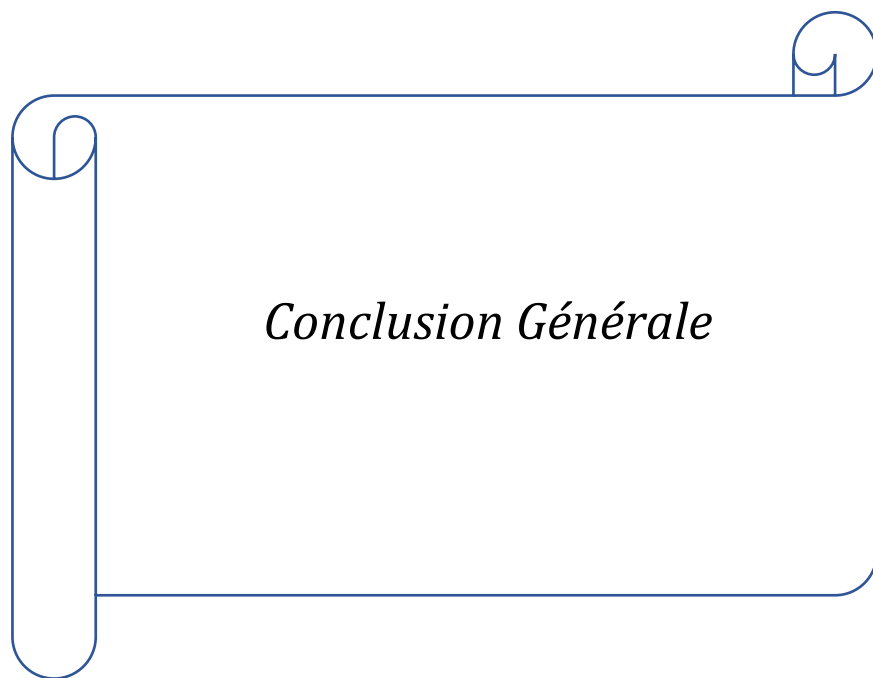
La méthode de Mapping utilisé qui se base sur l'algorithme « Bat » n'est pas une meilleure méthode mais nous prouvons prouver qu'elle est acceptable, car elle nous permet d'obtenir des bons résultats par rapport les autres algorithmes surtout dans le cas d'architecture 3D avec des petits graphes, et des résultats approchés des résultats des algorithmes standard.

4.7 Conclusion

Dans ce chapitre, nous avons d'abord présente les outils utilisés afin de réaliser notre travail, et les différents benchmarks utilisées pour les tests.

Notre but était d'optimiser le coût de communication dans la conception de trois topologie 2D MeSH, 2D wifi et 3D, dans ce but nous avons utilisé une méthode méta heuristique appelée l'algorithme des chauves-souris (Bat)

Une série des tests sont effectué pour prouver notre technique, nous avons commencé par une étude paramétrage de Bat afin de fixer les paramètres de notre technique, ensuite nous avons effectué une étude comparative entre les topologies proposées pour démontre l'effet de changement d'architecture et sa taille sur le résultat, une étude comparative a été également réalise entre notre méthode et les méthodes d'autres publiées dans la littérature.



Le mapping est l'un des problèmes rencontrés dans la phase de conception de Nos, Ce problème vise à trouver de manière optimale un placement des différentes composantes logicielles d'une application sur les nœuds de calcul de l'architecture. il se classe parmi les problèmes NP-difficile, donc L'idée d'utiliser des méthodes dites « méta-heuristique » est le plus choix convient qui donnent d'assez bon résultats en un temps raisonnable.

Le but principal de notre travail est de proposer une technique de mapping des IPs d'une application sur une structure de réseau sur puce afin de minimiser le coût de communications, l'énergie de communication et la latence. En concéderont que les architectures Noc sont homogènes.

La technique proposée se base sur l'Algorithme de chauves-souris (Bat), inspiré de comportement des chauves-souris qui utilisent un sonar biologique, appelé écholocation en raison de leur manque de capacité de vision pour identifier et localiser leurs proies, les obstacles.

Une application a été réalisée pour tester notre algorithme sur les graphes TGFF et d'autres applications réelles.

Pour atteindre notre objectif, nous avons commencé par définir dans un premier temps toutes les notions essentielles qui sont en relation avec le concept de Noc, et par la suite, nous avons décrit d'une manière générale le problème de mapping et ses types, Nous avons également présenté des travaux connexes pour mieux comprendre cette problématique. Enfin, plusieurs expérimentations sont effectuées sur des TGFF et des applications réelles retenus comme échantillons de test pour prouver l'efficacité de l'algorithme utilisé et implémenté.

Perspectives

Un nombre d'enrichissements pourra être apporté à notre travail :

- ❖ Appliquer le mapping sur une topologie hétérogène.
- ❖ Appliquer la même méta-heuristique ou autre pour faire la phase assignement.
- ❖ Développer de Topologie 3D avec TSV minimisés.

Bibliographies

- Agrawal, A., Iskander, C., & Shankar, R. (2009). Survey of Network on Chip (NoC) Architectures & Contributions. *Journal of engineering, Computing and Architecture*, 3(1), 21-27.
- Al Attar, S. (2012). *Conception et mise au point d'un procédé 3D d'assemblage de puces silicium amincies, empliées et interconnectées par des via électrique traversant latéralement les résines polymères d'enrobage*. Thèse de Doctorat, Institut National des Sciences Appliquées, Toulouse.
- Alyona, S., & Pickersgill, C. (2014). *Developing Applications with NetBeans IDE Release 8.0*.
- Avertin, S. (2012). *Développement et caractérisation de procédés de gravure plasma de TSV (Through Silicon Via) pour l'intégration tridimensionnelle de circuits intégrés*. Thèse de Doctorat, Université de Grenoble.
- BELKACEMI, D. (2020). *Mapping d'applications parallèles sur des architectures embarquées multiprocesseurs à base de réseaux sur puce*. Thèse de doctorat, MOULOUD MAMMERI, TIZI OUZOU.
- Benatchba, K., Admane, L., & Koudil, M. (2005). Using Bees to Solve a Data-Mining Problem Expressed as a Max-Sat One. *International Work-Conference on the Interplay Between Natural and Artificial Computation*, 212–220.
- Bjerregaard, T., & Mahadevan, S. (2006). A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1), 71–121.
<https://doi.org/10.1145/1132952.1132953>
- Bononi, L., Concer, N., & Grammatikakis, M. (2017). NoC Topologies Exploration based on Mapping and Simulation Models. *Euromicro Conference on Digital System Design Architectures*, 543–546.
- Bouguettaya, A. (2017). *Génération d'un réseau sur puce au format VHDL RTL à partir d'une modélisation de haut niveau UML par raffinement*. thèse de Doctorat, UNIVERSITE BADJI MOKHTAR, ANNABA.
- Bouguettaya, A., Toumi, S., & Kimour, M. T. (2016). A New 2D Mesh Routing Approach for Networks on Chip = Une Nouvelle Approche de Routage pour les Réseaux sur Puce à Topologie Mesh 2D. *Synthèse : Revue Des Sciences et de La Technologie*, 105(32), 98–105.
<https://doi.org/10.12816/0027955>
- Carvalho, E., Marcon, C., Calazans, N., & Moraes, F. (2009). Evaluation of static and dynamic task mapping algorithms in NoC-based MPSoCs. *2009*

- International Symposium on System-on-Chip - Proceedings, SoC 2009, May 2014*, 87–90. <https://doi.org/10.1109/SOCC.2009.5335672>
- Çeløk, C. (2012). Effect of Application Mapping on Network-on-Chip Performance. *Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, 22(2), 927. <https://doi.org/10.1109/PDP.2012.48>
- Chariet, A. (2014). *Approches d'optimisation et de personnalisation des réseaux sur puce(NOC:Networks on Chip)*. Thèse de doctorat, Université de Technologie de Belfort, Montbéliard (UTBM).
- Claasen, B. T. A. C. M. (2006). An Industry Perspective on Current and Future State of the Art in System-on-Chip (SoC) Technology. *Proceeding of the IEEE*, 94(6), 17. <https://doi.org/10.1109/JPROC.2006.873616>
- Cota, E., Amory, A. M., & Lubaszewski, M. (2012). *Reliability, Availability and Serviceability of Networks-on-Chip* (M. S. L. Érika Cota, Alexandre Morais Amory (ed.); 1st ed.). Springer New York, NY. <https://doi.org/10.1007/978-1-4614-0791-1>
- Dageleh, M. Z., & Jamali, M. A. J. (2018). V-CastNet3D: A novel clustering-based mapping in 3-D Network on chip. *Nano Communication Networks*, 18, 51–61. <https://doi.org/10.1016/j.nancom.2017.11.002>
- Delorme, J. (2007). *Méthodologie de modélisation et d'exploration d'architecture de réseaux sur puce appliquée aux télécommunications*. Thèse de doctart, Institut national des sciences appliquées de Rennes.
- Djaballah, N. (2011). *Une architecture d'un réseau sur puce (NoC) dédiée au routage de paquets*. Thèse de doctorat, Université El Haj Lakhdar, BATNA.
- Fehmi, M., & Chatmen. (2016). *Conception d'un réseau sur puce optimisé en latence*. thèse de doctorat, Université de Bretenge sud Monastir, Bretagne Sud.
- Gregory, T. J., Steinberg, K. P., Spragg, R., Gadek, J. E., Hyers, T. M., Longmore, W. J., Moxley, M. A., Cal, G., Hite, R. D., Smith, R. M., Hudson, L. D., Crim, C., Newton, P., Mitchell, B. R., & Gold, A. J. (1997). Bovine Surfactant Therapy for Patients with Acute Respiratory Distress Syndrome. *Respir Crit Care Med*, 155(7).
- isiaka et al. (2013). Network-on-Chip - Architecture, Optimization, and Design Explorations. In A. Oluyomi, N. J. Muga, & A. L. Teixeira (Eds.), *Network-on-Chip - Architecture, Optimization, and Design Explorations* (Vol. 15, Issue 1, p. 13). IntechOpen. <https://www.intechopen.com/books/advanced-biometric-technologies/liveness-detection-in-biometrics%0Ahttps://doi.org/10.32725/jab.2004.016>
- KAMECHE, A. H. (2013). *Approches basées sur la BBO pour le Data Clustering et le NoC Mapping*. Thèse magister, Ecole nationale Supérieure d'informatique École, Alger.
- Katto, J., & Ohta, M. (1996). Mathematical analysis of MPEG compression

- capability and its application to rate control. *IEEE International Conference on Image Processing*, 2(October 1995), 555–558. <https://doi.org/10.1109/icip.1995.537539>
- Khan, S., Anjum, S., Gulzari, U. A., & Torres, F. S. (2018). Comparative analysis of network-on-chip simulation tools. *IET Computers and Digital Techniques*, 12(1), 30–38. <https://doi.org/10.1049/iet-cdt.2017.0068>
- Lemaire, R. (2006). *Conception et modélisation d'un système de contrôle d'applications de télécommunication avec une architecture de réseau sur puce (NoC)*. thèse de doctorat, Ecole doctorale Electronique Automatique et traitement du signal, Grenoble.
- Leroy, A. (2007). *Optimizing the on-chip communication architecture of low power Systems-on-Chip in Deep Sub-Micron technology*. thèse de doctorat, Université libre, Bruxelles.
- Luis, F., & Moncayo, G. (2002). *Optimisation multiobjectif* (61st ed.). Eyrolles: Germain.
- MEHAMDIA, L., & TIGRE, A. (2018). *Mapping et Assignement multi-objectif dans les réseaux sur puce sans fil avec MultiObjectif Cat Swarm Optimisation et MultiObjectif Bat Algorithm*. Mémoire de fin d'étude master, Ecole Normale Supérieure, KOUBA-ALGER.
- MESSAOUADI, D. (2019). *Mapping dans les réseaux sur puce 3D (3D NoCs) utilisant l'algorithme des bactéries*. Mémoire fin d'étude master, ECOLE NORMALE SUPERIEURE Kouba, ALGER.
- Michael et al. (2018). Energy and performance-aware application mapping for inhomogeneous 3D networks-on-chip. *Journal of Systems Architecture*, 89, 103–117. <https://doi.org/10.1016/j.sysarc.2018.08.002>
- Murali, S., & Micheli, G. De. (2004). Bandwidth-Constrained Mapping of Cores onto NoC Architectures. *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, 2, 896–901.
- Palesi, M., & Daneshtalab, M. (2014). *Routing Algorithms in Networks-on-Chip* (M. Palesi & M. Daneshtalab (eds.); 2013950182nd ed.). Springer :New YORK. <https://doi.org/10.1007/978-1-4614-8274-1>
- Pétrot, F. (2009). *Architectures Flexibles pour la Validation et L'exploration de Réseaux-sur-Puce*. Institut National Polytechnique de Grenoble.
- Pop, R., & Kumar, S. (2004). A Survey of Techniques for Mapping and Scheduling Applications to Network on Chip Systems. *Engineering, Computing and Architecture*, 1(3), 21–27.
- Rahman, M. A. U., Ahmed, I., Rodriguez, F., & Islam, N. (2009, November). Efficient 2d mesh network on chip (noc) considering gals approach. In *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology* (pp. 841-846). IEEE
- Sahu, P. K., & Chattopadhyay, S. (2013). A survey on application mapping strategies for Network-on-Chip design. *Journal of Systems Architecture*, 59(1), 60–76. <https://doi.org/10.1016/j.sysarc.2012.10.004>

- Sahu, P. K., Manna, K., & Chattopadhyay, S. (2015). Application Mapping onto Butterfly-Fat-Tree based Network-on-Chip using Discrete Particle Swarm Optimization. *International Journal of Computer Applications*, 115(19), 13–22.
- Sappa, A. D., & Dornaika, F. (2019). An Edge-Based Approach to Motion Detection Conference. *International Conference Computatio*, 11538(May), 648–657. <https://doi.org/10.1007/978-3-030-22744-9>
- Saravankumar, U., Rangarjan, R., Haripriya, R., Nithya, K., & Rajasekar, K. (2013). Cluster Based Hierarchical Routing Algorithm for Network on Chip. *Circuits and Systems*, 04(05), 401–406. <https://doi.org/10.4236/cs.2013.45053>
- Sheibanyrd, & Pétrot. (2011). 3D integration for Noc-based Soc architector. In A. Sheibanyrad & F. Pétrot (Eds.), *Paper Knowledge . Toward a Media History of Documents* (Berlin: Sp, Vol. 7, Issue 2).
- Tatas, K., Siozios, K., Soudris, D., & Jantsch, A. (2014). Designing 2D and 3D network-on-chip architectures. In *Designing 2D and 3D Network-on-Chip Architectures* (1st ed., Vol. 9781461442). Springer New York, NY. <https://doi.org/10.1007/978-1-4614-4274-5>
- TOUATI, L. (2012). Développement d' une plateforme hétérogène de mapping dans les réseaux sur puce (NoCs) Remerciements. In *Institut National de formation en Informatique*. mémoire fin d'étude, ECOLE NATIONAL SUPERIEURE D'INFORMATIQUE, Alger.
- Toubaline, N. (2018). *AIDE A LA CONCEPTION D' UN RESEAU SUR*. Thèse de doctorat, SAAD DAHLAB , BLIDA.
- Vallerio, K. (2008). Task Graphs for Free (TGFF v3 . 0). *Official Version Released*, 1–9.
- Wang, X., Yang, M., Jiang, Y., & Liu, P. (2016). Power-Aware Mapping for Network-on-Chip Architectures under Bandwidth and Latency Constraints. *Fourth International Conference on Embedded and Multimedia Computing*, 60873112, 1–6.
- Yang, X. (2010). A New Metaheuristic Bat-Inspired Algorithm. *Nature Inspired Cooperative Strategies for Optimization*, 1–10.
- Yin, S., Hu, Y., Zhang, Z., Liu, L., & Wei, S. (2012). Hybrid Wired / Wireless On-Chip Network Design for Application-Specific SoC. *IEICE Transactions on Electronics*, E95(4), 495–505. <https://doi.org/10.1587/transele.E95.C.495>
- Zerioul, L., Bourdel, E., & Ariaudo, M. (2015, June). Modélisation et prise en compte du bruit et des interférences dans un RFNoC. In *XIXèmes Journées Nationales Microondes*.

