REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DE LAGHOUAT
ECOLE DOCTORALE SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE COMMUNICATION

N° d'ordre :

MEMOIRE

Soutenu publiquement le 14/01/2017

Pour l'obtention du diplôme de magistère

Spécialité : Informatique

Option : Informatique Répartie et Mobile (IRM)

Par

CHAIRA Mahmoud

Optimisation Multi-objectifs pour le Mapping d'Applications sur NOC

Devant le jury

Président : Pr. YAGOUBI Mohamed Bachir, Professeur à l'Université de Laghouat.

Examinateurs : Pr. CHERROUNE Hadda, Professeur à l'Université de Laghouat.

Dr. LAGRAA Nasreddine, Maître de Conférences à l'Université de Laghouat.

Directeur : Pr. BENMOHAMMED Mohamed, Professeur à l'Université de Constantine.

Remerciements

Je remercie Dieu le tout puissant, qui m'a donné la force et la patience pour l'accomplissement de ce travail.

Je tiens à exprimer mes vifs remerciements à monsieur Mohamed Benmohammed, mon encadreur pour son orientation, ses précieux conseils et ses remarques constructives qui ont largement contribué à l'aboutissement de ce travail.

Je remercie Les membres de jury d'avoir consacré de leur temps pour examiner ce travail malgré leurs nombreuses responsabilités :

1. **Pr. YAGOUBI Mohamed Bachir** Université de Laghouat.
2. **Pr. CHERROUNE Hadda** Université de Laghouat.
3. **Dr. LAGRAA Nasreddine** Université de Laghouat.

Tous mes professeurs de la première année magistère qui m'ont donné l'occasion d'améliorer mes connaissances et mon niveau d'étude.

Enfin, je tiens à remercier tous ceux qui ont contribué de près ou de loin à l'aboutissement de ce travail.

Résumé

Actuellement, Les systèmes embarqués sont de plus en plus complexes, car ils permettent d'intégrer de nombreuses fonctionnalités sur une seule et même puce. Ce qui conduit à un problème au niveau d'interconnexion, vu qu'un grand volume de données doit être traité. Une solution proposée est le concept de réseau intégrer sur silicium (abrégé en NoC signifiant Network-on-chip). Cependant, Avec la croissance d'intégration des blocs IP (Intellectual Property) dans une même puce rend le mapping d'applications sur NoC plus difficile de trouver le mapping optimale des blocs IP sur les nœuds du NoC.

L'objectif de ce mémoire consiste à proposer une technique pour le mapping des blocs IP sur les nœuds d'une architecture de type NoC. Afin de minimiser : (i) la consommation d'énergie (ii) la bande passante maximale d'un lien, sous contraintes de performance. Notre solution est basée sur l'hybridation entre une technique de clustering avec une méthode évolutive (Algorithme génétique) afin d'accélérer la convergence de ce dernier. L'étude expérimentale est réalisée sur les benchmarks E3S et trois autres benchmarks générés par l'outil TGFF afin de montrer l'efficacité de l'approche proposée.

Mots-clés : Réseau sur puce (NoC), Mapping, Algorithmes évolutionnaires, Clustering, Algorithmes génétiques, Optimisation (multi-objectif).

Abstract

Nowadays, embedded systems are becoming more and more complex as they allow to integrate many features. This leads to a problem in interconnection level, since a large volume of data must be processed. A proposed solution is the concept of integrated network on silicon (Network on chip). However, With the increasing number of IP blocks (Intellectual Property) in a single chip makes NoC applications mapping more difficult to find the optimal mapping of IP blocks onto the nodes of the NoC.

the purpose of this paper consists of proposing a technique for the placement of IP blocks onto NoC architecture. In order to minimize: (i) the energy and (ii) the maximum bandwidth of a link, under consumption performance constraints. Our solution is based on hybridization between a clustering technique and evolutionary method (genetic algorithm) to accelerate the convergence of the algorithm. The experimental study was performed on E3S benchmarks and three generated benchmarks by TGFF tool to demonstrate the effectiveness of the proposed approach.

Keywords: Network-on-Chip (NoC), Application mapping, Evolutionary Algorithms, Clustering, Genetic Algorithms, (Multi-objective) Optimization.

ملخص

الأنظمة المدمجة حاليا تزداد تعقيدا لأنه يمكن إدماج العديد من الوظائف على نفس الرقاقة. وه ذا ما يؤدي إلى مشاكل على مستوى الربط، نظرا للكمية الكبيرة من البيانات التي يجب معالجتها. من بين الحلول المقترحة نجد مفهوم الشبكة المدمجة على الرقاقة (NoC : مختصر للشبكة على رقاقة). ومع ذلك، مع تزايد إدماج وحدات أو عناصر IP(intellectual Property)على رقاقة واحدة مما يجعل عملية وضع التطبيق على الرقاقة أكثر صعوبة للعثور على الوضعية الأمثل لهذه العناصر على العقد (أو العناصر) المتواجدة في الرقاقة.

الهدف من هذه المذكرة اقتراح تقنية لكيفية وضع عناصر IP على عناصر الشبكة من نوع NoC. وذلك لهدف التقليل من: (أ) استهلاك الطاقة (ب) الحد الأقصى للنطاق الترددي للوصلة. الحل المقترح في هذه المذكرة يرتكز أساسا على عملية تهجين بين تقنية التجميع مع طريقة تطورية (الخوارزمية الجينية). الدراسة التجريبية تم إجراؤها على المعايير القياسية E3S وثلاثة معايير قياسية أخرى تم إنشاؤها بواسطة أداة TGFF لإظهار فعلية الحل المقترح.

الكلمات الرئيسية : شبكة على الرقاقة، وضع التطبيق، خوارزميات تطورية، التجميع، خوارزميات جينية، أمثلية متعددة الأهداف.

Table des matières

CHAPITRE III :

Optimisation Multi Objectif

CHAPITRE IV :

Mapping et Ordonnancement dans Les NOCs

CHAPITRE V :

Contribution

CHAPITRE VI :

Implémentation et Expérimentation

LISTE DES FIGURES

Liste des tableaux

Introduction générale

1 Cadre du travail

Les progrès croissants des technologies semi-conducteurs vont permettre l'intégration de nombreuses ressources (CPU, DSP, mémoires...etc.) dans une même puce, les besoins de communication et l'interconnexion des blocs fonctionnels au sein d'une même puce deviennent de plus en plus importants. Cependant, avec l'apparition d'applications de plus en plus complexes et gourmandes en ressources de calcul (systèmes de vision embarqués, téléphonie portable, en particulier les mobiles de quatrième génération), les structures de communications actuelles ne semble pas s'adapter aux applications futures.

Après l'année 2000, les réseaux d'interconnexion sur puce, appelées architectures de réseaux sur puce (NoC) ont été proposées comme une alternative possible aux réseaux de bus. Les NoCs ont des avantages importants comme la modularité, la scalabilité, mais extrêmement limités aux ressources. Cependant, il ya beaucoup de problèmes de recherche en cours dans le domaine de NoC. L'étape de mapping joue un rôle important pour placer les PEs[1] sur les nœuds (tuiles) dans NoC.

Le mapping d'application sur une architecture NoC est un problème NP-difficile. Depuis, une approche exhaustive est impossible, les algorithmes heuristiques sont utilisés pour résoudre ce problème.

2 Objectifs du travail

Notre travail se concentre sur l'optimisation des architectures du réseau sur puce pour une application spécifique. Il s'agit alors de proposer une technique pour le mapping des blocs IP sur les nœuds d'une architecture de type NoC. Afin de minimiser l'énergie consommée et la bande passante requise. Le travail réalisé dans ce mémoire propose une solution basée sur l'hybridation entre une technique de clustering avec une méthode évolutive (Algorithme génétique)

3 Plan de travail

Notre mémoire est organisé comme suit :

Le premier chapitre est consacré à une introduction aux systèmes embarqués, où nous présentons les motivations de ce dernier, ainsi que les exemples d'applications et les

[1] PEs : Processeurs Elémentaires

contraintes de ces systèmes, puis nous aborderons brièvement les différentes architectures des systèmes embarqués.

Le deuxième chapitre s'intéresse en particulier aux éléments constitutifs de l'architecture NoC, ainsi que les différentes techniques de commutation et routage dans NoC. .

Le troisième chapitre introduit, dans un premier temps, les concepts de l'optimisation multi-objectifs. Dans un deuxième temps, il présente les différentes classes d'approches de résolution.

Le quatrième chapitre présente un état de l'art des problèmes de mapping et l'ordonnancement dans les NOC (Network On Chip) et comment ces problème sont ils réalisés jusqu'ici.

Le cinquième chapitre présente la description détaillée de notre approche.

Le sixième chapitre décrit la partie implémentation et présente les différents résultats obtenus.

Enfin, une conclusion récapitule l'intérêt de ce mémoire et ouvre également de nouveaux éléments de réflexion et quelques perspectives de recherche.

Chapitre I

Introduction aux systèmes Embarqués

1.1 Introduction

Les systèmes électroniques et informatiques sont désormais devenus omniprésents dans le monde qui nous entoure. Souvent, un système embarqué ce n'est qu'un composant dans un système plus large. Par exemple, les véhicules modernes contiennent plusieurs systèmes embarqués. Un système de freinage antiblocage (ABS), un autre système surveille et contrôle les émissions du véhicule, et un qui affiche des informations sur le tableau de bord, etc. Les systèmes embarqués représentent une classe des systèmes informatiques dédiés, conçus à des fins spécifiques. La plupart de ces systèmes sont fiables et prévisibles.

1.2 Motivation

Il suffit de regarder autour de soi pour voir que les systèmes embarqués prennent une place de plus en plus importante dans notre société. À la maison : machine à café, lecteur DVD, console de jeux, etc. Au travail : PDA, téléphone portable, ordinateur portable, etc. La liste est encore longue des domaines de vie quotidienne sans même qu'on ne s'en rendre compte.

1.2.1 Définition

En effet, le système embarqué s'applique à différents domaines. Une définition précise est difficile. Mais quelques définitions sont tirées des articles, qui nous aident à le comprendre:

1. Un système embarqué peut être défini comme un système électronique et informatique autonome qui est dédié à une tâche bien précise. avec des contraintes plus ou moins sévères tel que la consommation, la température, la taille, les performances... etc. [1].

2. Un système embarqué est un système dans lequel le matériel et le logiciel sont intimement liés : Le logiciel est enfoui, noyé dans le matériel donc il n'est pas facile à les distinguer clairement comme dans un environnement du système de type ordinateur PC [2].

1.2.2 Exemples d'applications [3]

On trouve les systèmes embarqués dans des domaines très varier tels que :

L'électronique grand public : Les téléphones cellulaires, appareils photo numériques, caméscopes, les jeux vidéo.

Appareils électroménagers : fours à micro-ondes, répondeurs téléphoniques, thermostat, machines à laver, et les systèmes d'éclairage

Bureautique : télécopieurs, photocopieurs, imprimantes et scanners.

Automobile : contrôle de transmission, système de régulateur de vitesse, système d'injection de carburant, système de freins antiblocage.

1.3 Contraintes d'un système embarqué

Les contraintes les plus courantes auxquelles le système embarqué doit satisfaire sont les suivants:

1.3.1 Contraintes de temps

Dans les systèmes embarqués on distingue le *temps réel dur* (*hard real-time*) et le *temps réel souple* (*soft real-time*) selon le domaine d'application visé.

Temps-réel dur

Dans les systèmes temps-réel *dur*, les contraintes temporelles doivent être respectées, souvent exprimées sous la forme d'échéances (deadline), le dépassement d'une échéance a des conséquences inacceptables telles que la mise en danger de vies humaines ou des pertes financière importantes. Les applications militaires et missions spatiales sont des exemples typiques de systèmes temps réel dur [4].

Systèmes temps-réel souple

Un système temps réel souple est moins contraignant, certaines fautes temporelles (dépassements d'échéances) sont tolérables, Dans les systèmes temps-réel souple, bien que l'instant de livraison d'un résultat soit important, la violation des contraintes temporelles du système est acceptable si elle reste rare. Cette tolérance est acceptée car les applications concernées ne relèvent pas du domaine des applications critiques. Les exemples typiques d'applications ayant des contraintes temps-réel souples sont les applications multimédia telles que la téléphonie ou la vidéo [4].

1.3.2 Taille

L'espace physique requis par le système, souvent mesuré en octets pour le logiciel et portes ou transistors pour le matériel [5].

1.3.2 Consommation énergétique

La consommation d'énergie est un enjeu majeur pour de nombreux systèmes embarqués (téléphones cellulaires, ordinateurs de poche,...) dont l'alimentation en énergie est assuré par des sources électriques indépendantes comme les batteries. Puisqu'on sait bien que le fonctionnement de certains systèmes embarqués dépend de la durée de vie de la batterie, la gestion de la consommation doit être employée [4].

1.3.4 Contraintes de coût de communication

Le coût de la communication dans les nouveaux systèmes embarqués devient une contrainte très importante et plus spécialement dans celui des systèmes embarqués multiprocesseurs monopuces MPSOC. C'est le coût de communication entre les différents composants du système (l'envoi et la réception de messages), surtout dans des systèmes embarqués traitants de l'information multimédia ou de l'information de routage, et plus on avance dans ce domaine plus sera complexe l'interconnexion entre les différents composants liées à la complexité et à l'augmentation croissante des modules ou IPs[2] interconnectés [7].

1.4 Architectures des différents systèmes embarqués

Dans cette section, nous nous aborderons brièvement les différentes architectures des systèmes embarqués.

1.4.1 Mono-puce (SOC)

L'importante évolution des technologies de fabrication de circuits intégrés sur silicium en technologie CMOS a permis de réaliser des systèmes complets intégrés sur une même puce.

Un système monopuce (SoC) ou système sur puce est un circuit intégré rassemblent de nombreux composants sur une seule puce (processeurs, mémoires, bus, blocs analogiques et numériques) [8]. Les concepteurs peuvent cibler trois types de processeurs en fonction des contraintes imposés par le système, processeur spécifique à une application spécifique ASIP (Application Specific Instruction Set Processor), processeur spécifique à un domaine tel que DSP ou VSP (Digital/Vidéo Signal Processor), processeur d'ordre général tel que le processeur RISC (Reduced Instruction Set Computer) [6]. Comme illustré par la figure 1.1, Sur ces systèmes miniatures, on trouve aussi bien des cœurs de processeur, de la mémoire

[2] Intellectual Property

embarqué, des bus de communication (qui permettent d'interconnecter les différents blocs), des Convertisseurs Analogique/Numérique (CAN) et Numérique/Analogique (CNA), des contrôleurs vidéo...etc.

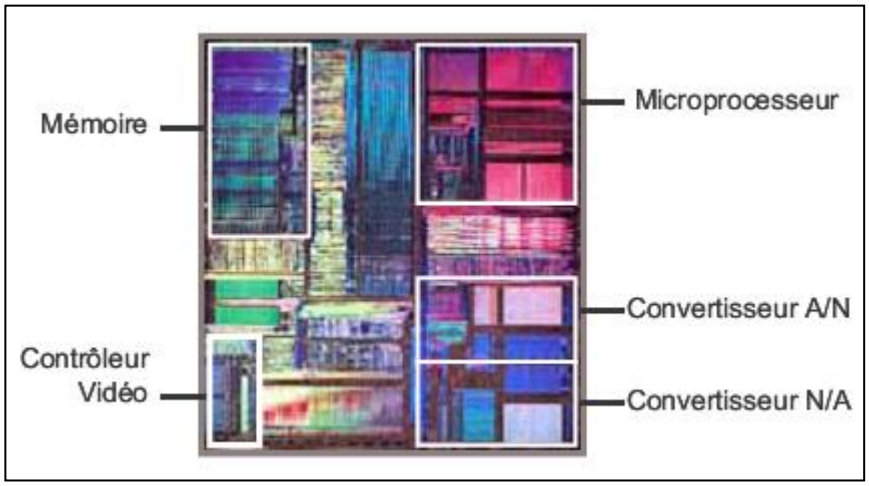


Figure 1.1 : Un exemple de système mono-puce (SOC) [9]

Un SOC est conçu toujours pour une application particulière bien définie, et résultant de la cohabitation sur silicium de nombreux composants, processeurs, convertisseurs, mémoire, bus…etc. Il doit comporter au minimum une unité de traitement de logiciel (un CPU) et ne doit dépendre d'aucun (ou de très peu) de composants externes pour exécuter sa tâche. En conséquence, il nécessite à la fois du matériel et du logiciel.

1.4.2 Les multiprocesseurs dans SOC (MPSOC)

Les systèmes multiprocesseurs sur puce (MPSOC) sont semblables aux SoCs, mais comptent plusieurs processeurs souvent spécialisés. Les MPSoCs sont généralement hétérogènes [10].

La Figure 1.2 montre une vue schématique d'une architecture MPSoC. Celle-ci est composée de plusieurs sous-systèmes interconnectés par une structure de communication interne (par exemple, un bus, un commutateur ou un Réseau sur Puce).

On peut distinguer trois types de sous systèmes : - Les sous-systèmes dédiés au calcul, tout d'abord. Ceux-ci sont constitués d'une ou plusieurs unités de calculs, de même type ou de types différents, et d'une ou plusieurs mémoires locales, de taille réduite en général.

- Les sous-systèmes de type mémoire, qui fournissent un espace mémoire partagé au sein du MPSoC, sous forme d'une mémoire embarquée ou d'une interface vers une mémoire externe.

- Les sous-systèmes périphériques, qui permettent de sortir du MPSoC, pour communiquer avec le monde extérieur, avec d'autres MPSoCs ou un matériel de stockage par exemple.

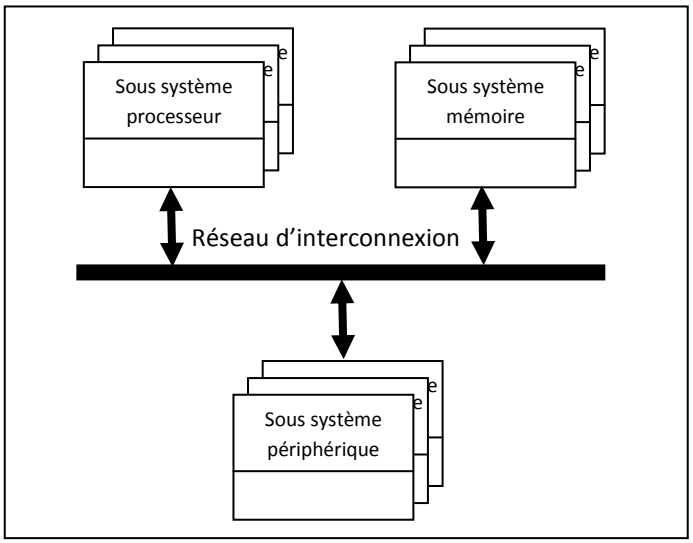


Figure 1.2 : Une architecture typique d'un réseau NOC de structure de maille [11]

ST Nomadik, le Philips Nexperia et le Diopsis D940 sont des exemples d'une architecture multiprocesseur monopuce.

L'évolution de ces architectures entraine une nécessité grandissante des besoins de communication entre les différents composants. La quantité énorme de données traités et le nombre important des IPs qui peuvent contenir un SoC, rendent la gestion de l'infrastructure de communication difficile.

1.5 Les différents types des communications sur puce

Parmi les principaux modes de communication sur puce, on peut citer :

1.5.1 Connexion point-à-point

La manière la plus simple pour communiquer les composants du système est les reliés entre eux directement. Cette connexion permet de relier directement chaque paires IPs du SoC. La connexion Point à point est simple à mettre en œuvre et efficace (un échange de données rapide). Mais elle est limitée en termes de flexibilité et scalabilité (une certaine rigidité dans le système) [12].

1.5.2 Connexion par Bus

L'architecture traditionnelle des interconnexions dans un MPSOC est le bus; il est le plus utilisé car il est inspiré des architectures monoprocesseurs. Certains changements (comme l'ajout des règles de priorités et d'arbitrage pour l'accès au bus) rendent approprié pour les systèmes multiprocesseurs. Pour les architectures en bus, la politique d'arbitrage a un impact direct sur les performances des MPSOCs [12]. L'architecture n'est pas très exigeante en coût et en surface (beaucoup moins de liens que la connexion point à point). L'inconvénient majeur du bus est la limitation en bande passante. Cependant, cette connexion se caractérise par sa faible flexibilité/scalabilité.

1.5.3 Network on Chip (NOC)

Les réseaux sur puce ont été introduit afin d'y remédier le problème de limitation en bande passante du bus qui reste plus ou moins insuffisant pour la plupart des applications complexes, et permettent d'intégrer d'avantage de nouveaux processeurs et de composants matériels. Un réseau sur puce est constitué d'un ensemble de routeurs, auxquels s'interconnectent les différents composants via l'interface réseau [13]. Les NoCs ont des performances similaires que la connexion point à point, Plus efficace que le bus avec moins de consommation d'énergie. Ils sont moins exigeants en surface que la connexion point à point et plus scalable que les deux autres techniques de communication. En conclusion, pour les NoCs nous investirons davantage dans la conception afin d'avoir un meilleur compromis coût/performances [12].

1.6 Conclusion

Dans ce chapitre, nous avons mis en évidence les systèmes embarqués. Nous avons donné quelques exemples d'applications embarquées. Nous avons cité ensuite les contraintes qu'ils imposent. Nous avons introduit le concept des systèmes sur puce, ainsi que les différentes architectures de ce type de système embarqué.

Chapitre II :

Les Réseaux sur puce

2.1 Introduction

L'évolution de la technologie de fabrication des circuits intégrés a permis aux concepteurs d'embarqués un nombre importants de composants comme les microprocesseurs, les mémoires, et les interfaces dans une même puce. Les solutions d'interconnexion classique généralement basées sur des bus partagés ne sont pas adaptées pour gérer les communications au sein d'un grand nombre de composants.

Afin de faciliter l'étape de conception et réduire les délais de communications, un nouveau paradigme de communication a été proposé par différents groupes de recherche appelé réseau sur puce (NoC) dérivé des réseaux informatiques. Le paradigme NoC permet de construire des architectures flexible et extensible de système sur puce et offre par la suite une solution performante et économique.

L'objectif de ce deuxième chapitre est de présenter les éléments constitutifs de l'architecture NoC ainsi que les différentes techniques de commutations et routages dans les NoCs.

2.2 Les paramètres de performances

La bande passante correspond au taux maximal de propagation de données dans le réseau de la source à la destination. L'unité de mesure de la bande passante est bits par seconde (bps) et il considère généralement l'ensemble du paquet, y compris les bits de l'en-tête, la charge utile et la file d'attente.

La latence (délai de transit, ou retard) est le temps écoulé entre le début de la transmission d'une donnée à travers le réseau et le moment de réception de celle-ci par le nœud cible. La latence est généralement mesurée en milliseconde.

Extensibilité consiste à pouvoir augmenter le nombre d'IPs interconnectées tout en conservant le même niveau de performance.

Flexibilité est la possibilité d'interconnecté plusieurs IPs hétérogènes grâce à un interfaçage adéquat [14].

2.3 Les éléments de base d'un réseau sur puce

L'interface réseau (NI)

L'interface adapte les données issues de l'unité de traitement dans un format compatible avec les modes de transfert du réseau (et inversement). Comme l'illustre la figure 2.1. Les blocs IPs sont reliés aux unités de routage à travers l'interface réseau [15].

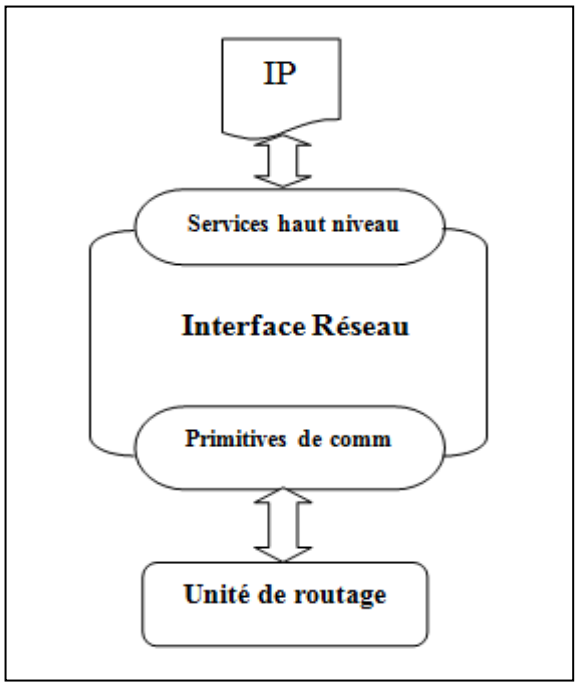


Figure 2.1 : Architecture d'une interface réseau

Les nœuds routeurs

Les routeurs sont les éléments de base de l'architecture du réseau, permettent d'aiguiller les paquets dans le réseau [15]. La figure 2.2 illustre l'architecture générique du routeur qui est constitué des éléments suivants:

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

La matrice de commutation qui connecte les ports d'entrée aux ports de sortie [16]. Un crossbar ou plusieurs multiplexeurs en cascade sont utilisés afin de multiplexer les données en entrées vers ses ports de sortie.

L'unité de routage et d'arbitrage qui définit comment doit être configurée le commutateur pour que les données contenues dans les tampons d'entrée soient correctement aiguillées. De

plus, il doit gérer les conflits d'accès lorsque plusieurs files d'attente d'entrée doivent être connectées à la même file de sortie.

Les liens permettent de relier les différents nœuds entre eux. Ils peuvent constitués d'un ou plusieurs canaux logique ou physique [17].

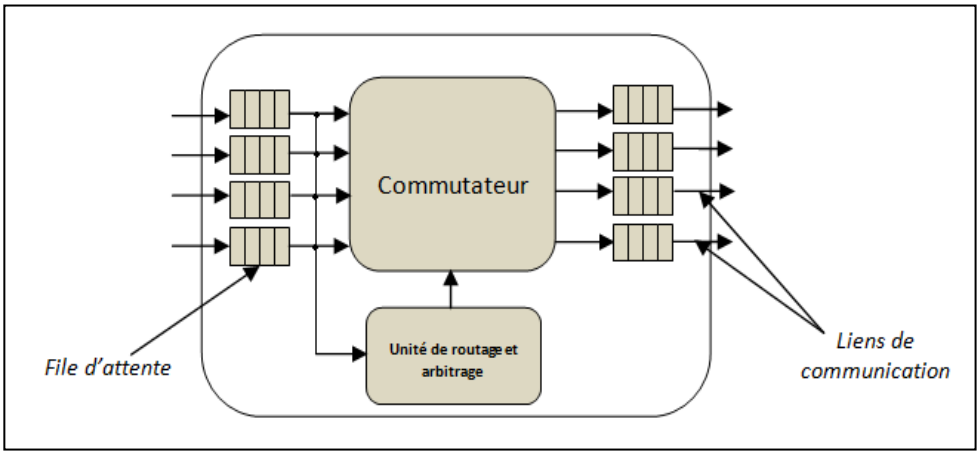


Figure 2.2 : Architecture d'un nœud routeur

2.4 Les Topologies des réseaux sur puce

La topologie d'un réseau définit l'organisation des routeurs dans le réseau. La topologie dans NoC peut être vu comme une carte routière où les liens (similaire aux routes) qui transportent des paquets (Véhicules) d'un nœud routeur (passage) à un autre [18].

Topologie de maillage à deux dimensions

Cette topologie est la plus souvent utilisée représentée sur la figure 2.3. Chaque routeur est lié aux quatre routeurs voisins et à une unité de traitement, à l'exception de ceux situés aux frontières extérieures [18]. Elle offre plusieurs avantages tel que:

- La facilitée d'implémentation dans les technologies actuelles
- La simplicité des algorithmes de routage
- La scalabilité du réseau.

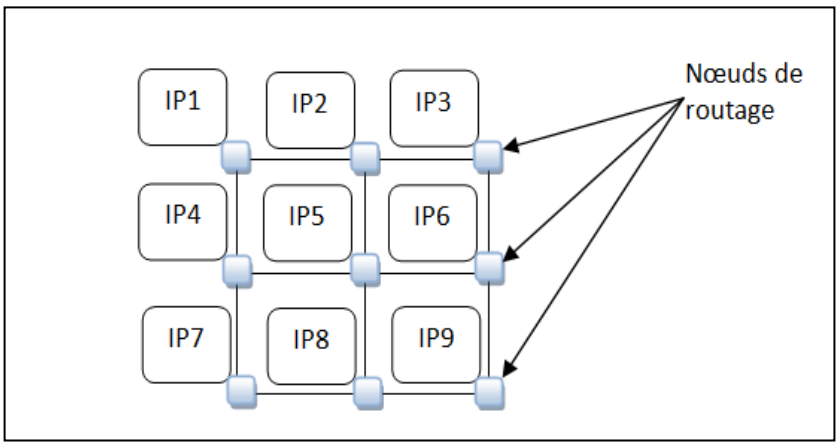


Figure 2.3 : La topologie Mesh-2D

Ce type de réseau possède une distance moyenne relativement grande et une bande passante limitée, ce qui réduit les performances lorsque le réseau est chargé [19].

La topologie Tore (Figure 2.4) C'est une extension de la structure 2D est utilisé dans le but de réduire le diamètre du réseau et d'augmenter la bande passante mais elle est plus complexe à implémenter [20].

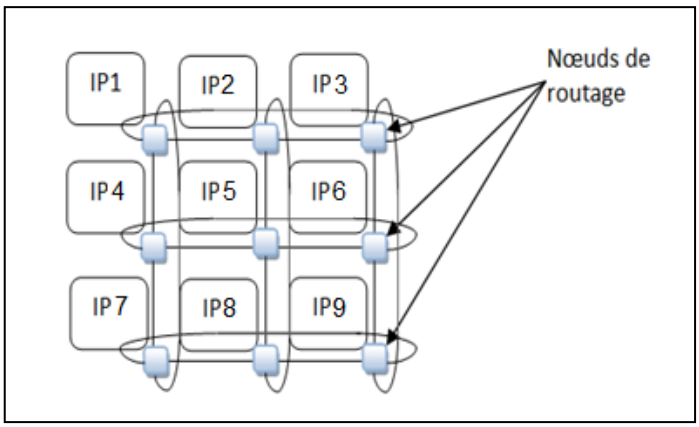


Figure 2.4 : La topologie Tore

La topologie en arbre Dans ce type de topologie le nœud au sommet est le nœud racine et les nœuds inférieurs sont les feuilles, les nœuds feuilles correspondent aux IPs et les nœuds intérieurs correspondent aux routeurs [19]. Le nœud racine peut être connecté à plusieurs nœuds de niveau inférieur, qui peuvent eux-mêmes être connectés à d'autres nœuds de niveaux inférieurs, et ainsi de suite. Cette topologie est limitée en termes de bande passante et pose des problèmes de goulots d'étranglement. Une topologie dérivée est celle de l'arbre élargi (fat tree) qui conserve la même structure que la topologie en arbre, à l'exception que les interconnexions aux niveaux supérieurs peuvent avoir plusieurs connexions entre elles, comme présenté sur la figure 2.5.

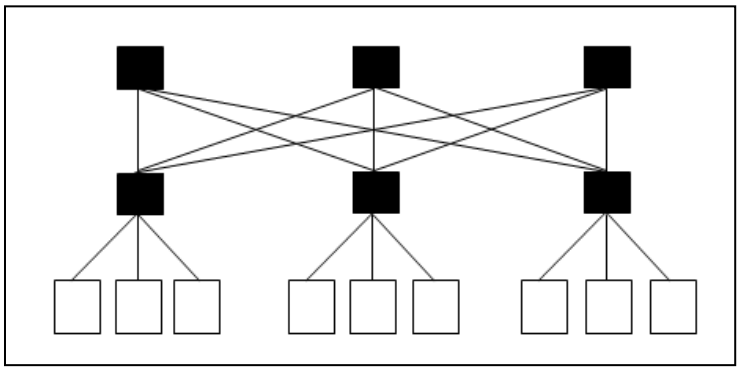


Figure 2.5 La topologie d'arbre élargi

La topologie anneau Dans une topologie en anneau, tous les nœuds sont connectés en formant une boucle (Figure 2.6). Chaque routeur est relié à leurs deux voisins quelle que soit la taille de l'anneau [19]. Cette topologie est facile à mettre en œuvre au niveau des algorithmes de routage. Cependant, elle n'est pas extensible car ses performances se dégradent (la bande passante devient limitée) lorsque le nombre d'IPs connectés augmente.

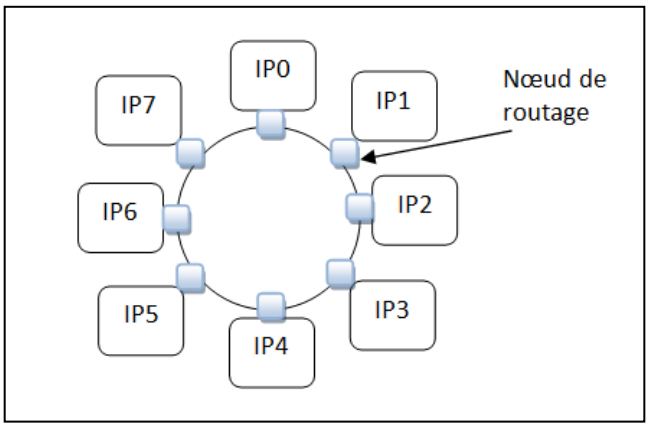


Figure 2.6 : La typologie anneau

2.5 Les techniques de commutations

Les techniques de commutation définissent la manière dont les flux de données sont acheminés à travers le réseau. Les deux techniques de commutation principalement utilisées par les réseaux sur puce sont :

La commutation de circuit

Dans cette technique, un chemin entre la source et la destination est établi avant la transmission de données [21]. Le principal avantage de cette technique est le temps de latence réduit entre la source et la destination. Lorsque le chemin est établi, la possibilité de perte des

paquets est faible. La technique de commutation de circuit ne s'adapte pas avec la taille du NoC, les éléments du réseau (nœud, lien) sont alloués pour le transfert de données et ils ne permettent pas d'être utilisés pour d'autres communications tant que la connexion est active.

La commutation de paquet

La commutation de paquet est la technique la plus couramment utilisée dans NoC. Dans cette technique, les paquets sont routés individuellement de sa source à sa destination à travers différents routeurs, ce qui introduit des délais plus importants comparé à la technique de commutation de circuit [21].

Il existe trois principaux modes de commutation de paquets :

- Store and Forward
- Virtual Cut-through
- Wormhole

Store and Forward

Dans ce mode de commutation, les nœuds de routage vont stocker totalement les paquets avant de les renvoyer, comme le montre la figure 2.7. Cela implique que chaque routeur doit être en mesure de stocker la totalité d'un paquet de données [22].

Figure 2.7 : Commutation Store and Forward

Comme les ressources de mémorisation sont très coûteuses en termes de surface et de consommation, la taille de paquet est donc limitée. De plus, le délai des paquets augmente à chaque routeur car tous les flits du paquet doivent être reçus par le routeur actuel avant d'être envoyé au suivant. La latence totale est donc la multiplication du temps de transfert d'un

paquet entre deux routeurs par le nombre de routeurs du réseau traversés par le paquet de sa source à sa destination.

Virtual Cut-through

Ce mode de commutation [22] a été proposé afin de réduire la latence des paquets à chaque étape de routage. Dans ce mode de commutation, un paquet peut commencer à être transféré au routeur suivant avant la réception complète de ce paquet par le routeur actuel, comme illustré dans la figure 2.8.

Au lieu d'attendre que tout le paquet arrive dans le routeur et avant d'examiner l'en-tête pour déterminer les paramètres de routage. Celui-ci est examinés dès son arrivé dans le routeur. Du coup, il ne faudrait même pas stocker le paquet qui peut être directement transmis au routeur suivant s'il garantit le stockage du paquet dans sa totalité. Le routeur doit pouvoir stocker le paquet (dans sa totalité) si le routeur suivant n'est pas disponible. La capacité de mémorisation du routeur est donc la même que dans SAF mais la latence est diminuée, en absence de contention, car il n'est plus nécessaire d'attendre la réception complète du paquet pour passer d'un routeur à l'autre.

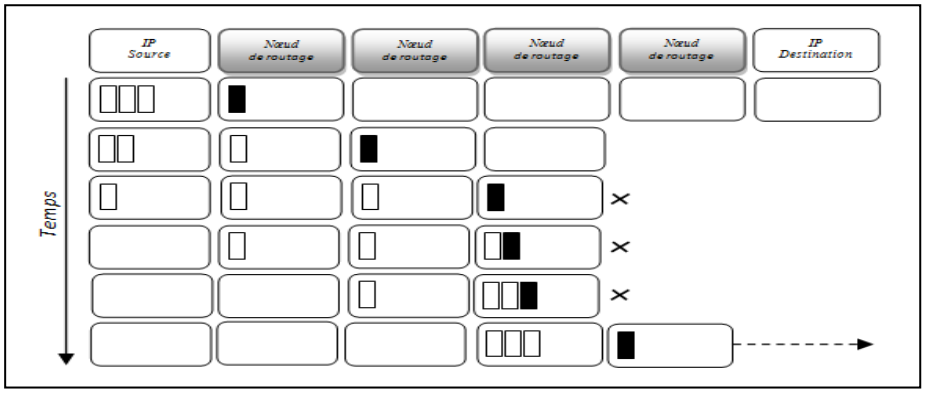


Figure 2.8 : Commutation virtual cut through (VCT)

Wormhole

Le mode de commutation wormhole [23] ou trou de ver est basé sur le principe que les flits d'un paquet peuvent être transmis d'un routeur à l'autre dès qu'il y a de la place pour un flit et plus nécessairement pour un paquet complet (figure 2.9).

Plus précisément dans ce mode de commutation, on considère qu'un paquet est composé de flits et que seul petit nombre de flits peut être stocké dans chaque routeur.

L'en-tête du paquet (i.e., le flit de tête) contient la destination. Les paquets progressent flit par flit dans le réseau. Dès que l'en-tête d'un paquet arrive dans un routeur, ce dernier le traite et détermine le canal de sortie que doit emprunter le paquet. Si ce canal est disponible, le flit d'en-tête est transmis directement sur ce dernier, le reste des flits le suivent à la queue.

Les principaux avantages du mode de commutation de trou de ver sont d'avoir une faible latence en moyenne et surtout de permettre d'utiliser des files d'attente de faible taille. Cette dernière propriété est la principale raison pour laquelle ce mode de commutation de paquet est utilisé. En effet la surface occupée par les points de mémorisation est une des contraintes les plus fortes des réseaux sur puce.

Cependant, il peut arriver qu'un paquet occupe plusieurs routeurs en même temps, et dans ce cas, il est possible qu'il bloque la transmission d'autres paquets.

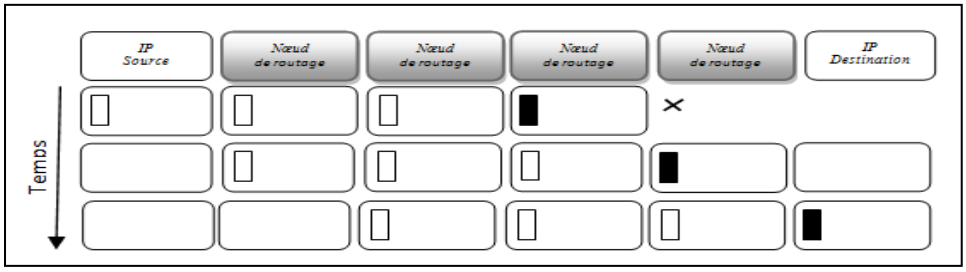


Figure 2.9 : Commutation wormhole

2.6 Algorithmes de routage

Un algorithme de routage détermine le chemin emprunté par un paquet pour atteindre sa destination, il doit être décidé au sein de chaque routeur. Il existe plusieurs types d'algorithmes de routage différenciés selon leurs caractéristiques.

Routage déterministe

Dans un algorithme de routage déterministe le chemin qui prit par un paquet d'un nœud source à un nœud destination est fixe et identique. Ce type de routage ne prend pas en compte l'état actuel du réseau. Le routage déterministe est simple à mettre en œuvre. Parmi les algorithmes les plus connus et les plus utilisés on trouve le routage appelé routage par ordre de dimension (Dimension Order Routing) [24].

Plusieurs variétés de cet algorithme ont été proposé et la plus usuelle est celle XY, est le plus simple, il ne peut fonctionner que dans une grille 2D ou un tore 2D car il fait acheminer le paquet horizontalement uniquement avant de l'acheminer verticalement lorsqu'il se trouve dans la colonne de sa destination.

Routage Adaptatif

Dans le routage adaptatif, le chemin emprunté par un paquet est déterminé en fonction de l'état actuel du réseau (la charge du réseau, la disponibilité des liens) [25]. L'idée consiste à éviter/contourner les régions du réseau fortement chargées. Le routage adaptatif est préférable pour les applications dans lesquelles le trafic est irrégulier. L'inconvénient est qu'il est complexe à mettre en œuvre et qu'il créé des conflits. Lorsqu'un paquet n'arrive pas à sa destination et tourne en rond dans le réseau.

2.7 Conclusion

Dans ce chapitre, nous avons présenté les éléments constitutifs de l'architecture NoC. Nous avons vu aussi les différentes techniques de commutation et de routage.

Le réseau sur puce est une solution émergente pour les problèmes de communications des futures SoCs, et devrait être idéale pour la complexité accrue du système.

Chapitre III :

Optimisation multi-objectif

3.1 Introduction

Le plus souvent, la plupart des problèmes du monde réel, plusieurs objectifs doivent être satisfaits simultanément afin d'obtenir la solution optimale. Cette solution n'est pas unique, mais un ensemble de solutions. Par conséquent, cet ensemble représente chacune un compromis entre les différents objectifs à optimiser et connu comme *Pareto-optimal*. Ainsi, l'optimisation multi-objectif a pour but d'optimiser plusieurs objectifs simultanément

3.2 Concepts de base concernant l'optimisation

Les concepts communs à n'importe quelle méthode d'optimisation sont les suivants :
- **Fonction objectif :** Elle est aussi appelée critère d'optimisation, fonction coût, représente le but à atteindre par le décideur [26].

- Paramètres : correspondent aux variables de la fonction objectif. Ils sont ajustés pendant le processus d'optimisation pour obtenir le(s) solution(s) optimale(s) [27].

- **Espace de recherche :** Il correspond à l'espace des solutions (ensemble de solutions). Constitué des différentes valeurs prises par les variables de décision [28].

- **Espace des objectifs :** ensemble image de l'espace de recherche, déterminé par toutes les valeurs possibles des fonctions objectif [27].

- Contraintes : spécifications du problème qui limitent les espaces des paramètres (contraintes constructives, etc.) et/ou qui interdisent une certaine bande de valeurs dans les objectifs [27].

- Domaine réalisable : la notion d'espace réalisable représentant l'ensemble des valeurs des variables satisfaisant les contraintes [5].

- Domaine non-réalisable : la notion d'espace non réalisable représentant l'espace où les contraintes ne sont pas respectées (violés) [27].

3.3 Définitions

3.3.1 Définition (Problème d'optimisation multi objectif)

Un problème d'optimisation multi-objectifs (multi-objective optimization problem, MOP) peut être défini de la manière suivante dans un contexte de minimisation :

$$PMO\begin{cases}Min\ F(x) = (f_1(x), f_2(x), \dots, f_k(x)) \\ s.c.\ x \in C\end{cases} \qquad (3.1)$$

où $k > 2$ est le nombre de fonctions objectifs, $x = (x_1,\dots, x_n)$ est le vecteur représentant les variables de décision, C représente l'ensemble des solutions réalisables associées à des contraintes d'égalité, d'inégalité et des bornes explicites et $F(x) = (f_1(x), f_2(x),\dots, f_k(x))$ est le vecteur des objectifs à minimiser [29,30].

3.3.2 Définition (Notion de dominance)

Soit une solution $x \in C$ (ensemble de solutions réalisables), *x domine* une autre solution $x' \in C$ si $\forall$ i $\in [1..k]$ $f_i(x) \leq f_i(x')$ et pour au moins un j tel que $f_j(x) < f_j(x')$. La Figure 3.1 présente les relations de dominance pour deux objectifs de minimisation. Le rectangle bleu foncé contient les solutions dominant la solution représentée par le point B. Le rectangle bleu claire entoure la région contenant les solutions dominées par B. Toutes les autres solutions ne se trouvant ni dans le rectangle bleu foncé, ni dans le rectangle bleu claire, sont indifférentes à la solution représentée par B.

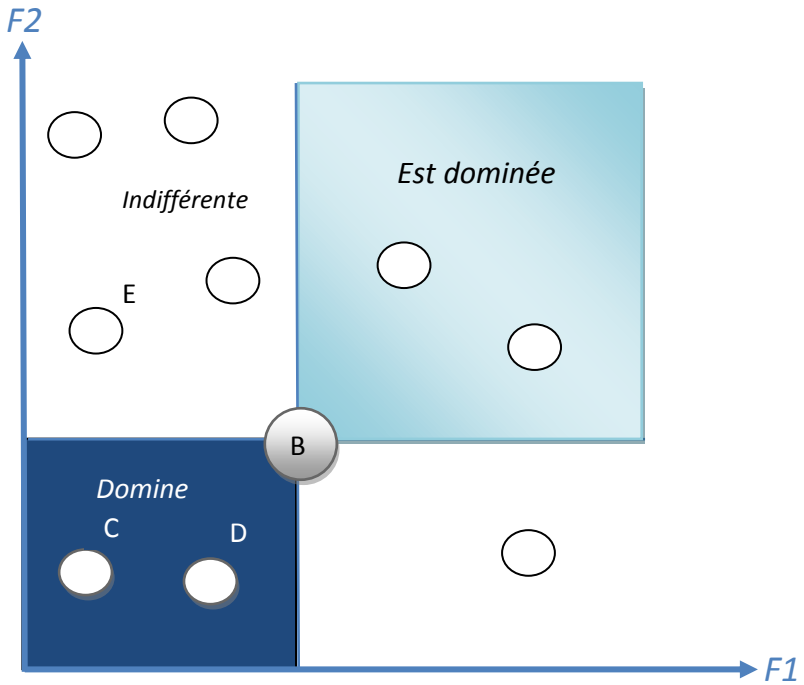


Figure 3.1 : Illustration des relations de dominance entre les solutions pour deux objectifs de minimisation [31]

Un point $x^* \in C$ est Pareto optimal si et seulement s'il n'existe aucun autre point $x \in C$ tel *que* $f_i(x) \leq f_i(x^*)$ pour tout i = 1,..., k et qu'il existe au moins un j tel *que* $f_j(x) < f_j(x^*)$. Ces points

sont également appelés solutions *non inférieures* ou *non dominées* comme illustré par la figure 3.2. L'ensemble de tous les points Pareto optimaux constituent la frontière Pareto.

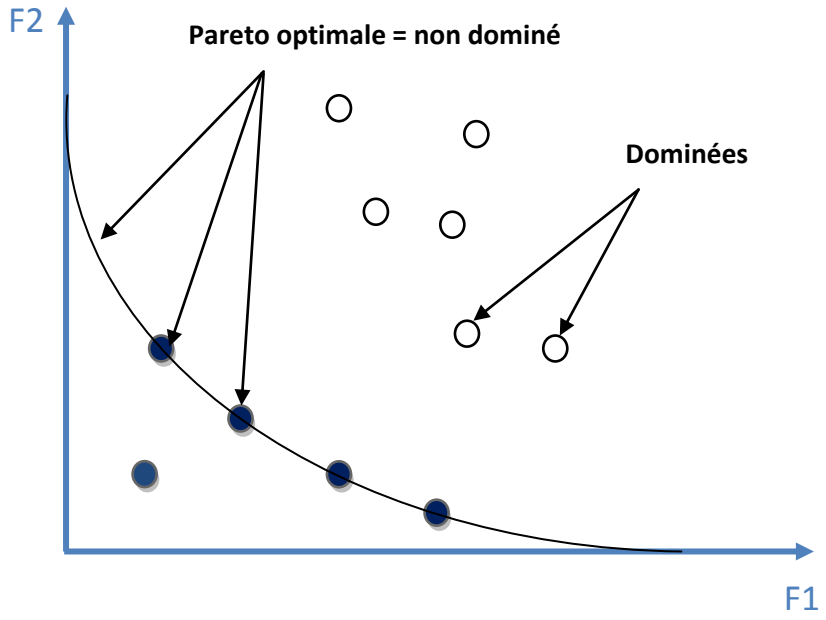


Figure 3.2 : Illustration du concept d'optimum Pareto [31]

3.4 Classification

La résolution d'un problème multi-objectif menant à la détermination d'un ensemble de solutions Pareto, d'où la nécessité de l'intervention humaine à travers un décideur pour le choix de la solution finale à implémenter.

Donc, avant de lancer la résolution d'un problème multi-objectif, la question qui se pose du moment où le décideur intervient [32,29]. Il existe trois types d'approches distinctes pour la résolution d'un problème multi-objectif

- **A priori :** le décideur intervient dans le processus d'optimisation dès le début en donnant des préférences (exemple : agrégation) à chacun des objectifs [32]. Dans cette catégorie, les méthodes utilisées pour résoudre les PMO consistent souvent à combiner les différentes fonctions objectifs en une fonction unique [31]. Cette approche permettra d'obtenir la solution recherchée en une seule exécution. Elle est donc rapide, mais il faut cependant prendre en compte le temps de modélisation du compromis et la possibilité pour le décideur de ne pas être satisfait de la solution trouvée et de relancer la recherche avec un autre compromis [9].

- **Interactive :** Dans cette méthode il y a coopération progressive entre le décideur et le système. Les processus de décision et d'optimisation sont alternés [31]. Ce dernier propose régulièrement une solution au décideur qui devra alors réorienter la recherche en fonction de ses objectifs. L'inconvénient de cette approche réside dans le fait que le décideur doit attendre entre chaque étape avant d'obtenir à nouveau la main, ce qui est parfois long [32].

- **A posteriori :** L'ensemble du front Pareto est obtenu par le décideur afin qu'il choisisse la solution la plus adaptée [31]. L'obtention du front complet peut s'avérer très longue et le nombre des solutions peut être très grand en entraînant de ce fait un problème de décision de la solution à choisir [32].

3.5 Approches de résolution

Les problèmes d'optimisation combinatoire multi-objectifs sont pour la plupart NP-difficiles. Leur taille change d'un problème à un autre, en fonction de cette taille deux classes de méthodes ont été distinguées : Les algorithmes exacts pour les petits problèmes et les heuristiques pour résoudre les problèmes de grande taille et/ou les problèmes avec plus de deux objectifs.

3.5.1 Approches exactes

Dans le cas où l'on souhaite résoudre d'une manière exacte le problème considéré, des techniques comme la programmation linéaire, programmation dynamique ou la méthode de *branch and bound* (séparation et évaluation) sont souvent utilisées. Il est utile de rappeler que les temps d'exécution de ces méthodes deviennent de plus en plus prohibitifs à mesure que les données deviennent de plus en plus grandes et le nombre d'objectifs augmente. Lorsque c'est le cas, ces problèmes font appel à des méthodes heuristiques.

3.5.2 Méthodes heuristiques

Contrairement aux méthodes exactes, les méthodes heuristiques permettent de trouver des solutions aussi bonnes que possible, pas nécessairement optimales, mais d'une qualité suffisante.

Ces méthodes peuvent être classées en trois catégories :

- **Approches basées sur la transformation du problème en un problème uni-objectif :** une classe d'approches basées sur l'agrégation qui combinent les différentes fonctions du problème en une seule fonction objectif.

- **Approche non-Pareto :** les méthodes de cette approche utilisent des opérateurs de recherche qui traitent séparément les différents objectifs.

- **Approche Pareto :** utilise directement la notion d'optimalité Pareto et la notion de non dominance dans leur processus de recherche et de sélection.

3.6 Les algorithmes d'optimisation

3.6.1 Le recuit simulé

La méthode du Recuit Simulé (RS) est une technique de recherche locale inspirée d'un processus utilisé en métallurgie. Ce processus alterne des cycles de refroidissement lent et de réchauffage ou de recuit qui tendent à minimiser l'énergie du matériau.

Le recuit simulé exploite généralement le critère défini par l'algorithme de Metropolis et *al* [33], pour une température donnée, à partir d'une configuration donnée, et on fait subir au système une modification élémentaire. Si cette perturbation a pour effet de diminuer la fonction objectif f (ou *énergie*) du système, $\Delta = f(S') - f(S) < 0$, elle est acceptée. Sinon, elle est acceptée avec la probabilité $p = \exp(-\Delta/T)$. En appliquant itérativement cette règle, on engendre une séquence de configurations qui tendent vers l'équilibre thermodynamique.

On peut systématiser l'algorithme avec le pseudo-code suivant :

Pseudo code de L'algorithme : recuit simulé

Engendrer une configuration initiale S0 de S ; S := S0

Initialiser T en fonction du schéma de refroidissement

Répéter

- Engendrer un voisin aléatoire S' de S

- Calculer $\Delta = f(S') - f(S)$

- Si CritMetropolis (Δ, T), alors S : = S'

- Mettre T à jour en fonction du schéma de refroidissement

Jusqu'à <condition fin>

Dans un premier temps, *T* étant généralement choisi très grand, beaucoup de solutions, même celles dégradant la valeur de *f*, sont acceptées, et l'algorithme équivaut à une visite aléatoire de l'espace des configurations. Mais à mesure que la température baisse, la plupart des mouvements augmentant l'énergie sont refusés, et l'algorithme se ramène à une amélioration itérative classique. A température intermédiaire, l'algorithme autorise de temps en temps des transformations qui dégradent la fonction objectif. Il laisse ainsi une chance au système de s'extraire d'un minimum local **[**34].

Le recuit simulé possède plusieurs applications dans des domaines différents : Traitement d'images, Problème d'ordonnancement, Bioinformatique, etc. Le recuit simulé donne généralement des solutions de bonnes qualités. C'est une méthode générale et facile à programmer, Souple d'emploi (de nouvelles contraintes peuvent être facilement incorporées). Cependant, les inconvénients principaux sont le nombre important de paramètres à ajuster et le temps de calcul excessif dans certaines applications.

3.6.2 Les algorithmes génétiques

Les algorithmes génétiques sont très bien adaptés aux problèmes d'optimisation de placement multi-objectif. En témoigne le nombre important d'articles qui ont été publiés sur ce sujet. De plus, ce domaine est très dynamique et ne cesse de se développer. Les algorithmes génétiques sont inspirés de la génétique classique, on considère une population des points répartis dans l'espace. Avant d'expliquer en détail le fonctionnement d'un algorithme génétique, nous allons présenter quelques mots de vocabulaire relatifs à la génétique :

Gène : un gène est la suite de symboles qui codent la valeur d'une variable. Dans le cas général, un gène correspond à un seul symbole (0 ou 1 dans le codage binaire).

Génotype ou chromosome : individu

Individu : un individu est constitué de gènes, il représente le codage d'une solution potentielle à un problème d'optimisation.

Population : un ensemble fini de taille N d'individus.

Performance : l'étape dans laquelle on mesure la qualité (fitness) de chaque individu, basé sur l'objectif de l'optimisation et ainsi de le comparer aux autres afin d'en déterminer les plus et moins aptes.

Opérateurs génétiques : Sont des opérations qui s'appliquent à partir de la population initiale et sur toutes les générations suivantes.

- **Sélection**

Cet opérateur est l'un des principaux opérateurs utilisés dans les algorithmes évolutionnaires, permet aux individus d'une population de survivre, de se reproduire ou de mourir. En règle générale, la probabilité de survie d'un individu est directement liée à son efficacité au sein de la population [35].

On trouve plusieurs méthodes de sélection différentes, par exemple :

1) La méthode de la « loterie biaisée » (roulette wheel) : Le premier type de sélection utilisé dans les algorithmes génétiques est la technique de la roue biaisée (Roulette wheel Selection RWS) proposée par Goldberg. La chance de chaque individu d'être sélectionné est proportionnelle à sa performance, donc les individus possèdent une plus grande fonction d'adaptation ayant plus de chance d'être sélectionnés [35]. La roue est divisée en plusieurs secteurs (parties) en fonction de la fitness des individus, les meilleurs individus obtiennent une plus grande partie de la roue et les mauvais individus obtiennent une petite partie de la roue. Ainsi, la probabilité d'être sélectionné est directement proportionnelle à la valeur de fitness, on fait tourner la roue N fois, où N est le nombre d'individus dans la population et quand elle cesse de tourner, l'individu correspondant au secteur désigné par le pointeur de la roue après qu'elle se soit arrêtée de tourner est sélectionné [36].

2) La sélection par tournois : Le principe de la sélection par tournoi consiste à choisir aléatoirement k individus de la population, et de choisir le meilleur individu parmi les k individus. On répète ce processus n fois de manière à obtenir les n individus qui serviront de parents. Lorsque k=2, la sélection est dite par tournoi binaire. Cette méthode est celle avec laquelle on obtient les résultats les plus satisfaisants [37].

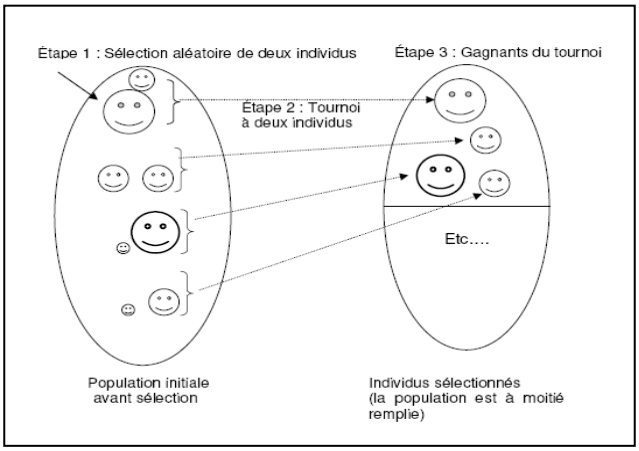


Figure 3.3 : Représentation d'une sélection par tournoi d'individus pour un critère de maximisation. Chaque individu représente une solution possible [38].

- **Croisement**

Le croisement utilisé par les algorithmes génétiques a pour but de faire varier la qualité des individus en combinant les caractéristiques désirés des deux parents, de nombreuses variantes de croisement ont été développées pour l'AG [37]. Son rôle fondamental est de permettre la recombinaison des informations présentes dans le patrimoine génétique de la population (figure 3.4).

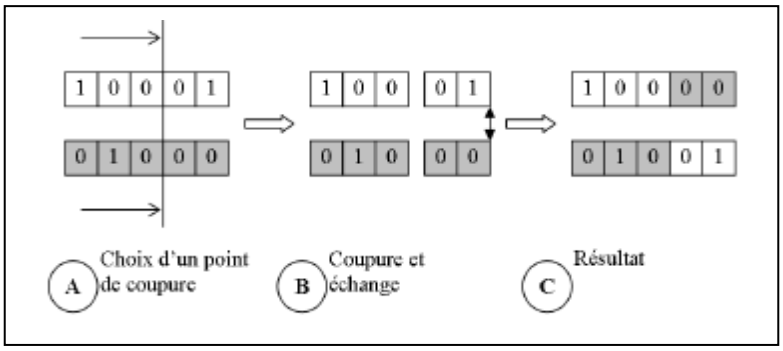


Figure 3.4 : Croisement dans un codage binaire [39]

1) Croisement SBX (Simulated Binary Crossover) : l'opérateur de croisement binaire simulé SBX proposé par Deb et Agrawal [40] est considéré comme une autre version de croisement standard à 1 point de coupure appliqué dans les chaines binaires.

Cet opérateur permet de créer deux enfants à partir de deux parents avec les relations suivantes :

$$C_1(i) = \frac{1}{2}[(1+\beta)p_1(i) + (1-\beta)p_2(i)] \qquad (3.1)$$

$$C_2(i) = \frac{1}{2}[(1-\beta)p_1(i) + (1+\beta)p_2(i)] \qquad (3.2)$$

Où β est un nombre aléatoire pouvant être obtenu en définissant un nombre aléatoire uniforme $v\epsilon[0,1]$:

$$\beta(v) = (2v)^{\frac{1}{(\eta_c+1)}} \qquad \text{si } v \leq \frac{1}{2} \qquad (3.3)$$

$$\beta(v) = \frac{1}{[2(1-v]^{\frac{1}{(\eta_c+1)}}} \qquad \text{sinon} \qquad (3.4)$$

η_c est l'indice de distribution du croisement.

2) Croisement PBX (Position based Crossover) [41] : Dans cet opérateur, les enfants sont formés à partir d'un sous-ensemble de positions issues d'un parent et les gènes correspondants à ces positions de l'autre parent seront supprimés. Les autres positions sont remplies avec les gènes restants dans le même ordre relatif que le deuxième parent. Le fonctionnement de l'opérateur PBX est illustré par la Figure 3.5. D'abord, on choisit au hasard un ensemble de positions (positions en bleu) à partir d'un parent (Parent 1 par exemple) et on les copie dans les mêmes positions dans l'enfant correspondant (Enfant 1). Ensuite, on supprime les gènes qui sont déjà choisis dans l'autre parent (Parent 2). Ainsi, le dernier parent (Parent 2) contient seulement les gènes dont l'enfant (Enfant 1) a besoin. Enfin, on met ces gènes dans les positions à compléter dans l'enfant 1, de gauche à droite, dans l'ordre dans lequel ils apparaissent dans ce parent (Parent 2).

Figure 3.5 : Opérateurs de croisement PBX [41]

3) Croisement PMX (Partially Mapped Crossover) : *PMX* a été proposé par Goldberg et Lingle [42]. L'idée de base consiste à choisir deux points de coupure au hasard. Puis, une série d'échange successif d'éléments est effectuée dans les deux parents qui est déterminée par les éléments de la section de coupure P1 et P2. Pour l'illustration, considérons l'exemple suivant [43] et supposons les deux parents suivants, figure 3.6 :

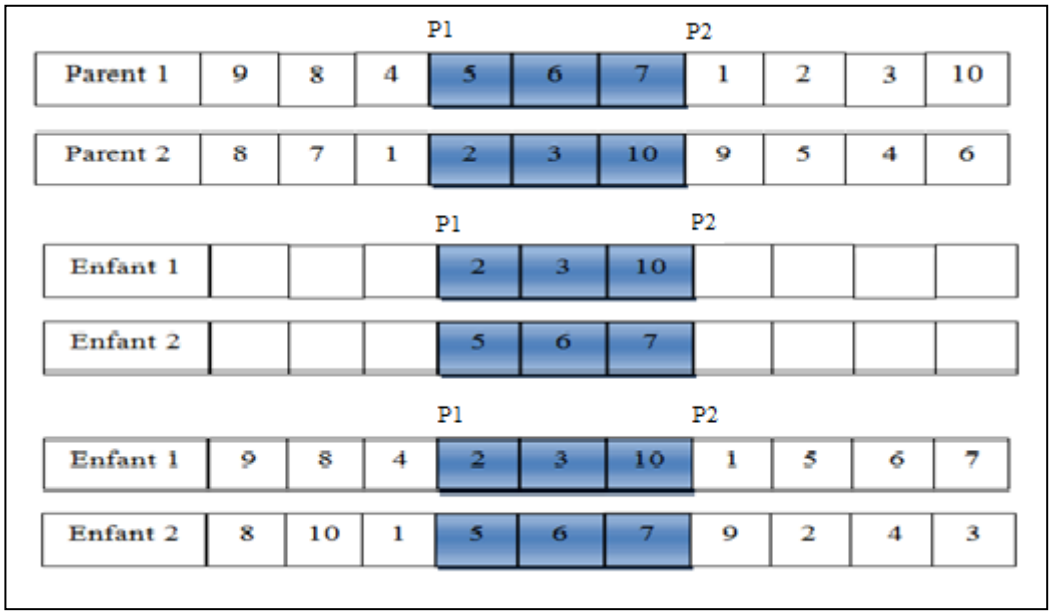


Figure 3.6 : Opérateur de croisement PMX [43]

PMX procède par des échanges de position. Au départ, les enfants 1 et 2 sont des copies conformes des parents 1 et 2, respectivement. Puis les gènes se trouvant entre les points de coupure sont échangées deux à deux, position par position (5 avec 2, 6 avec 3, et 7 avec 10).

Ensuite, pour les gènes placés avant et après les points de coupure, il reste à traiter le cas des gènes répétés (en double). Dans l'enfant 1, c'est le cas pour les gènes qui étaient placés dans la zone délimitée par les points de coupure dans le parent 2, mais en dehors de cette zone dans le parent 1. Dans ce cas, le gène placé en dehors de la zone prend la valeur du gène du parent 1 placé à la même position à l'intérieur de la zone, et par conséquent à la même position dans l'enfant 2. Par exemple, dans la figure 3.6, le gène 3 apparait à deux endroits dans l'enfant 1 : une première fois en cinquième position (entre les points de coupure) et une seconde fois en neuvième position, après le second point de coupure. La valeur 6, placée en cinquième position dans le parent 1 et le second enfant (entre les points de coupure P1 et P2) est donc utilisée pour remplacer le gène 3 à la neuvième position dans l'enfant 1. De même, les gènes 5 et 7 placés en quatrième et sixième position dans l'enfant 2 sont utilisés pour remplacer les gènes 2 et 10 au huitième et dixième positions de l'enfant 1. La même procédure est répétée jusqu'à éliminer tous les gènes apparaissant deux fois dans l'enfant 2 [43].

- **Mutation**

Une mutation consiste simplement en l'inversion d'un bit (ou de plusieurs bits, mais vu la probabilité de mutation c'est extrêmement rare) se trouvant en un locus bien particulier et lui

aussi déterminé de manière aléatoire. Cet opérateur consiste à changer la valeur d'un gène avec une probabilité très faible, généralement comprise entre 0.01 et 0.001. Ce qui permet d'introduire et de maintenir la diversité dans la population de solutions.

1) **Mutation par insertion:** L'opérateur d'insertion choisit un gène aléatoirement et l'insère dans une autre position du chromosome, figure 3.7 [43].
2) **Mutation par permutation :** L'opérateur de mutation par permutation consiste à prendre au hasard deux gènes du chromosome et à les permuter, figure 3.8 [43].
3) **Mutation par inversion :** Deux positions sont sélectionnées aléatoirement et tous les gènes situés entre ces positions sont inversés, figure 3.9 [43].

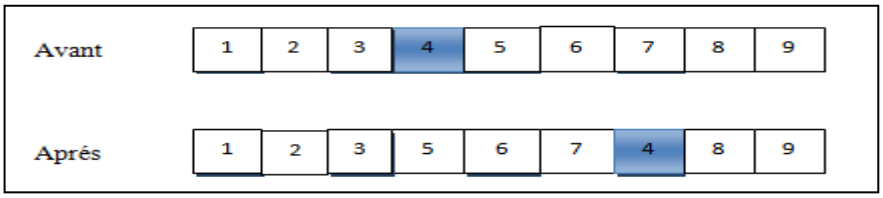


Figure 3.7 : Mutation par insertion

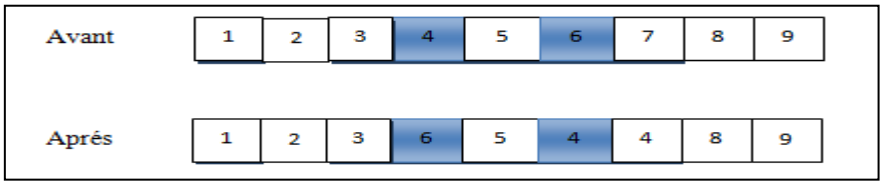


Figure 3.8 : Mutation par permutation

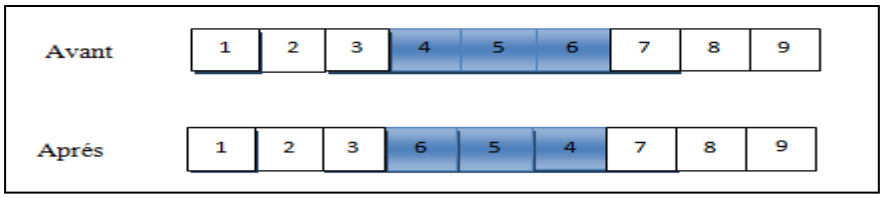


Figure 3.9 : Mutation par inversion

Finalement l'algorithme génétique peut se résumer par :

Pseudo Code de l'algorithme : AG

```
1 : Initialisation de la population
2 : Evaluation des fonctions objectif
3 : pour i =1 à MaxItér faire
4 : Sélection
5 : Croisement
6 : Mutation
7 : Evaluation des fonctions objectif
8 : Fin pour
9 : Fin
```

3.6.3 L'algorithme de Branch & Bound (séparation et évaluation):

Branch and Bound, est une méthode de résolution exacte de problèmes d'optimisation combinatoire introduite par Land et Doig [44]. Cette méthode consiste à une énumération systématique de toutes les solutions candidats, où un grand sous ensembles de candidats sont éliminés, à l'aide des procédures de bornes inférieures et supérieures [45]. Dans l'idéal, seules les solutions potentiellement bonnes sont donc énumérées.

Elle consiste à diviser le problème en sous problèmes de manière à représenter le problème sous forme d'une arborescence. L'exécution de la méthode B&B repose sur le principe de parcours en profondeur d'un arbre de l'espace des solutions. Elle commence d'abord par considérer la racine qui représente l'ensemble de solutions de départ. L'ensemble des sous problèmes (descendances de la racine) résultants de la séparation de la racine en appliquant les procédures de bornes inférieures et supérieures [30].

La méthode est ensuite appliquée récursivement à ces sous problèmes, engendrant ainsi une arborescence. Et la recherche continue jusqu'à ce que tous les nœuds soient explorés ou éliminés [29,30].

Pseudo code de l'algorithme Branch&Bound

Initialisation : Best-sol = ∞ ; Bi(p_0) := f(p_0) ; p := {(p_0,Bi(p_0))}
Tant que p ≠ 0 **faire**
Sélection : sélectionner un nœud p de P : p := p/{p} ;
Pour i := 1 à k
Évaluation $B_i(p_i)$:= f(p_i) ;
Si $B_i(p_i)$ == f(x), x réalisable et (f(x) < best-sol) **Alors**
best-sol := f(x) ;
Solution = x ;
Sinon
Si $B_i(p_i)$ > best-sol **Alors**
Élaguer p_i ;
Sinon
Séparation : Décomposer p en p_1, p_2, p_3, ...p_k ;
p := p ∪ {(p_i, $B_i(p_i)$)} ;
FinSi
FinSi
FinPour

3.7 Conclusion

Ce chapitre nous a permis de faire un tour d'horizon des problèmes d'optimisation multi-objectifs (MOP). Après avoir rappelé les principales définitions nécessaires à la présentation des problèmes d'optimisation multi-objectifs, nous avons passé en revue des principales approches de résolution.

Chapitre IV :

Mapping et Ordonnancement

dans Les NOCs

4.1 Introduction

Le problème de mapping d'applications sur une structure de réseau sur puce est un problème de placement topologique des IPs sur les nœuds (tuiles) d'une architecture NoC. Il est souvent présenté comme une variante du problème d'affectation quadratique, qui est prouvé d'être un problème NP-difficile. Où l'espace de recherche augmente de manière factorielle avec la taille du système. Dans ce cas, le routage est traité de manière implicite, il s'agit de minimiser les distances entre IPs.

4.2 Objectifs de conception des NoCs

L'ordonnancement et le mapping sont considérés parmi les étapes les plus difficiles lors de la conception d'un NoC. Cependant, un meilleur algorithme d'ordonnancement et de mapping est crucial pour obtenir des performances maximales. La consommation d'énergie, la minimisation du temps d'exécution sont les objectifs les plus souvent traités [46] par l'ordonnancement et le mapping dans les NoCs.

4.3 Problématique de mapping

Avant de faire le mapping des blocs IPs (ou PE) sur les tuiles (routeurs) d'un réseau sur puce, les tâches et des opérations de communication doivent être affectées aux ressources NoC. En outre, l'ordre d'exécution des tâches doit être établi. C'est ce qu'on appelle le problème d'ordonnancement pour les architectures NoC [47] qui est un problème NP-difficile. Il existe une influence considérable sur l'énergie consommée par ces IPs lors du calcul, en raison de leur hétérogénéité. Par exemple, un DSP peut consommer moins d'énergie qu'un processeur à usage général lors du calcul d'une transformée de Fourier rapide. En outre, l'énergie consommée par les communications est influencé par l'assignation des tâches (à cause des chemins de routage).

Par conséquent, le problème de mapping d'applications est lié aux problèmes d'ordonnancement et de routage. Comme il est montré sur La figure 4.1 :

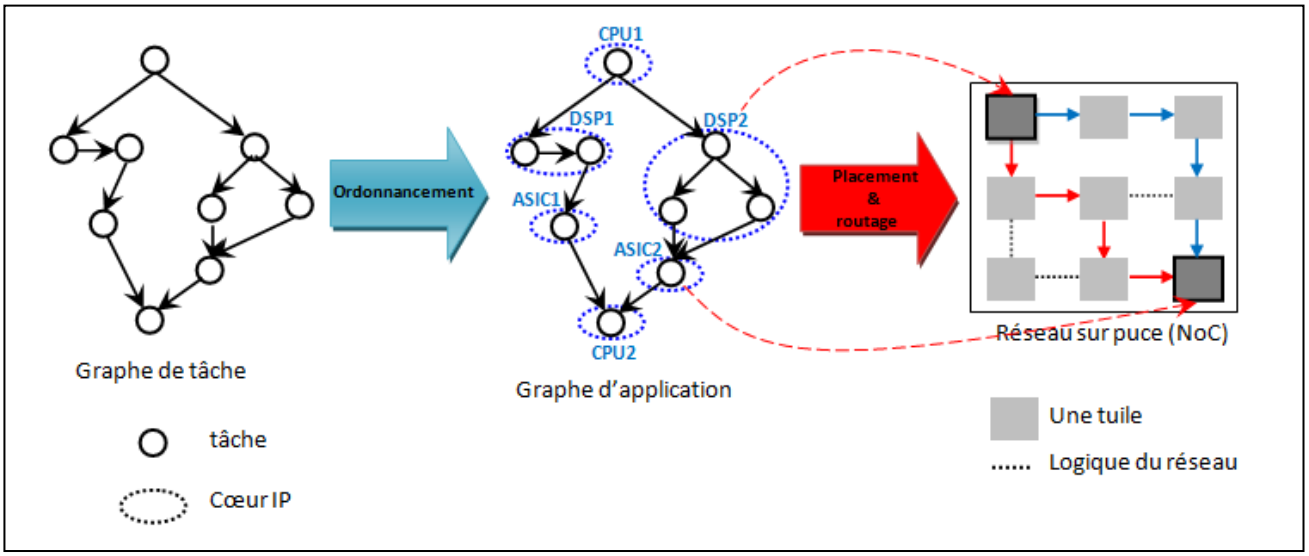


Figure 4.1 : Description du problème de mapping

Une application est décrite par un graphe de tâches. Un algorithme d'ordonnancement est ensuite utilisé pour assigner ces tâches (threads) aux IPs disponibles et préciser leur ordre d'exécution. Après l'étape d'ordonnancement, on obtient un graphe d'application, en utilisant un algorithme de mapping (ce qui peut générer la fonction de routage), ces IPs doivent être placés sur les nœuds (routeurs) du NoC.

Nous observons que les deux algorithmes ordonnancement et mapping pour les réseaux sur puce ont des objectifs similaires. L'augmentation des performances et la réduction de la consommation d'énergie.

Idéalement, les deux problèmes devraient être traités ensemble. En d'autres termes «l'ordonnancement» signifie le placement des tâches de l'application sur les IPs disponibles, et le «mapping» signifie le placement des IPs sur les nœuds du NoC disponibles. Par conséquent, à la fois les problèmes d'ordonnancement et de mapping traitent le mapping d'applications sur un réseau sur puce.

Néanmoins, en raison de la complexité du problème, le mapping d'applications sur NoC est divisé en deux étapes: l'ordonnancement, suivi par le placement (mapping).

4.4 Définitions et contraintes

Définition 1 (Modèle d'application) :

Le graphe d'application APG (Application Graph) est un graphe directe APG(C, A) où chaque vertex $c_i \in C$ représente un PE sélectionné et l'arc directe (c_i,c_j) noté $a_{ij} \in A$ représente la communication entre c_i et c_j. Deux quantités sont associées pour chaque a_{ij}

- v_{ij} : représente le volume de communication entre c_i et c_j.
- bw_{ij} : représente la bande passante entre c_i et c_j.

Un exemple d'un graphe d'application est illustré dans la figure 4.2

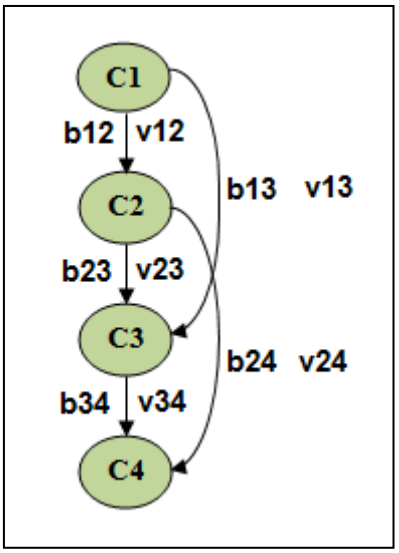


Figure 4.2 : Un graphe d'application APG (Exemple) [48]

Définition 2 (Modèle d'architecture) :

Le graphe d'architecture ARG (Architecture Graphe) est un graphe directe ARG(T, P) où chaque vertex r_i représente une tuile (un routeur) de l'architecture, et chaque arc directionnel p_{ij} représente le chemin de routage entre r_i et r_j. Les propriétés suivantes sont associées pour chaque arc p_{ij} :

- e_{ij} : le coût de l'arc entre r_i et r_j représente la consommation d'énergie moyenne pour envoyer un bit de données à partir d'un routeur r_i vers un routeur r_j.
- $L(p_{ij})$: un ensemble de liens qui forment le chemin p_{ij}.

Un exemple d'un réseau donné par un graphe d'architecture ARG est représenté sur la figure 4.3.

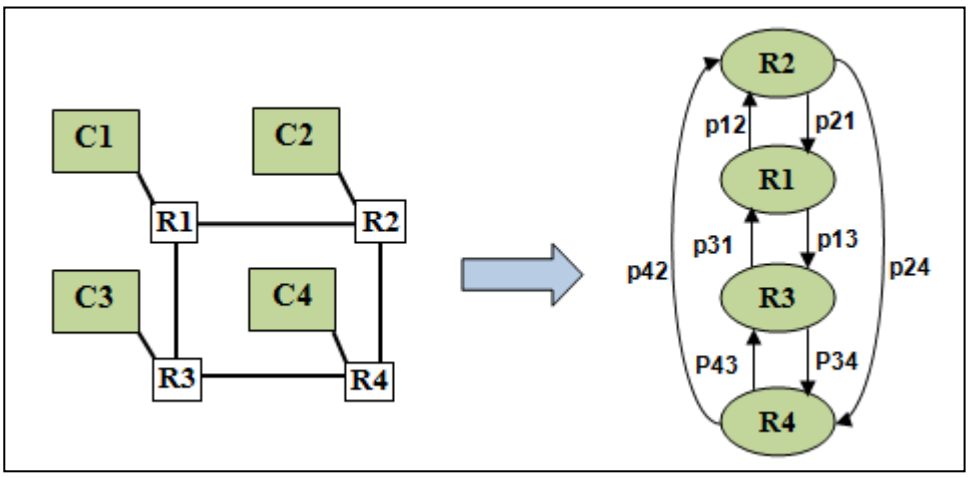


Figure 4.3 : Un graphe d'architecture ARG (Exemple) [48]

Définition 3 (Mapping)

Le mapping du graphe d'application APG(C, A) sur le graphe d'architecture ARG(T, P) est définit par la fonction de mapping suivante :

$$map : C \rightarrow T \Longrightarrow t_i = map\,(c_i), \forall c_i \in C, \exists t_i \in T, \forall c_i \neq c_j \in C, t_i \neq t_j \quad (4.1)$$

Le mapping est définit si $|C| \leq |T|$

Le but de la fonction de mapping est de trouver :

$$Min\left\{\sum_{\forall ai,j} bw_{ij} \times e_{map\ (c_i),map\ (c_j)}\right\} \quad (4.2)$$

La figure 4.4 montre un exemple de mapping d'un graphe d'application APG sur un réseau de type mesh 2D

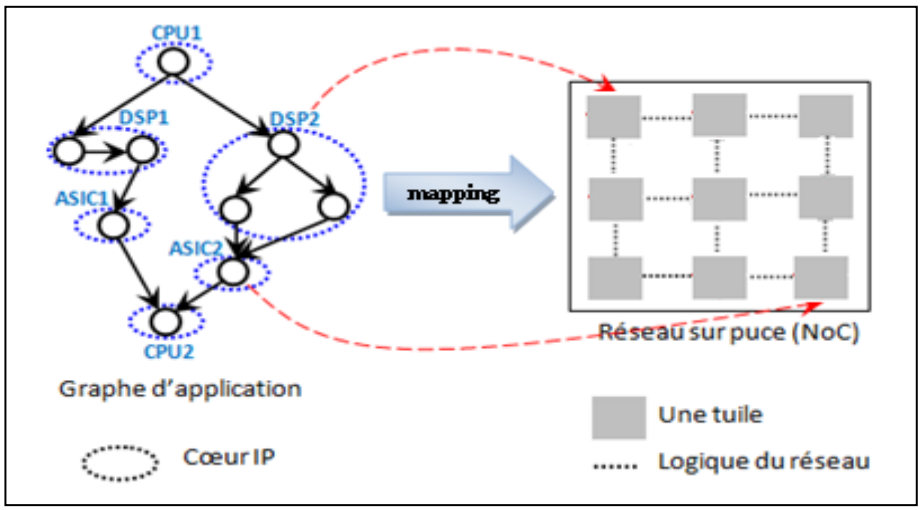


Figure 4.4 : Exemple de mapping d'un graphe d'application sur un réseau NoC

4.5 Etat de l'art des approches existantes

Dans cette section, on va présenter brièvement les approches de mapping et d'ordonnancement les plus connus dans la littérature pour les NoCs.

D.shin et Al. [49] ont proposé une technique basée sur l'allocation des vitesses des liens et l'assignation réseau afin de réduire l'énergie des communications dans NoC. Le placement des tâches permet la vérification des contraintes de dimensionnement [46]. Pour le placement des tâches et l'assignation réseau ils ont utilisé l'algorithme génétique (AG). Et pour l'ordonnancement des tâches et l'assignation des vitesses des liens ils ont utilisé une liste d'ordonnancement nommée LS (list scheduling). Pour le placement des tâches, l'algorithme génétique utilise un croisement à deux points et une mutation aléatoire. Pour l'allocation des tuiles, l'AG utilise la permutation, et le croisement cyclique pour générer des solutions légales et des permutations aléatoires comme mutation. Pour l'allocation des chemins de routage l'AG utilise des chromosomes binaires pour coder les mouvements sur les directions X et Y. Un croisement de coordonnées avec un point de croisement à l'interconnexion des chemins. L'AG lors d'allocation des chemins de routage a un impact sur le volume de communication de chaque lien et par conséquent sur les délais.

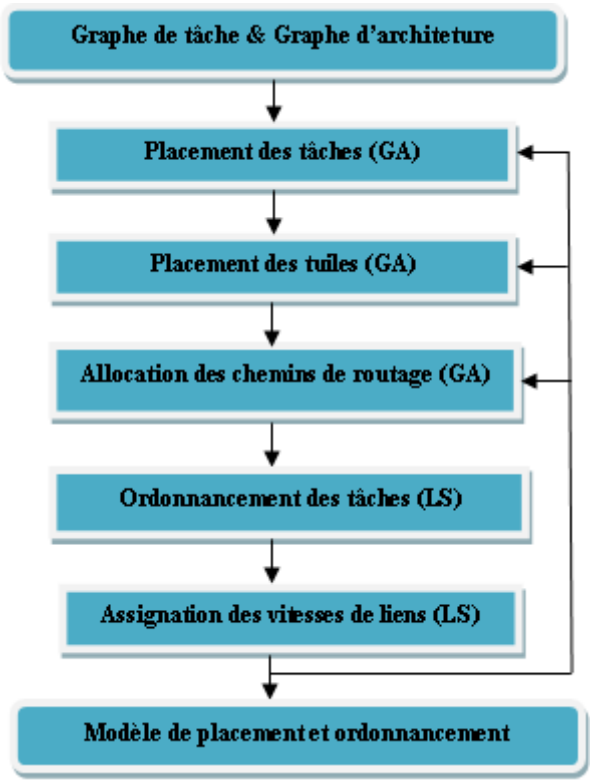


Figure 4.5 : Méthodologie de D. Shin et al

J. Hu et R. Marculescu [50] ont proposé un algorithme Branch and Bound (B&B) pour le placement des PEs sur les nœuds du réseau NoC comparé avec l'algorithme de recuit simulé. Le but étant d'optimiser l'énergie totale des communications tout en garantissant la contrainte de bande passante. Cette technique est basée sur un réseau mesh 2D de taille n x n sur laquelle on veut placer une application, ils ont utilisé l'algorithme de routage XY déterministe.

L'algorithme construit un arbre de recherche où chaque nœud contient un mapping (partielle). Un nœud de l'arbre de recherche est un tableau, chaque élément du tableau est un PE. La valeur d'un élément i signifie que le nœud (routeur) est affecté au PE avec l'identificateur i. B&B est divisé en deux étapes :

- Branche: un nœud de l'arbre de recherche non exploré est choisi. Alors, le prochain PE qui n'est pas encore effectué à un nœud (donnée par le nœud choisi) sera affecté à un nœud inoccupés, et les nœuds enfants sont générés;

- Bound: le nouveau nœud enfant sera vérifiée s'il est valide (c'est à dire qu'il répond à des contraintes de bande passante) et aussi s'il peut générer les meilleurs nœuds feuilles. Sinon, on peut élaguer ce nœud, c'est-à-dire évité de l'explorer.

Chaque nœud de l'arbre de recherche a un coût, ce coût représente l'énergie nécessaire pour la communication entre les PEs qui sont déjà mappés. On peut voir cette valeur comme le coût de mapping détenue par le nœud. Le coût d'un nœud enfant ne peut pas être inférieur au coût de son parent parce que l'enfant contient un PE de plus assigné au NoC (et évidemment ce PE va donner un trafic supplémentaire). En outre, un nœud est considéré comme valide, si et seulement si les exigences en bande passante entre les PEs actuellement mappés sont garanties. Quand un nœud interne est considéré comme invalide, le sous-arbre déterminé par ce dernier est évidemment invalide et donc n'a pas besoin d'être exploré.

En plus, pour vérifier si un nœud est valide pour décider de l'explorer ou non, les auteurs ont utilisé les deux paramètres suivants, qui sont attribués à chaque nœud (à partir de l'arbre de recherche):

- Upper Bound Cost (UBC): La valeur d'un nœud qui ne peut pas être inférieur au coût minimal de ses descendants (nœuds feuilles) valides;

- Lower Bound Cost (LBC): Le coût minimal d'un nœud que ses descendants peuvent éventuellement atteindre.

Une architecture mesh 2D avec le routage XY a été utilisée, cet algorithme peut être adapté à n'importe quelle topologie du réseau, avec un mécanisme de routage statique.

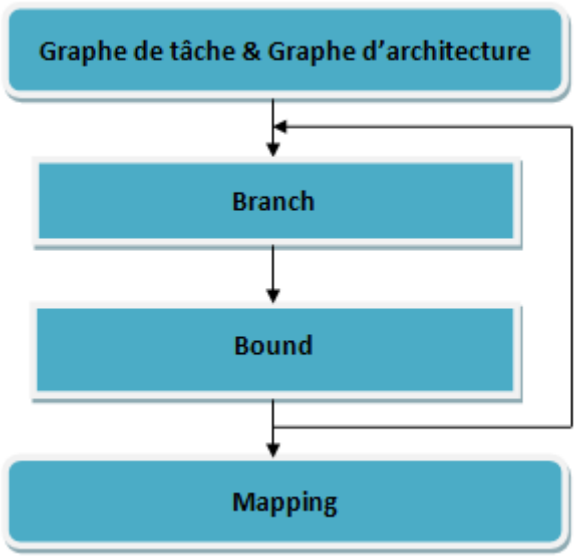


Figure 4.6 : Méthodologie de J. Hu et R. Marculescu

L'algorithme Branch & Bound a l'avantage d'être en mesure de trouver la solution optimale. Cependant, cela peut prendre un temps prohibitif, car la taille de l'arbre de recherche peut augmenter de façon exponentielle. L'élagage des nœuds invalides réduit considérablement l'espace de recherche. Cela ne suffit pas pour obtenir un algorithme de mapping assez rapide.

La qualité des solutions fournies par l'algorithme de mapping est fortement dépendante de la façon dont UBC et LBC sont calculés.

B&B doit faire un compromis entre l'énorme besoin en consommation de l'espace mémoire et la partie élaguée de l'arbre de recherche.

Le seul avantage de l'algorithme B&B comparé au recuit simulé est le temps d'exécution. En termes d'énergie, les améliorations fournies par B&B sont presque négligeables.

Les auteurs ont aussi défini un modèle d'énergie adapté aux architectures NoC Mesh-2D. Dans ce modèle, l'énergie consommée par l'envoi d'un bit du nœud i au nœud j est donnée par l'équation suivante :

$$E_{bit}^{i,j} = h_{i,j} E_{Rbit} + (h_{i,j} - 1) E_{Lbit} \qquad (4.3)$$

Où E_{Rbit} est l'énergie consommée par les routeurs et E_{Lbit} l'énergie consommée par les liens. $h_{i,j}$ est le nombre de routeurs traversés du nœud source au nœud destination. Cependant, l'équation (4.3) est linéaire constituée d'un variable $h_{i,j}$ et E_{Rbit} , E_{Lbit} qui sont considérés comme des constants. Alors, quelque soit les valeurs E_{Rbit} et E_{Lbit}, la minimisation de la distance moyenne donne les mêmes résultats de mapping qu'avec la minimisation de l'énergie consommée. Pour NoC mesh-2D avec le routage minimale, $h_{i,j}$ est déterminée par la distance de manhattan entre le nœud $i(x_i,y_i)$ et le nœud $j(x_j,y_j)$:

$$h_{i,j} = |x_i - x_j| + |y_i - y_j| \qquad (4.4)$$

F.Vardi et al. [51] ont proposé une heuristique pour le placement des tâches afin de réduire le coût de communication globale, ils ont utilisé une liste de priorité et un modèle de crinkle. Le placement d'une application détermine la manière de placer cette dernière dans une architecture NOC (Figure 4.7). Un graphe représentant une application appelé un graphe de tâche (TG) et un graphe représentant une architecture.

Le graphe d'application G (V, E) est un graphe direct où chaque sommet v_i représente un IP ou PE et l'arc orienté (v_i, v_j) noté e_{ij} représente la communication entre v_i et v_j. Le poids de l'arc e_{ij} noté par $b(e_{i,j})$ représente le volume de communication de v_i à v_j.

Le graphe d'architecture G'(T, L) est un graphe direct où chaque sommet t_i représente une tuile dans l'architecture, et un arc l_{ij} représente le lien physique entre t_i et t_j. Le poids de l'arc l_{ij} noté par $E_{bit}^{ti,tj}$ représente l'énergie moyenne consommée pour transférer un bit de t_i à t_j. Le calcule de l'énergie consommée a été proposé dans les travaux dans [52].

L'algorithme de placement Crinkle est basé sur l'approche LS (Liste Priority). L'idée de base de l'algorithme est de construire trois listes TLS (Task Liste Priority) nommées respectivement TPL1, TPL2 et RTPL et une liste PPL (Platform Priority Liste) pour la plateforme.

TPL1 peut contenir une partie ou même toutes les tâches du graphe TG, tandis que TPL2 contient certaines tâches du TG, qui ne sont pas dans TPL1, et enfin RTPL (Remaining Task Priority List) contient les tâches qui ne sont ni dans TPL1 ni dans TPL2.

D'abord, l'algorithme détermine les tâches qui présentent un degré de connectivité le plus élevé dans TG comme FP (First Priority). Supposant que v_i est le FP, alors le volume des données échangées entre v_i et ses voisins est examiné jusqu'à ce qu'un voisin de v_i avec le

volume de communication le plus grand est trouvé comme la prochaine priorité nommé v_j selon l'équation Eq (4.5).

Ce processus continue à examiner les voisins de toutes les tâches dans le graphe TG selon l'équation Eq (4.5). Dans ce processus, si l'algorithme arrive à examiner une tâche qui n'a pas de voisin avec un volume de données grand, l'algorithme sera terminé et la liste TPL1 sera créée.

$$NextPriority(v_i) = v_j \quad \forall v_j \in V \; avec \; Max(b(e_{i,j})) \qquad (4.5)$$

Si plusieurs voisins ont le même volume maximal de données que v_i. Le voisin avec le plus grand volume total de données à transmettre sera choisi.

Pour certains TGs, l'algorithme proposé ne peut pas explorer toutes les tâches en une seule étape. Lorsque cela s'est produit, une seconde liste TPL2 sera construite en appliquant le même algorithme pour les autres tâches à partir de la même liste FP qui a été utilisé pour construire TPL1.

Si toutes les tâches sont dans TPL1 et TPL2, l'algorithme sera terminé. Autrement, l'algorithme va créer une autre liste nommée RTPL qui doit vérifier dans un premier temps les tâches voisines à partir de FP, qui ne sont ni dans TPL1 ni TPL2 en se basant sur le volume de données le plus élevé.

Si toutes les tâches dans le graphe TG sont dans TPL1, l'algorithme inverse la liste TPL1, cette liste sera ensuite placée sur la plateforme en se basant sur le motif (modèle) de crinkle. Le but d'inverser TPL1 est de mettre le FP plus proche du centre en vue d'atteindre l'un des objectifs des heuristiques. Autrement, l'algorithme génère TPL2. Après, si toutes les tâches du graphe TG sont dans TPL1 et TPL2 inversées, l'algorithme doit fusionner les deux listes TPL1 et TPL2 inversées. Cette liste fusionnée sera mappé sur la plate-forme puis sur le modèle de crinkle. Sinon, l'algorithme génère une liste nommée RTPL et doit déterminer sa position. Il existe trois positions de base pour RTPL, au début de TPL1, à la fin de TPL2 et au centre. Le choix de la position de RTPL a un impact considérable sur la réduction du coût de communication.

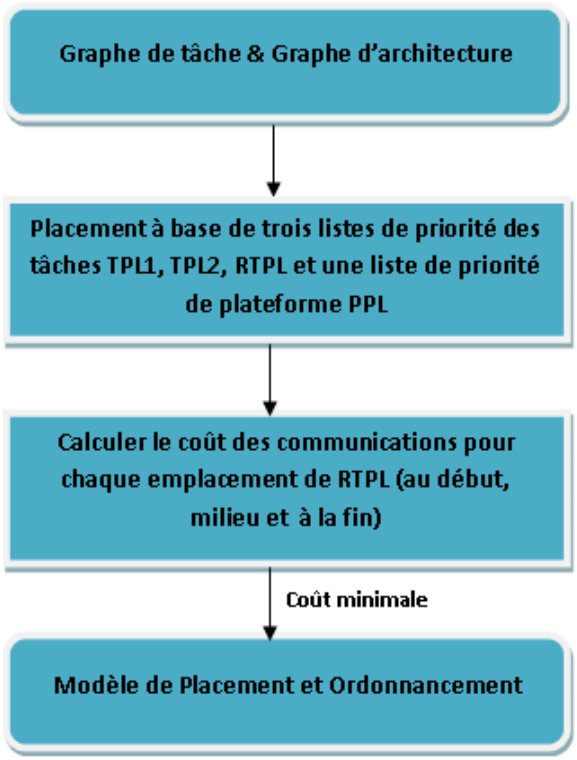


Figure 4.7 : Méthodologie de F.Vardi et al.

La figure 4.8 illustre cette approche avec l'application VOPD. Le chemin en bleu présente TPL1, en rose TPL2, et RTPL contient un élément nommé v16. Dans ce cas, RTPL n'a qu'un seul élément, mais dans autres TGs, elle peut avoir plusieurs éléments, en tout cas, la stratégie de placement est similaire. Après le placement de RTPL, le coût nommé CommCost de tous les différentes mappings qui ont été obtenus par l'algorithme de mapping de Crinkle sont calculés en se basant sur l'équation suivante :

$$CommCost = \sum_{\forall e_{i,j}} b(e_{i,j})\ x\ dist(map(v_i), map(v_j)) | \forall e_{i,j} \in E, \exists b(e_{i,j}) \neq 0 \qquad (4.6)$$

Où dist(a, b) est le nombre minimum de sauts entre le nœud a et b. Le meilleur choix est le placement avec la plus faible valeur CommCost.

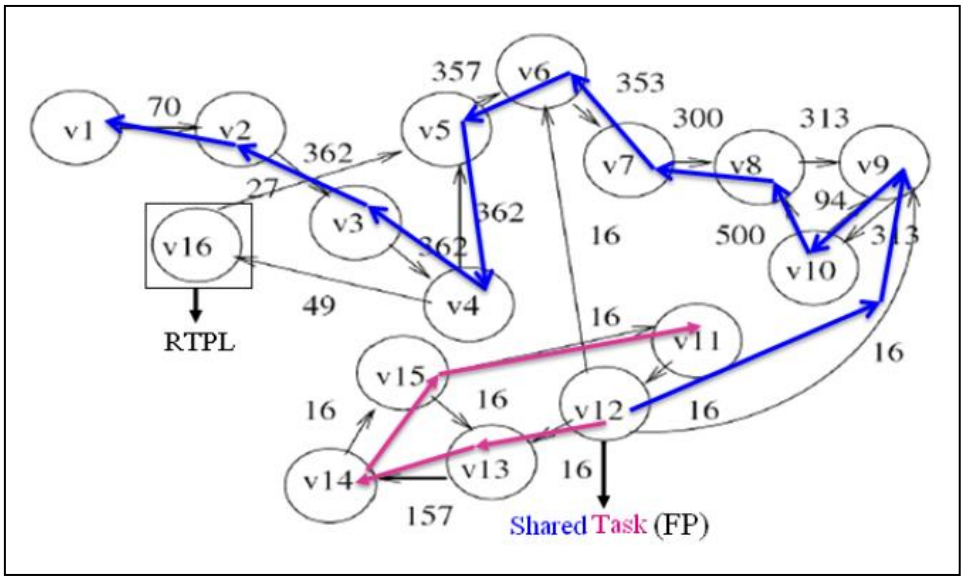


Figure 4.8 : Un chemin spécifique dans l'application VOPD

Harmanani et al [53] ont proposé une méthode efficace pour le placement d'applications sur une architecture mesh et l'allocation des chemins de routage tout en minimisant la surface occupée et la bande passante utilisée (figure 4.9). Formellement, pour une application donnée avec un nombre de cœurs N_c, et les contraintes en espace et en bande passante, trouver le placement des cœurs sur les nœuds dans une architecture mesh à 2D de telle sorte que la bande passante totale des communications soit réduite.

Placement basé sur le recuit simulé : les éléments de base pour implémenter l'algorithme de recuit simulé sont :

1) La définition de la configuration initiale
2) La définition de l'espace de configuration pour les voisins
3) La fonction de coût
4) L'ordonnancement

Représentation de la configuration : les auteurs ont proposé un algorithme de recuit simulé, qui a été modélisé en utilisant un vecteur à deux dimension, où le PE et le switch forment une position (x, y) dans le mesh à 2D. Chaque cœur est connecté au réseau de communication en utilisant le switch qui est connecté au switch voisin. Une cellule (une tuile) peut changer le cœur et le switch durant l'étape d'exploration du voisinage selon l'état de ses voisins.

La configuration initiale : La configuration initiale est générée par le positionnement aléatoire des cœurs PEs sur le mesh.

La fonction de voisinage : L'algorithme explore les placements possibles en choisissant aléatoirement deux tuiles et en échangeant leurs positions sur le mesh. Le processus est répété si la contrainte de surface n'est pas respectée.

Fonction de coût : pour N nombre de PEs, l'objectif est de trouver le mapping et l'allocation des chemins qui minimisent la bande passante. Le coût de communication d'un lien est représenté comme une fonction de la longueur du chemin et la possibilité de blocage lorsqu'il occupe un certain nombre de canaux d'acheminement. La fonction de coût est de minimiser:

$$\sum_{message} Numbre\ of\ routes\ x\ bandwidth \qquad (4.7)$$

En respectant la contrainte de la surface occupée.

Les auteurs ont proposé ainsi un algorithme de routage qui choisit un chemin parmi les chemins alternatifs selon l'état du réseau et l'occupation des files d'attente. Ainsi, si la congestion est détectée, les paquets seront acheminés vers un autre chemin libre. L'algorithme proposé est un algorithme de recherche en profondeur d'abord, le degré maximale de l'arbre est 4, qui est le nombre de sauts adjacents possible d'un paquet. L'algorithme est basé sur une méthodologie labyrinthe, qui a été prouvée pour être efficace dans les applications d'intelligence artificielle, où un paquet est acheminé à travers un labyrinthe selon un ensemble de règles, et re-routé seulement quand la congestion s'est produite, un lien en panne ou une tuile manquante.

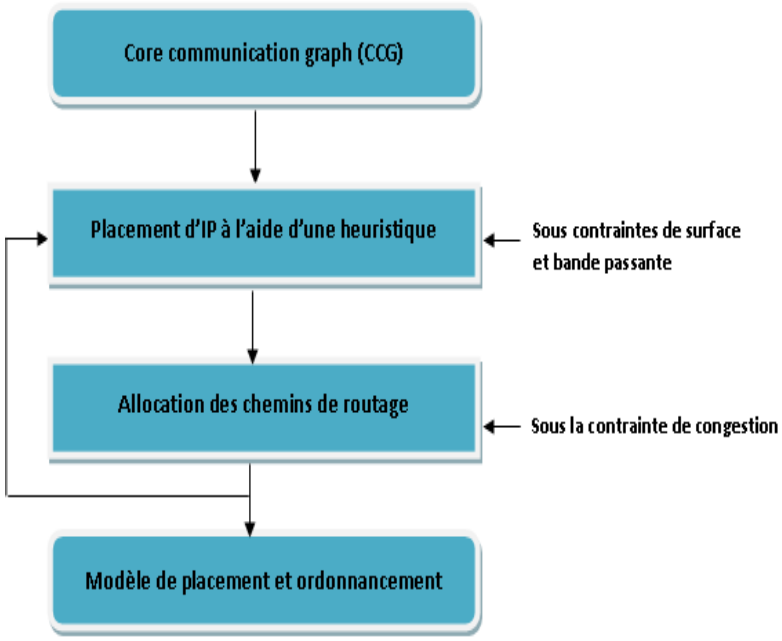


Figure 4.9 : Méthodologie de Harmanani et al

L'algorithme possède des informations sur la position des paquets. Si le chemin choisi conduit à une impasse, en raison de congestion par exemple, le paquet est alors redirigé vers le saut précédent et suit un autre chemin jusqu'à son arrivé à destination finale. À chaque fois, le saut suivant est choisi à partir de la position du destinataire par rapport à la position actuelle en utilisant les priorités présentées dans le tableau 4.1.

La position de la destination relative au routeur source	Priorité			
	première	**deuxième**	**troisième**	**quatrième**
En haut	En Haut	A droite	A Gauche	En bas
Haut vers la gauche	En haut	A gauche	A droite	En bas
haut vers la droite	En haut	A droite	A gauche	En bas
En bas	En bas	A droite	A gauche	En haut
Bas vers la droite	En bas	A droite	A gauche	En haut
A droite	A droite	En haut	En bas	A droit
A gauche	A gauche	En haut	En bas	A droite

Tableau 4.1 : Les règles de sélection du prochain saut [53].

Les auteurs ont utilisé le langage java pour implémenter leur méthode, et ils ont développé une application multi-thread où chaque cœur et switch ont été implémentés comme un thread indépendant. L'interconnexion entre les threads est basée sur l'algorithme de mapping proposé. Ils ont ensuite utilisé différents benchmarks qui varient en matière de connectivité, le volume de communication, et la taille.

Ascia et al [54] ont proposé une méthodologie de mapping multiobjectif basée sur l'AG. Dans cette méthodologie une architecture mesh-2D de taille n x m est utilisée. Ils essaient de choisir le placement optimal des blocs IPs sur les nœuds du réseau dans le but de minimiser l'énergie consommée et maximiser les performances. Dans leurs travaux, l'opérateur de croisement choisit à partir deux parents, celui qui a le meilleur mapping, puis l'IP avec lequel communique le plus de données est échangé avec un autre IP aléatoirement. L'opérateur de mutation choisit d'une manière aléatoire un IP, ensuite ce dernier va être placé sur un nœud voisin avec lequel il communique plus de données. Ils ont conclu que la définition des

opérateurs génétiques appropriés a un impact important sur les performances de l'algorithme génétique, ils admettent que la recherche dans ce domaine est nécessaire dans les futures recherches. Leurs études expérimentales sont menées sur les applications réelles (codeur/décodeur MPEG-4) qui confirme l'efficacité de leur approche.

Sahu et al [55] ont utilisé l'algorithme de Kernighan Lin [56] pour identifier les cœurs IP voisins en analysant leur besoin en bande passante.

Initialement, au niveau 0 tous les IPs se trouvent dans une seule partition, au niveau 1, la première partition sera divisée en deux partitions, dont le numéro des partitions est 0 et 1. Contenant chacun la moitié des IPs dans le graphe d'application. L'algorithme continue à partitionner le graphe jusqu'à ce qu'il ne reste que deux partitions. Lorsque les résultats de partitionnement de l'algorithme K-L dépendent de la partition initiale, on exécute l'algorithme L fois (prédéfinis) avec une partition initiale aléatoire différente à chaque itération. KL est un algorithme bi-partitionnement, alors le nombre d'IPs doit être en puissance de deux, dans le cas contraire on ajoute un certain nombre d'IPs complémentaires. Ces IPs doivent se connecter à tous les IPs et entre eux, un lien qui relie un IP avec un autre IP complémentaire a un coût 0. Cependant, les liens entre les IPs complémentaires possèdent un coût ∞. Leurs études expérimentales avec les benchmarks VOPD, MPEG-4 et PIP montrent que même si les performances statique de l'approche est similaire à celles des meilleures approches précédemment proposées, une amélioration de 8-17% en terme de latence lorsqu'on prend en considération les communications dynamiques entre les cœurs IPs.

Algorithme de Partitionnement Kernighan-Lin(G)

Entrée : Graphe d'application APG(C, E)

Sortie : ensemble de partitions

Début

Si |C| ≤ 2 **Alors** retourner

Meilleur_partition = null ;

Meilleur_coût = ∞ ;

Pour num_itér = 1 jusqu'à L **faire**

Partition = K-L(G) ;

Si (coût (partition) < meilleur_coût) **Alors**

Meilleur_coût = coût (partition) ;

Meilleur_partition=partition ;

FinSi

Générer les graphes G1 et G2 avec le meilleur_partition ;

Partitionnement K-L (G1) ;

Partitionnement K-L (G2) ;

FinPour

Fin

K-L (G)

Générer une bi-partition du graphe G

Répéter

Déverrouiller tous les nœuds ;

Pour i=1 jusqu'à |C|/2

Sélectionner le couple de sommets dont l'échange constitue le plus grand gain ;

Verrouiller les deux sommets ;

Enregistrer le gain g_i de l'échange ;

Echanger temporairement les deux sommets ;

FinPour

Trouver k telle que la somme partielle des gains $Gain_k = \sum_{i=1}^{|k|} g_i$ soit maximale ;

Si $Gain_k > 0$ **alors**

Echanger les 2 ensembles de sommets dont la somme des gains est maximale ;

Jusqu'à $Gain_k \leq 0$.

Zonghai et al [55] ont utilisé un algorithme du recuit simulé SA pour traiter le problème de mapping dans une architecture mesh 2D. Ils ont combiné une technique de clustering avec le recuit simulé afin d'accélérer la convergence vers les solutions optimales. L'algorithme de clustering exploite les propriétés de connectivité et de distance du réseau ainsi que les exigences de localité et de bande passante dans le graphe d'application. Dans leurs études expérimentales ils ont conclu que le recuit simulé peut être efficacement utilisé pour résoudre le problème de mapping, et la stratégie de clustering améliore le recuit simulé en temps d'exécution jusqu'à 30% sans compromettre la qualité des solutions.

Résumé

Le tableau 4.2 résume les méthodologies vues précédemment. Comme montré dans le tableau, la consommation d'énergie est l'objectif le plus important et le plus traité dans les approches d'optimisation [49] [50] [54]. Les deux approches [[49], [54]] sont les seules qui tentent de minimiser le volume des communications entre les cœurs IPs durant le placement des tâches, notamment la consommation d'énergie des tâches et des communications.

Réf	Auteurs	Tasks Mapping	Communication Mapping	Tasks Scheduling	Communication Scheduling	Voltage Selection	Voltage Selection Tasks	Objectives
[49]	D. Shin and al	GA	GA	LS	-	LS	-	Min Energy Consumption
[50]	J. Hu and R. Marculescu	BB and SA	-	-	-	-	-	Min Energy Consumption
[51]	F.Vardi and al	List Priority (TPL1, TPL2, RTPL, PPL)	-	-	-	-	-	- Max Performance Time - Min Energy Consumption
[53]	Harmanani and al	SA	Depth-first search	-	-	-	-	- Min area - Min Bandwidth
[54]	Ascia and al	AG	-	-	-	-	-	- Max Performance - Min Energy Consumption
[55]	Sahu et al	LMAP	-	-	-	-	-	- Min communication delay - Min Bandwidth
[57]	Zonghai et al	SA + Clustering technique	-	-	-	-	-	- Quatlity of Solutions - Min execution Time

Tableau 4.2 : Méthodologies pour les NOC

4.6 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art concernant le problème de mapping et d'ordonnancement. Nous constatons que les deux problèmes sont liés. Cependant, les deux problèmes sont NP-difficile, c'est la raison pour laquelle ils sont souvent traités séparément par les chercheurs.

Dans le chapitre qui suit, nous détaillerons notre approche pour résoudre le problème de mapping d'une application sur une architecture NoC.

Chapitre V :

Contribution

5.1 Introduction

Le mapping est une étape importante dans la conception du NoC. Le résultat de mapping a un impact significatif sur le délai de communication, l'énergie consommée par les communications et le temps d'exécution du système.

Le but de ce travail est de proposé une technique permet la résolution du problème de mapping des cœurs PEs sur les nœuds d'un réseau sur puce NoC, afin de minimiser l'énergie consommé et la bande passante requise.

5.2 Algorithmes de Clustering et de Mapping

5.2.1 Algorithme de Clustering

L'algorithme KL (Kernighan-Lin) [56] est souvent utilisé pour résoudre le problème de partitionnement des graphes. L'idée de cet algorithme est de trouver deux sous-ensembles de sommets de même taille chacun dans une partie de la bissection, puis chaque sous-ensemble est à son tour est partitionné en deux d'une manière récursive. A chaque itération, l'algorithme échange successivement deux paires de sommets entre deux partitions pour parcourir un sous-ensemble de solution jusqu'à ce que plus aucun sous-ensemble diminuent le coût de coupe ne puisse être trouvé. Le dernier partitionnement obtenu est donc le partitionnement de coût de coupe minimal trouvé par l'algorithme. Dans [55] les auteurs ont utilisé l'algorithme KL pour identifier les cœurs PE voisins en analysant leur besoin en bande passante.

Notre algorithme de clustering utilisé est basé sur les travaux effectués dans [57] pour générer le mapping (la solution) initial, qui est divisé en trois étapes :

1- Clustering (groupement) les nœuds (tuile) du réseau

2- Clustering les cœurs PEs du graphe d'application

3- Mapping les cœurs PEs sur les nœuds du réseau

1- Clustering les nœuds du réseau

Cette étape consiste à grouper les nœuds du réseau dans le même cluster (région), ce groupement est basé sur la distance minimale entre les nœuds du réseau. La figure 5.1 montre un exemple de clustering d'une architecture NoC Mesh 2D de taille 4x4 nœuds. On définit la distance d'un nœud t_i par la formule suivante :

$$d(t_i) = (s_i, h_i) \qquad (5.1)$$

Tel que

s_i : le nombre de liens sortants du nœud t_i.

h_i : la distance totale d'un nœud t_i vers tous les autres nœuds du réseau.

h_i trouve la connectivité totale d'un nœud par la formule suivante :

$$h_i = \sqrt{\sum_{j=1}^{k}(x_j - x_i)^2 + (y_j - y_i)^2} \qquad (5.2)$$

Tel que

k : le nombre de nœuds sur NoC

(x_j,y_j) : les coordonnées d'un nœud t_j

En utilisant cette propriété de distance, le clustering peut être réalisé par les étapes suivantes :

1- Calculer la distance $d(t_i)$ de chaque nœud.

2- Trier les nœuds dans un ordre descendant de leur distance. Et comparer le tuple (s, h) selon les relations suivantes :

$(s_1, h_1) > (s_2, h_2)$ si $s_1 > s_2$ ou $s_1 = s_2$ et $h_1 > h_2$.

$(s_1, h_1) < (s_2, h_2)$ si $s_1 < s_2$.

$(s_1, h_1) = (s_2, h_2)$ si $s_1=s_2$ et $h_1= h_2$.

3- Grouper séquentiellement les nœuds dans des clusters selon leurs distances $d(t_i)$.

Le nombre de clusters ne devrait pas être fixé auparavant. Plutôt, selon la distance $d(t_i)$.

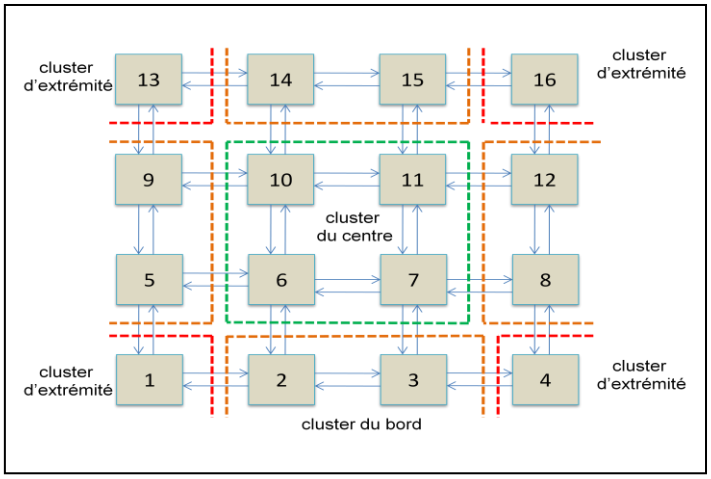


Figure 5.1 : Clustering d'un réseau 4x4 Mesh 2D [57].

2- Clustering des cœurs PEs du graphe d'application

Dans cette étape, le nombre de clusters et le nombre de cœurs PE dans chaque cluster est hérité du clustering utilisé pour les nœuds du réseau (la topologie).

La distance d'un cœur PE dépend du nombre de connexions et la bande passante pour chaque connexion.

On définit la distance d'un PE (un cœur c_i) par la relation suivante:

$$d(c_i) = (e_i , w_i) \qquad (5.3)$$

Tel que :

e_i : le nombre de connexions (liens) sortantes d'un PE

w_i : la somme des bandes passantes requises par un PE_i pour communiquer avec tous ses voisins.

w_i capture les communications nécessaires totales

$$w_i = \sum_{j=1}^{l_i} bw_{i,j} \qquad (5.4)$$

Tel que

l_i : le nombre de PEs voisins connectés au PE_i

$bw_{i,j}$: la bande passante requise par un PE_i pour communiquer avec PE_j

Le clustering est réalisé par les étapes suivantes :

1- Calculer la distance $d(c_i)$
2- Trier les PEs selon l'ordre descendant de leurs distances. La comparaison est similaire au clustering des nœuds du réseau
3- Grouper séquentiellement les PEs par clusters.

3- Mapping les cœurs PEs sur les nœuds du réseau (Génération du mapping initial) : la génération du mapping initial consiste à placer les cœurs PEs d'un cluster sur les nœud d'un cluster correspondant dans le réseau selon leurs distances les un après les autres, tel que les PEs avec une grande distance seront placés sur les nœuds d'un cluster du centre (le cluster qui peut satisfaire les exigences en bande passant élevés), ensuite sur les nœuds du bord, enfin sur les nœuds d'extrémités comme illustré par la figure 5.1.

5.2.2 Algorithme de Mapping (Algorithme génétique proposé)

Dans notre travail, l'algorithme génétique a été choisi parmi une variété d'heuristiques utilisées pour la résolution du problème de mapping des cœurs PEs sur les nœuds d'une architecture NoC, ce choix a été fait pour les raisons suivantes :

- Les différents mappings peuvent être directement trouvés en appliquant les opérateurs génétiques.
- L'efficacité des algorithmes génétique revient de bien choisir ses paramètres, ainsi que les opérateurs génétiques (sélection, mutation et croisement) sont appropriés à l'exploration de l'espace de recherche.
- Les Algorithmes génétiques permettent de prendre en considération le multi-objectif. Avec plus de précision.
- Un compromis entre la simplicité et le temps d'exécution dépend au type et la taille du problème.

Notre algorithme met en œuvre un mécanisme élitiste, l'algorithme est développé pour NoC mesh 2D de taille NxM. Il utilise un modèle analytique nommé bit-energy [54] pour calculer l'énergie de communication sur NoC. Et la bande passante est également envisagée.

Avant de décrire la façon dont notre algorithme fonctionne, nous montrons comment nous avons représenté notre problème de mapping en termes de gènes et de chromosomes.

Chaque IP est identifié de façon unique par un nombre entier positif, qui forme un gène. Ainsi, le chromosome est un tableau à une dimension contenant des nombres positifs non répétitifs (codage des PEs), une tuile vide (un routeur non occupé) est représenté par la valeur "-1" comme illustré par la figure 5.2.

Le $i^{ème}$ gène (i = 1, N × M) du chromosome encode un PE placé dans la

$[i/M]\ i^{ème}\ ligne\ et \begin{cases} i\%M, & i\%M \neq 0 \\ M, & i\%M = 0 \end{cases}$ $j^{ème}$ colonne (où % est l'opérateur modulo).

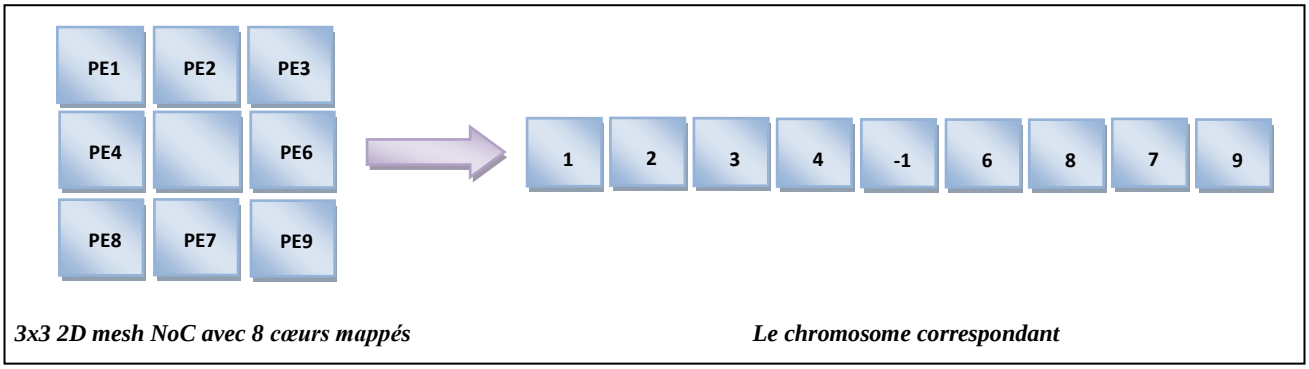


Figure 5.2 : Le codage utilisé

Ainsi, un chromosome peut contenir plusieurs valeurs « -1 ». Sa longueur est de N x M gènes. Avant d'appliquer les opérateurs génétiques, on remplace cette valeur « -1 » avec les identificateurs uniques de sorte que le chromosome contient un ensemble de nombres non

répétitifs. Cela nous permet de travailler avec des opérateurs génétiques pour des problèmes de permutation. Après que l'opérateur génétique est appliqué, les identificateurs "-1" seront remis en place.

Notre algorithme commence avec une population générée par l'algorithme de clustering vu dans la section précédente. La taille de la population est fixée par défaut à 100.

La fonction objectif de l'algorithme génétique est donnée par le modèle analytique bit-énergie [54]. Notre algorithme utilise la sélection d'un individu par tournoi binaire [38], cette technique est disponible dans jMetal[3]. Son principe est de sélectionner deux individus choisis aléatoirement pour reproduire de nouveaux individus avec une meilleure fonction de fitness. La reproduction consiste à appliquer les opérateurs génétiques (croisement et mutation).

Les probabilités de croisement et de mutation sont des paramètres de l'algorithme. Par défaut, nous travaillons avec une probabilité de 90% pour le croisement et 1/n pour la mutation (où n est le nombre de nœuds du NoC). Avec l'opérateur de croisement binaire simulé SBX (voir section 3.6.2), et la Mutation par permutation qui sont disponibles dans la bibliothèque jMetal [61] (voir section 6.2), nous avons utilisé deux autres opérateurs de croisement et de mutation.

L'algorithme doit s'arrêter après un nombre déterminé de générations. Il peut également s'arrêter après un certain nombre donné de mappings.

5.2.2.1 L'opérateur de croisement

L'opérateur proposé est basé sur l'opérateur de croisement PBX (Position Based Crossover). L'idée est de fixer un certain nombre de positions lors de la recombinaison, et de maintenir l'ordre relatif entre les gènes. Alors PBX est basé sur la position et l'ordre.

L'opérateur de croisement basé sur la position (PBXA)

L'opérateur PBXA est l'extension (amélioration) de l'opérateur à base de position vu précédemment (section 6.3.2), de telle sorte que les cœurs PEs qui restent inchangés ne sont pas choisis aléatoirement. Au contraire, les PEs qui communiquent plus de données, sont ceux qui ne changent pas de position. L'idée de base consiste à maintenir les positions des PEs avec des volumes d'échanges importants, car le déplacement de ces PEs notés HS[4] peut produire des mauvais mapping par rapport à ceux avec des volumes d'échanges faibles.

Il commence d'abord par trouver les cœurs PEs de volume d'échanges importants avec un seuil fixé à 50%. Ensuite, les deux parents P1 et P2 sont évalués en utilisant la formule suivante:

[3] JMETAL : Metaheuristic Algorithms in Java en anglais
[4] hot spots : les nœuds de trafic importants.

$$Coût = \sum_{\substack{i,j \in PE \\ i \# j}} v(Pe_i, Pe_j) \text{ x } h(Pe_i, Pe_j) \qquad (5.5)$$

Où PE est l'ensemble de PEs de type HS. Prenant en considération à la fois le volume des données *v* et la distance de Manhattan *h* entre les HS. Le parent avec le meilleur mapping des HS est celui qui va être sélectionné. Ensuite, un enfant est créé en plaçant les HS dans les mêmes positions où se trouvent dans le parent sélectionné. Le reste des gènes pour les enfants sont remplis à partir de l'autre parent, comme dans PBX.

L'opérateur PBXA est similaire à celui présenté dans [54]. La différence est qu'un PE de type HS n'est pas simplement échangé avec un PE choisi au hasard, mais plutôt, la moitié de la plupart des PEs est fixé. Alors que le croisement dans [54], il s'agit d'une mutation, PBXA se comporte comme un croisement basé sur la position. Il maintient la position des HS. Parce qu'il tente de minimiser la distance entre les PEs les plus communicants, PBXA devrait générer des mappings qui réduisent l'énergie consommée par les communications et la bande passante des liens.

5.2.2.2 L'opérateur de mutation

Cette section présente l'opérateur de mutation utilisé par notre algorithme.

L'opérateur de mutation utilisé dans ce travail est basé sur celui utilisé dans [54]. L'idée consiste à modifier les valeurs des gènes d'un mapping M pour obtenir un mapping M'. Un nœud (routeur) dans l'architecture NoC noté N_s d'un mapping M est choisi aléatoirement, un Pe_s désigne un cœur Pe connecté au nœud N_s, et le cœur Pe_t désigne le cœur avec lequel il communique plus de données. Pe_s va être replacé (reconnecter) au nœud voisin de N_s dans le but de réduire la distance entre Pe_s et Pe_t, de cette manière on obtient à la fin un nouveau mapping.

Pseudo-code de l'opérateur de Mutation

```
Mapping M' = M
 Noeud N_s = Noeud_Aléatoire (M') ;
 Cœur Pe_s = M'^-1 (N_s) ;
 Cœur Pe_t = Comm_Max (Pe_s) ;
 Noeud N_t = M'(Pe_t) ;
 SI ( Row (M', N_s) < Row (M', N_t)) Alors
    N'_s = Sud (M', N_s) ;
 SINON
     SI
       Row (M', N_s) > Row (M', N_t) Alors
       N'_s = Nord (M', N_s) ;
      SINON
         SI
           Col (M',N_s) < Col (M',N_t) Alors
           N'_s = east (M', N_s) ;
         SINON
           N'_s = ouest (M', N_s) ;
 SWAP (M', N_s, N'_s) ;
Retourner (M') ;
```

La fonction Nœud_Aléatoire (M) donne un nœud choisi aléatoirement à partir d'un mapping M.

La fonction Comm_Max (Pe) retourne comme résultat le cœur Pe (bloc IP) avec lequel il communique plus de données. Row (M, N) et Col (M, N) donnent respectivement la ligne et la colonne du nœud N d'un mapping M. Enfin, les fonctions Nord, Sud, Est, Ouest donnent les nœuds situent au Nord, Sud, Est, et Ouest du nœud N.

Notre algorithme génétique proposé peut se résumé comme suite :

1. Générer la population initiale de n chromosomes qui représentent un ensemble de mapping générés par l'algorithme de clustering. La longueur de chaque chromosome est égale au nombre de sommets dans le graphe d'application. Dans le chromosome, chaque cœur PE est identifié de façon unique par un nombre entier positif qui forme un gène. Ainsi, un chromosome est un tableau à 1 dimension contenant des nombres positifs non répétitifs (codage des cœurs PEs).

2. Créer une nouvelle population en appliquant les opérateurs génétiques (Sélection, Croisement et Mutation). L'opérateur de sélection utilisé est la sélection par tournoi binaire. Son principe consiste à sélectionner deux chromosomes parents basé sur leur fitness (celui qui a la fitness la plus élevée sera sélectionné).

3. Appliquer l'opérateur de croisement PBXA

4. Appliquer l'opérateur de mutation

5. Répéter les étapes 2 à 4 jusqu'à ce que la distance minimale n'a pas changée pendent un nombre d'itérations ou le nombre déterminé de générations est atteint.

6. Retourner le meilleur mapping.

5.3 Conclusion

Nous avons présenté dans ce chapitre les détails de notre contribution afin de trouver le mapping optimale d'applications sur les nœuds d'un NoC.

Notre approche est basé sur un algorithme de clustering pour trouver une population initiale assurant qu'on a quelques bons individus et un algorithme génétique afin d'obtenir les meilleurs mappings en appliquant les opérateurs génétiques.

Le chapitre suivant décrit en détails l'implémentation de l'approche proposée afin d'évaluer les résultats obtenus.

Chapitre VI

« Implémentation et Expérimentation »

6.1 Introduction

Le présent chapitre décrit l'implémentation et la mise en œuvre de l'approche proposée dans le chapitre précédent pour la résolution du problème de mapping multi-objectifs dont le but d'évaluer ses performances.

6.2 Environnement et outils de mise en œuvre

Notre approche a été implémentée sur un système d'exploitation linux (Ubuntu 12.04). Pour développer notre approche, on a choisi d'utiliser le langage java sous l'environnement de développent eclipse en raison de ces nombreux avantages.

Eclipse est un environnement de développement intégré (Intergrated Developpement Environnement) open source, multi-langages, très performant permet le développement des applications en vue de leur déploiement sous linux ou sous Windows.

Java est un langage de programmation orienté objet mise au point par Sun Microsystem. Initialement il a été conçu pour être intégré dans les appareils électroménagers. Il offre plusieurs avantages : libre, portable, indépendant du matériel, ce qui est un atout considérable pour les systèmes embarqués où les plates-formes matérielles sont très diverses.

jMetal (Metaheuristic algorithms in java) est une bibliothèque java fournit des outils pour résoudre les problèmes d'optimisation multi-objectifs basée sur des méta-heuristiques, cette bibliothèque fournit un ensemble de classes permettent de définir et implémenter n'import quel problème multi-objectif. Ainsi, la réalisation de différentes expérimentations. Les différents algorithmes de cette bibliothèque utilisent les mêmes bases (problème, solutions, opérateurs génétiques…etc)

6.3 Présentation du logiciel et description des modules

Notre logiciel de mapping est constitué de trois grands modules.

6.3.1 Le module ordonnanceur

Les tâches de l'application doivent d'abord être attribuées aux processeurs hétérogènes. Cela se fait habituellement avec un algorithme d'ordonnancement (cf. figure 6.1). En plus de simplement assigner des tâches aux processeurs, l'algorithme d'ordonnancement détermine également l'ordre d'exécution des tâches. Ceci est utile lorsqu'il s'agit de contraintes temps réel. À partir de la bibliothèque des processeurs, notre algorithme sélectionne, pour chaque tâche le PE qui l'exécute plus rapidement. La sortie de l'algorithme d'ordonnancement est un graphe d'application (APG). Par rapport au graphe de tâche TG, l'APG spécifie l'affectation

des tâches aux PEs et représente l'entrée pour la phase suivante (le mapping). Le graphe APG est représenté par un fichier XML.

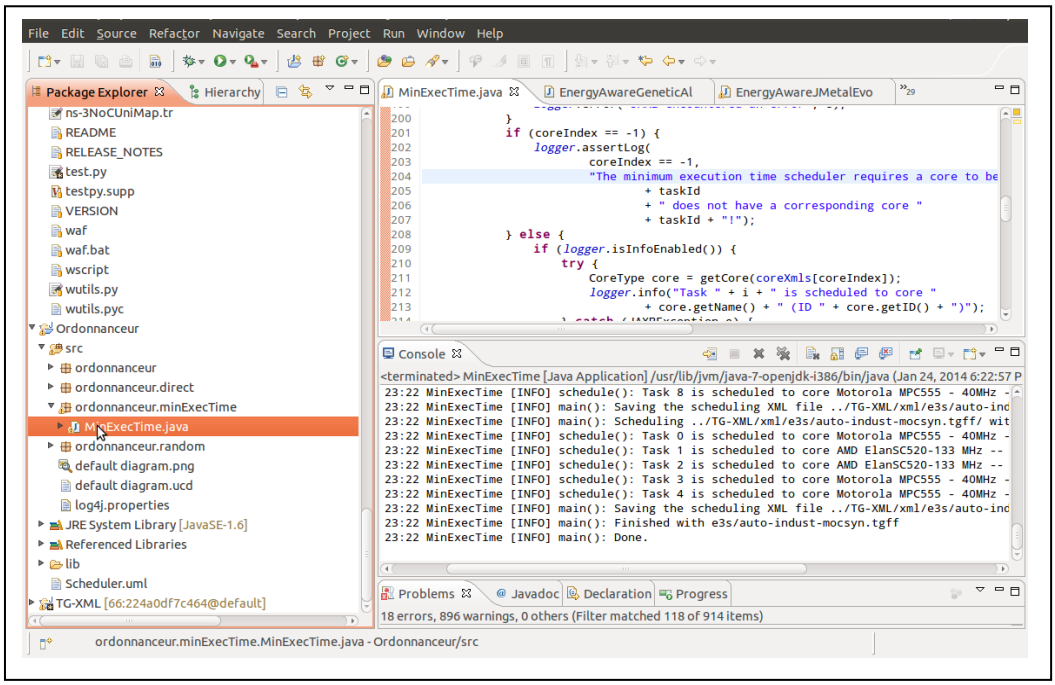


Figure 6.1 : Le module ordonnanceur

6.3.2 Le module Mapping

Le module Mapping contient la bibliothèque des algorithmes de mapping. L'algorithme de mapping joue le rôle de placement des PEs sur les nœuds (tuiles) disponibles dans NoC (cf. figure 6.2). Il utilise le graphe d'application fournit par le module Ordonnanceur et réalise ensuite le mapping de tous les PEs sur les nœuds du NoC, cela par l'utilisation de l'AG. Le mapping produit par l'algorithme est également décrit à l'aide d'un fichier XML.

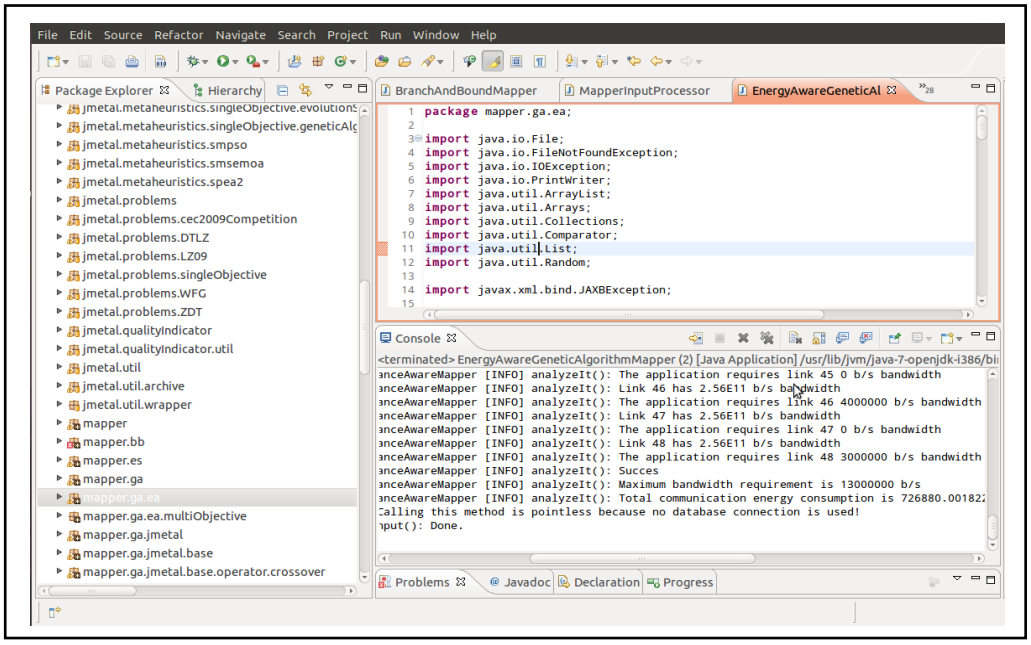


Figure 6.2 : Le Module Mapping

6.3.3 Le Module du réseau

Le module réseau est représenté par des schémas XML modélisent l'architecture NoC, contient des modèles pour les topologies, les nœuds et les liens.

Le but de ce modèle est de capturer les caractéristiques de l'architecture NoC utilisée par l'algorithme de mapping.

6.4 Les principales étapes d'utilisation du logiciel :

Nous présentons maintenant les étapes qui décrivent la manière d'utilisation de notre logiciel. D'abord, l'utilisateur doit charger un exemple à partir d'un fichier XML. Pour assigner les tâches de l'application aux PEs, on doit exécuter un algorithme d'ordonnancement. Le fichier XML obtenu en sortie est ensuite utilisé comme entrée par l'algorithme de mapping, ce qui va générer un autre fichier XML en sortie contenant le mapping des PEs sur les nœuds du NOC. On aura également besoin d'une topologie NoC (pour cela on utilise le module réseau).

On peut lancer l'algorithme génétique avec les paramètres suivants : la taille de la population initiale, le nombre de générations, probabilités de croisement et mutation.

Une fois que toutes ces données sont introduites, il reste à appuyer sur run pour lancer les calculs de recherche de placements (solutions) multi objectifs. Les résultats (cf. figure 6.3) de ces derniers seront affichés sous l'onglet « console » en bas de la fenêtre principale.

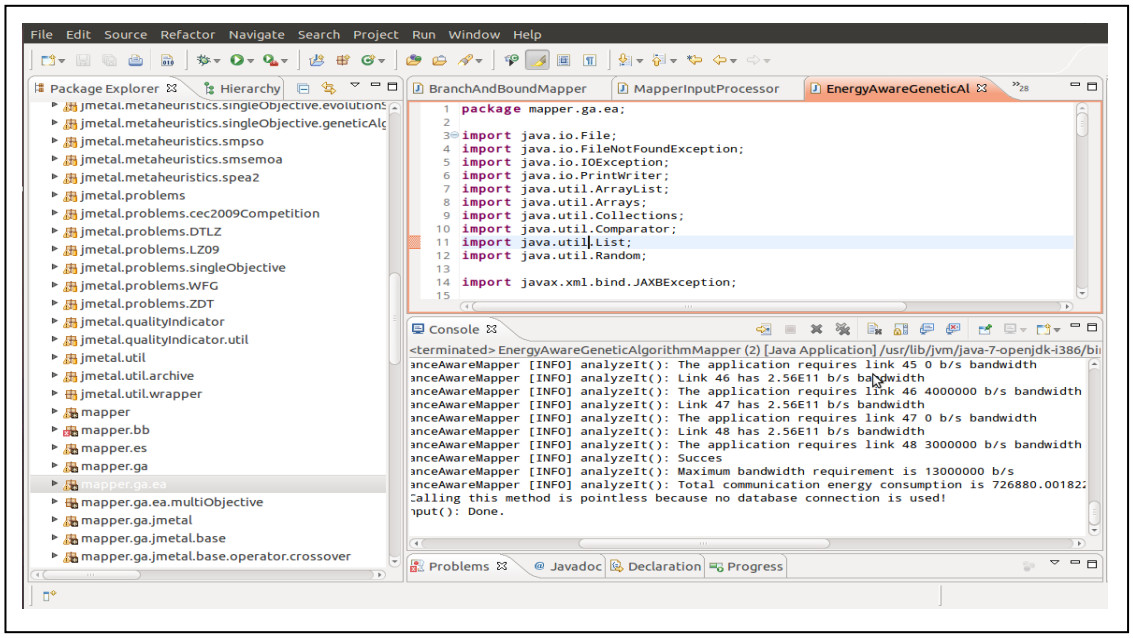


Figure 6.3 : Fenêtre principal du logiciel et l'affichage des résultats

6.5 Les Benchmarks

Nous présentons dans cette section les benchmarks utilisés dans notre travail

6.5.1 L'outil TGFF (Task Graphe For Free)

Un graphe de tâche peut être généré automatiquement en utilisant l'outil TGFF [59]. Cet outil permet à l'utilisateur de générer des graphes de tâches de taille théoriquement illimitées. Dans TGFF, les nœuds du graphe représentent les tâches et les arcs représentent les communications entre les tâches.

Un nœud peut également être considéré comme un processeur PE et l'arc peut être interprété comme une ressource de communication. TGFF permet à l'utilisateur de spécifier le nombre de tâches dans le graphe, leur degré maximal et minimal.

Un nombre d'attributs peut être associé aux processeurs et aussi aux ressources de communications. Ces attributs peuvent être : le temps d'exécution, l'énergie consommée, le coût, etc. les valeurs de ces attributs sont générés par TGFF, basés sur les paramètres spécifiés par l'utilisateur (par exemple : la valeur moyenne).

TGFF peut créer des graphes de taches acycliques, car TGFF prend en considération des systèmes temps réel. Les deadlines peuvent être fixés, où un délai est effectué à une tâche. Cela signifier que cette tâche doit re-exécuter par un processeur après un certain temps.

L'avantage d'utiliser TGFF est de fournir une méthode standard pour la génération des graphes de tâches.

6.5.2 Les Benchmarks E3S

E3S (Embedded System Synthesis Benchmarks Suite) [60] contient les graphes de tâches pour 5 applications embarquées : Automotive/Industrial, Consumer, Networking, Office Automation et telecommunication. Chaque benchmark est décrit par un ensemble de graphes de tâches. Il existe une version pour chaque graphe selon le type du système : Système distribué, Client/Serveur et Système sur Puce. Dans notre expérimentation nous avons utilisé les graphes de tâches conçus pour les systèmes sur puce.

Ces benchmarks sont basés sur les processeurs embarqués EEMBC (Embedded Microprocessor Benchmark Constoruim). Ils contiennent 34 processeurs embarqués caractérisés par : le temps d'exécution, l'énergie consommée, la fréquence, l'espace occupé, …etc.

Le benchmark Automotive/industrial

L'application automobile/industriel est composée de quatre Graphe de tâches: GT 0, 1, 2 et 3.

GT 0 modélise un système automobile embarqué avec deux interfaces CAN (Controller Area Network) d'une automobile. Un module pour réaliser les calculs des entiers, un module de calcule des nombres à virgule flottante et un actionneur basé sur la modulation d'impulsion en largeur «Pulse Width Modulation(PWM)»). La première interface CAN (can1) simule le traitement des requêtes à distance «Remote Data Request (RDR) » qui sont destinés aux unités de calcules des entiers et virgule flottante (FP). Le module à virgule flottante calcule la fonction arctan (x) en utilisant la série télescopique: $\arctan(x) = x.\frac{P(x^2)}{Q(x^2)}$, où P et Q sont deux polynômes et $x \in [0,1]$.

La deuxième interface CAN (can2) traite les messages RDR provient du module fp et les envoie à PWM. L'actionneur commande un moteur pas à pas.

GT 1 décrit un système automobile embarqué avec une réponse impulsionnelle infinie (IIR) et un module de transformation en cosinus discrète inverse (iDCT). Le filtre IIR teste la capacité du processeur d'effectuer des multiplications/accumulations.

Le module iDCT effectue la reconnaissance d'image par l'exécution d'une transformée en cosinus discrète inverse sur un ensemble de données d'une matrice d'entrée.

GT 2 modélise un système avec : une réponse impulsionnelle finie «Finite Impulse Response (FIR)», transformée de Fourier rapide (FFT), un module pour le calcule des matrices, module de calcule de la transformée de Fourier rapide inverse (iFFT), et un module qui calcule la vitesse de l'automobile.

Le module FFT exécute une analyse du spectre de puissance de l'onde d'entrée qui varie dans le temps.

Le module matrices effectue une décomposition LU d'une matrice carrée, calcule le déterminant de la matrice d'entrée et effectué un produit avec une autre matrice.

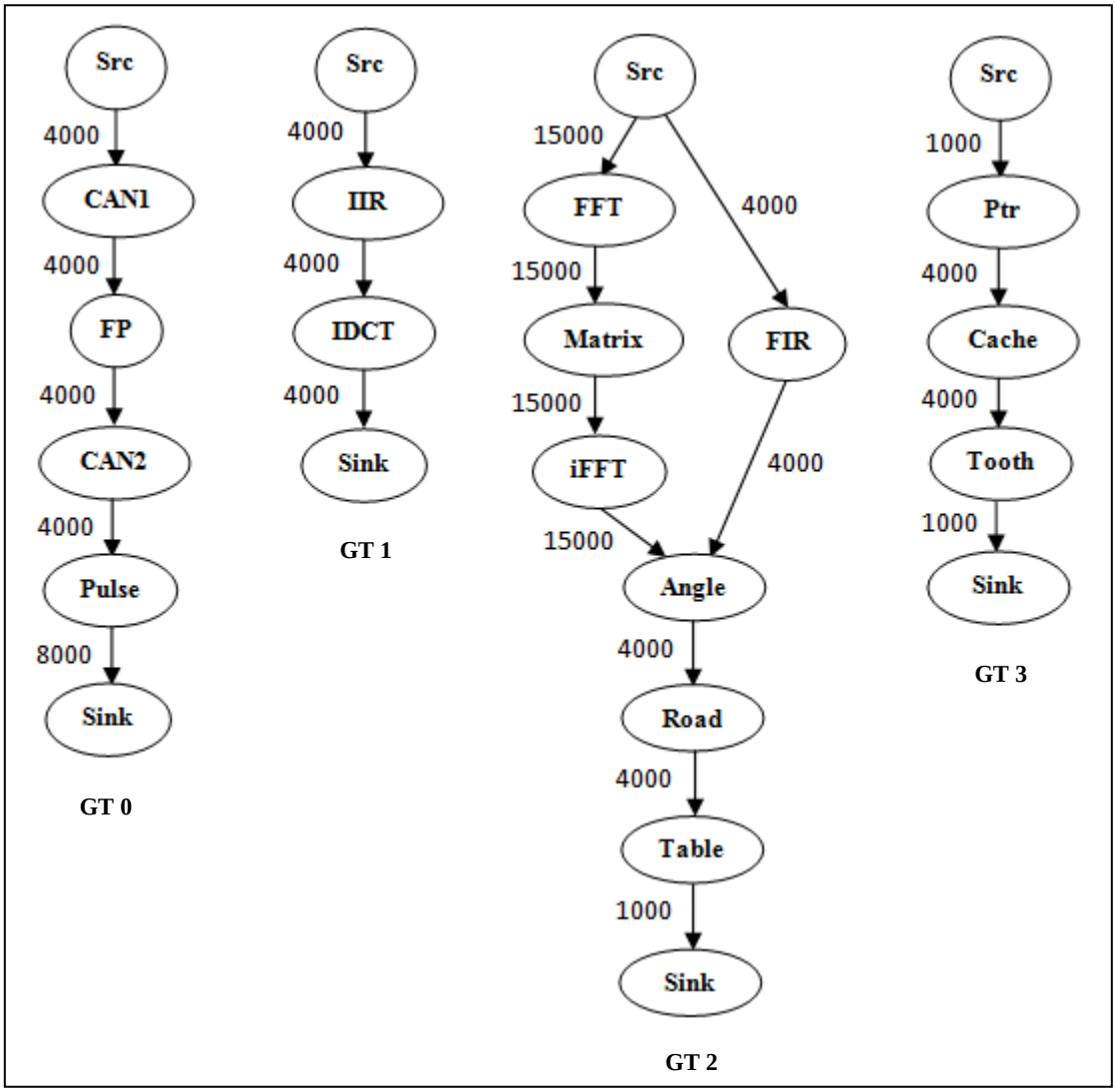


Figure 6.4 : Graphe de tâches du benchmark Automotive/industrial [60]

GT 3 contient les modules suivants : pointer chasing (ptr), cache "Buster" et Tooth-to-Spark.

Pointer chasing effectue les manipulations des pointeurs.

Cache « Buster » est une application qui utilise les données et les pointeurs de fonctions afin de résoudre le problème de localité des données.

Tooth-to-Spark est une partie de l'unité de contrôle du moteur (ECU)

Le benchmarks consumer

Le benchmark consumer est composé de deux modules : un module d'acquisition d'images à partir d'une caméra vidéo (GT 0) et un module d'impression d'image (GT 1).

Les tâches dans GT 0 traitent l'acquisition d'images, le filtrage (appliquée sur les couleurs vert, rouge et bleu de l'image), la conversion de l'espace des couleurs (RGB vers YIQ), la compression en utilisant JPEG et enfin le stockage.

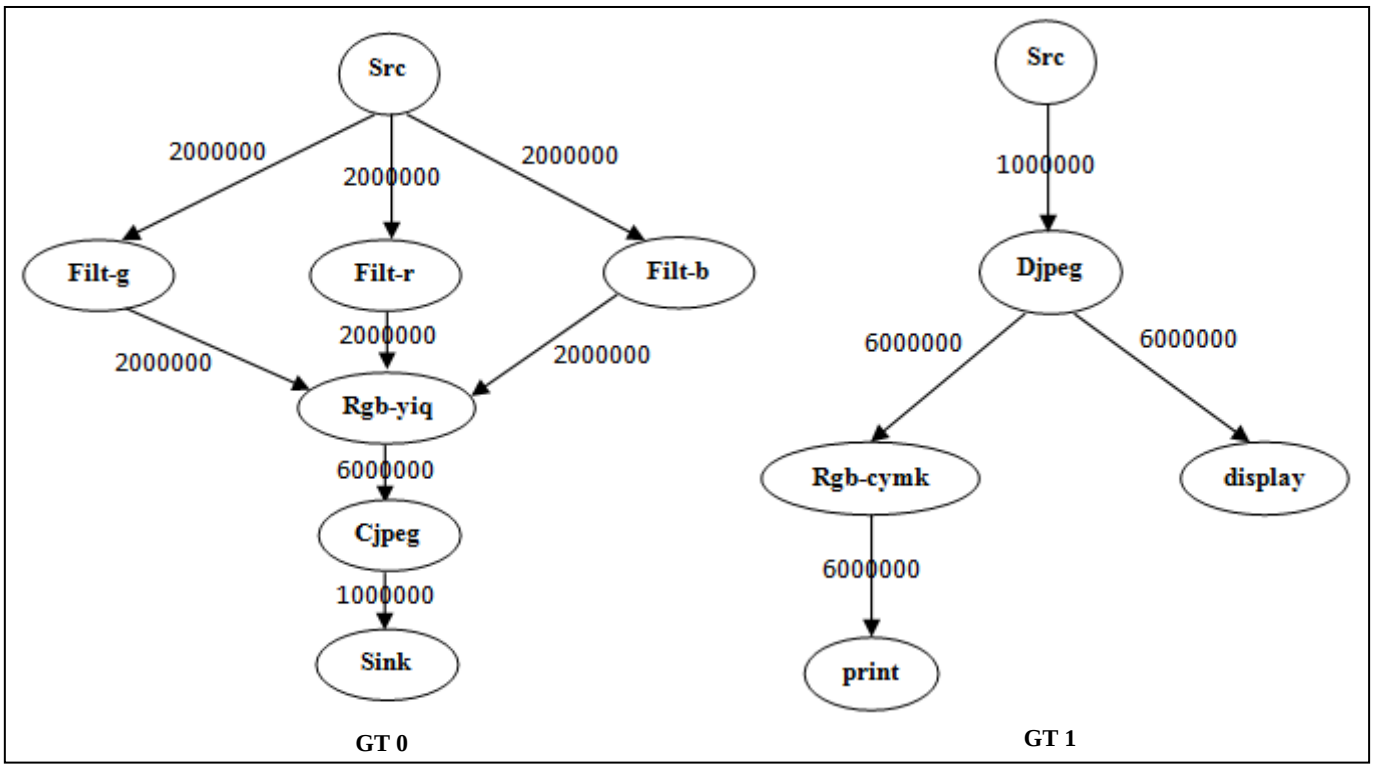


Figure 6.5 : Graphe de tâches du benchmark Consumer [60]

GT 1 modélise la récupération d'images à partir du stockage de données, JPEG pour la décompression d'image, la conversion à partir du modèle de couleur RGB en modèle de couleur CMYK et l'impression. Les images à imprimer peuvent également être consultées sur l'écran.

Le benchmark office-automation

Le benchmark office-automation contient des tâches de traitement d'images et de textes, effectuées par une imprimante.

La tâche rotation exécute un algorithme de rotation pour faire pivoter une image bitmap de 90°. Ceci est utile pour les imprimantes lors de la commutation entre les modes portrait et paysage.

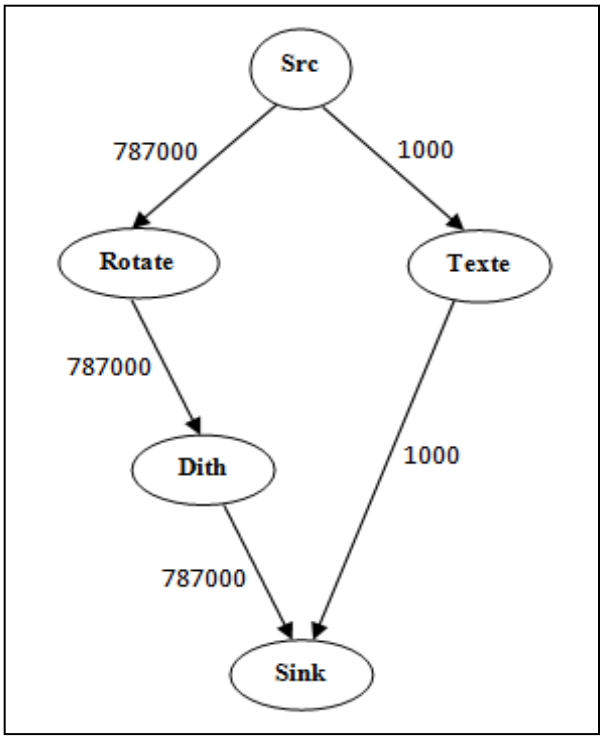


Figure 6.6 : Graphe de tâches du benchmark Office-automation [60]

La tâche dithering (dith) exécute l'algorithme de Floyd-Steinberg qui exécute un tramage par diffusion d'erreur pour convertir une image du niveau de gris en une forme prête pour l'impression.

L'analyse de texte est effectuée par la tâche texte. Il représente une application de l'imprimante qui analyse un langage de commande comme PCL ou PostScript.

6.6 Les résultats expérimentaux

Cette section présente l'évaluation de notre approche en se basant sur le modèle proposé dans [58], on a évalué les benchmarks E3S et on a comparé notre approche avec les approches B&B et SA cités auparavant dans [50], nous avons considéré l'architecture la plus souvent utilisée Mesh-2D. Pour chaque Benchmark la taille de la topologie (architecture du réseau) doit être supérieur ou égale au nombre de PEs.

Le tableau suivant présente la taille de la topologie Mesh-2D utilisé pour le mapping de chaque benchmark.

Benchmark	Nombre de tâches	Nombre de nœud dans la topologie	La taille du NoC
Automotive-industrial	24	16	4x4
Consumer	12	12	4x3
Office automation	5	9	3x3

Tableau 6.1 : Les caractéristiques des benchmarks et la taille des NoCs utilisés

Après l'exécution de notre approche en se basant sur les paramètres indiqués dans les tableaux 6.1 et 6.2, les résultats obtenus de la comparaison de notre algorithme avec les algorithmes B&B et recuit simulé SA proposés dans [50] sont présentés dans les figures qui suivent.

Algorithme	Paramètres de base	
Algorithme génétique	- Taille de la population	100
	- Nombre de génération	1000
	- Probabilité de croisement	90%
	- Probabilité de mutation	1/n

Tableau 6.2 : Représentation des paramètres de base

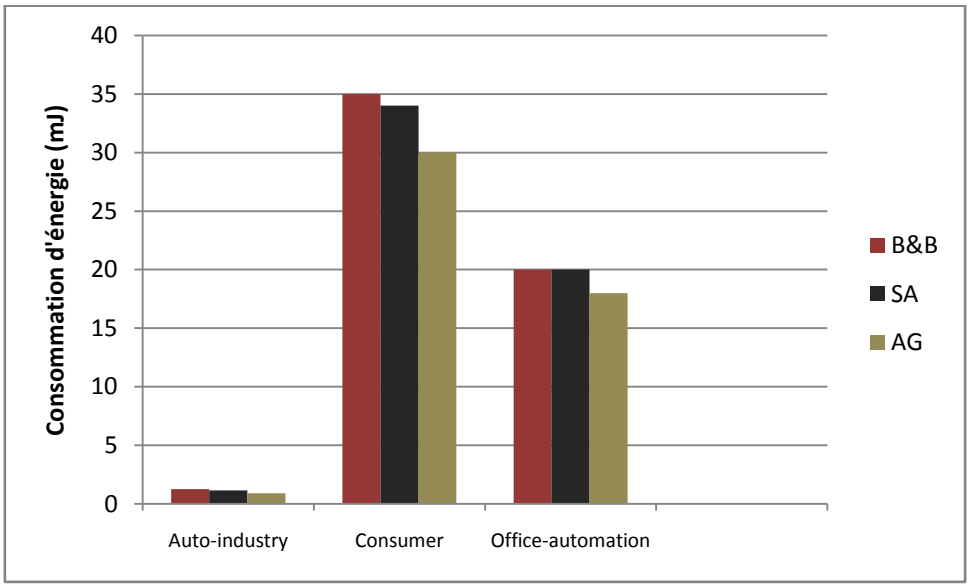


Figure 6.7 L'énergie consommée pour les 3 benchmarks (E3S)

La figure 6.7 présente une amélioration en consommation d'énergie moyenne de notre algorithme avec 17% comparé à l'algorithme B&B et de 15% pour l'algorithme SA qui donne lui aussi de bons résultats.

Avec la minimisation de l'énergie consommée par les communications, nous sommes également intéressés à la minimisation de la bande passante maximale. On a utilisé notre algorithme génétique avec les opérateurs génétiques, les mappings pour les trois benchmarks E3S ont été évalués.

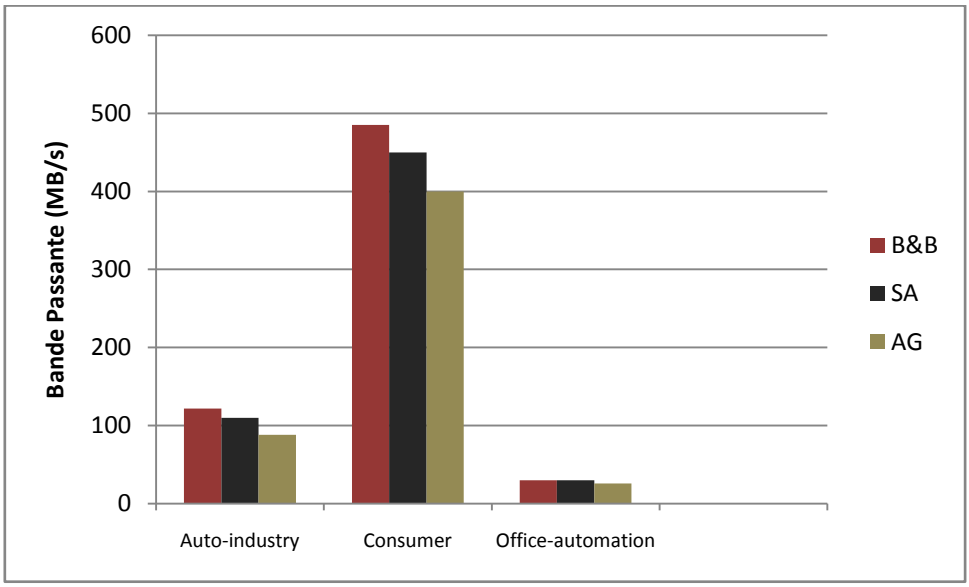


Figure 6.8 : La bande passante maximale pour les trois benchmarks (E3S)

La figure 6.8 montre l'utilisation maximale de la bande passante moyenne des liens de trois benchmarks E3S. Comme il est montré sur la figure notre approche permet d'économiser plus de 19% de la bande passante maximale utilisée par les liens comparé à B&B et de 15% pour SA.

Dans ce qui suit nous allons présenter les résultats expérimentaux réalisés sur trois benchmarks aléatoires (1, 2, 3) (TG) générés par l'outil TGFF [59], chaque benchmarks contient 9 PEs avec le nombre de tâches 9, 12, 18 respectivement. Qui peuvent être mappés sur une architecture 3×3 2D-mesh NoC. La bande passante nécessaire pour connecter deux nœuds est répartie uniformément sur l'intervalle [0, 500Mbytes]. Le volume de trafic d'un lien est réparti uniformément sur l'intervalle [0, 1Gbits].

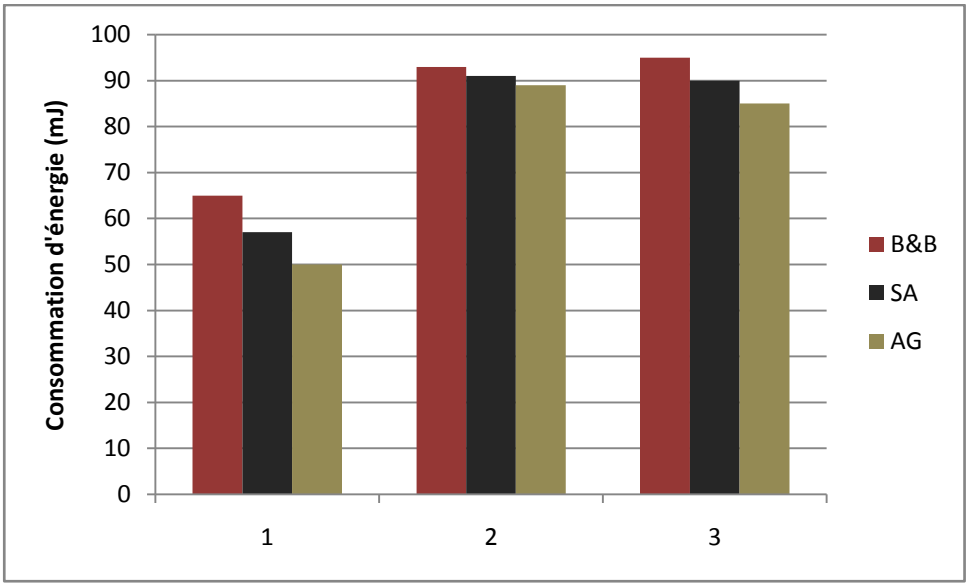


Figure 6.9 L'Energie consommée pour 3 benchmarks (TGFF)

La figure 6.9 montre que notre approche permet d'économiser 15% (en moyenne) pour B&B et de 7% pour SA de la consommation d'énergie.

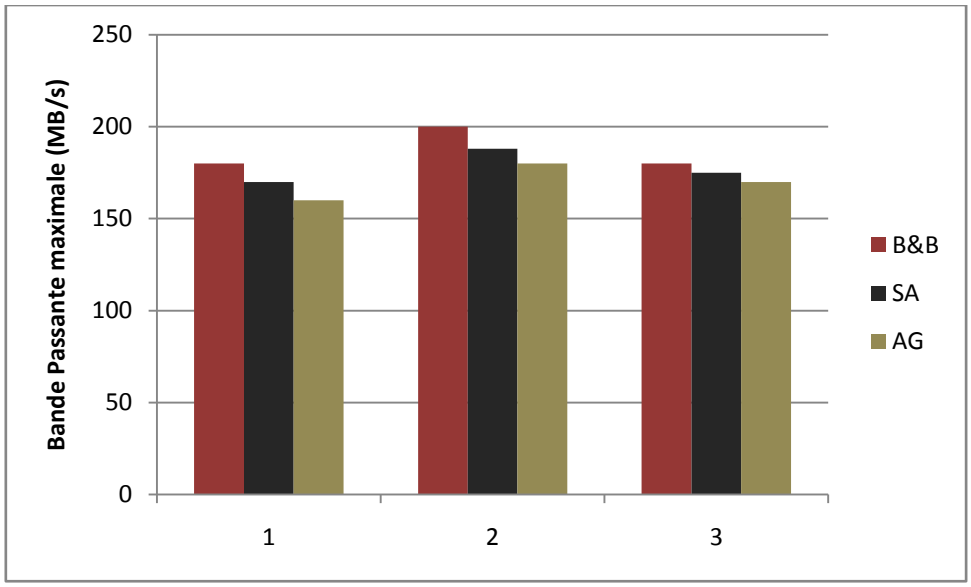


Figure 6.10 La bande passante maximale pour 3 benchmarks (TGFF)

La figure 6.10 montre que notre approche permet d'économiser plus de 10% de la bande passante comparée à l'approche B&B et de 5% pour SA des trois benchmarks.

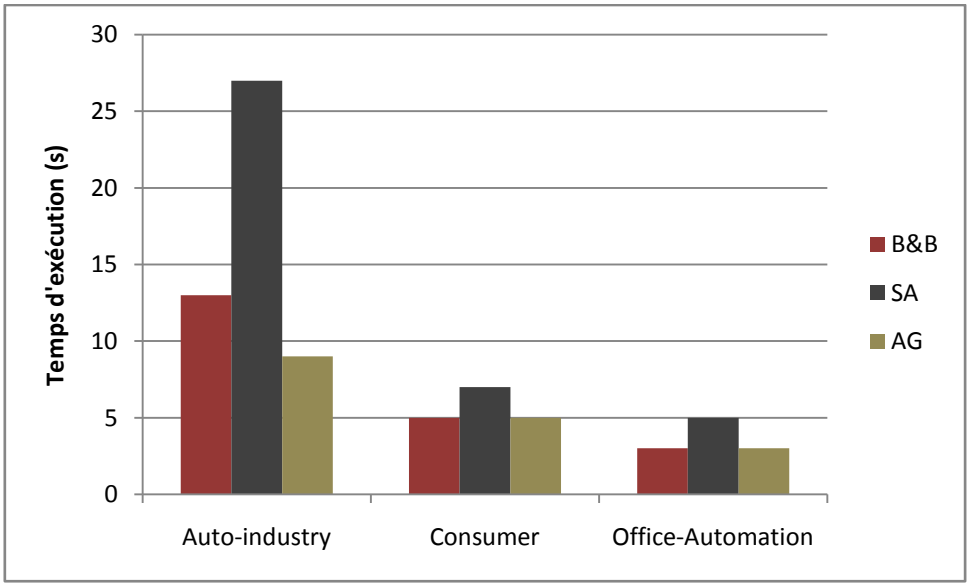


Figure 6.11 Temps d'exécution pour 3 benchmarks (E3S)

Le temps d'exécution de l'algorithme génétique dépend de divers facteurs tel que la taille de la population, la probabilité de mutation, et la condition de terminaison, qui est peut être efficace à condition que les paramètres soient bien choisis. En utilisant les paramètres cités dans le tableau 6.2. Nous obtenons le temps d'exécution pour chaque benchmarks (figure 6.11). On constate que l'algorithme génétique et l'algorithme B&B donnent presque le même temps d'exécution (temps de recherche) pour le benchmark Consumer et Office-automation, cependant, B&B est 2 fois plus rapide que SA pour le benchmark Auto-industry, et l'AG est 3 fois plus rapide que SA pour ce même benchmark. Néanmoins, l'algorithme génétique est meilleur que l'algorithme B&B lorsque le nombre de nœuds du réseau augmente.

6.7 Conclusion

Ce chapitre est complètement dédié à l'implémentation et l'évaluation de notre approche, nous avons présenté les différents outils utilisés pour implémenter notre approche, puis nous avons vu les différents modules que nous avons implémentés, ainsi que leurs paramètres. Ensuite, plusieurs tests ont été effectués sur différents benchmarks afin de bien évaluer les performances de notre approche.

CONCLUSION GENERALE ET PERSPECTIVES

Conclusion générale et Perspectives

Ce travail abord le problème de mapping d'applications sur un réseau sur puce qui est considéré comme un problème NP-difficile. Nous avons proposé une technique hybride afin de trouver le meilleur mapping. Cette technique est basée sur un algorithme de clustering qui exploite la propriété de distance du graphe d'application et celui de l'architecture et une méthode heuristique basée sur un algorithme génétique avec l'opérateur de croisement PBXA qui est une amélioration de l'opérateur de croisement à base de position vu dans le chapitre 3. Pour l'expérimentation, nous avons utilisé deux types de benchmarks, le premier type est généré par l'outil TGFF, et le deuxième type de benchmark ce sont des benchmarks E3S.

Nous considérons que notre solution donnent de bons résultats comparés à ceux obtenus par la méthode B&B et la méthode recuit simulé SA présentées dans [50]. En attendant des expérimentations et simulations plus poussées en utilisant d'autres benchmarks.

Parmi les objectifs futurs est de proposer des opérateurs génétiques plus efficaces pour améliorer la convergence de l'algorithme. Egalement sur la possibilité d'optimiser le mapping en agissant sur d'autres paramètres architecturaux tels que les stratégies de routage, la taille des buffers, etc.

Il serait aussi intéressant d'étudier et de mettre en œuvre d'autres algorithmes de mapping dans le réseau sur puce. Par exemple, la comparaison avec les algorithmes proposés par Harmanani et al [53] et Ascia et al [54]

Comme autre orientation des travaux futurs, nous avons l'intention d'utiliser un simulateur NoC pour mieux évaluer nos objectifs. Cela exige beaucoup plus de temps de simulation.

ANNEXE

Framework jMetal (java Metaheuritic)

1. Outils de développement

a) Télécharger et installer JDK 1.6 ou plus récent (utiliser le lien suivant pour le télécharger:
(http://www.oracle.com/technetwork/java/javase/downloads/index.html).

b) Télécharger et installer Eclipse Galileo pour Linux:
(http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR2/eclipse-jee-indigo-SR2-linux-gtk.tar.gz).

c) Télécharger le framework jMetal dans un répertoire local par exemple /home/linuxos/Downloads, à partir du lien suivant :
http://jmetal.sourceforge.net.

2. Etapes de développement

Nous décrivons brièvement comment compiler et exécuter les algorithmes développés avec jMetal en utilisant l'environnement de développement eclipse :

a) Décompresser le projet, cela peut être fait en tapant : tar zxf jmetal.tar.gz (à partir la ligne de commande).

b) nommer le répertoire où l'archive est décompressé JMETALHOME.

c) Lancer Eclipse.

d) Importer le projet java /home/linuxos/JMETALHOME de la manière suivante :

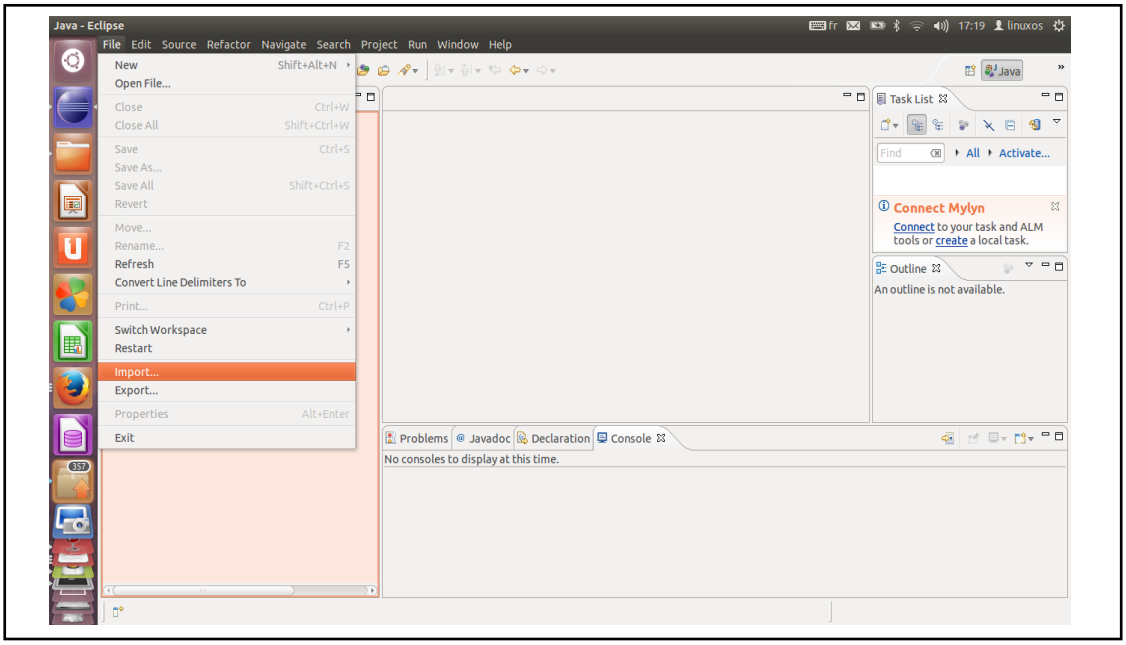
File → Import → General → Existing Projetct into WorkSpace → Browse

Cherchez l'emplacement où vous avez décompressé le fichier jmetal.tar.gz

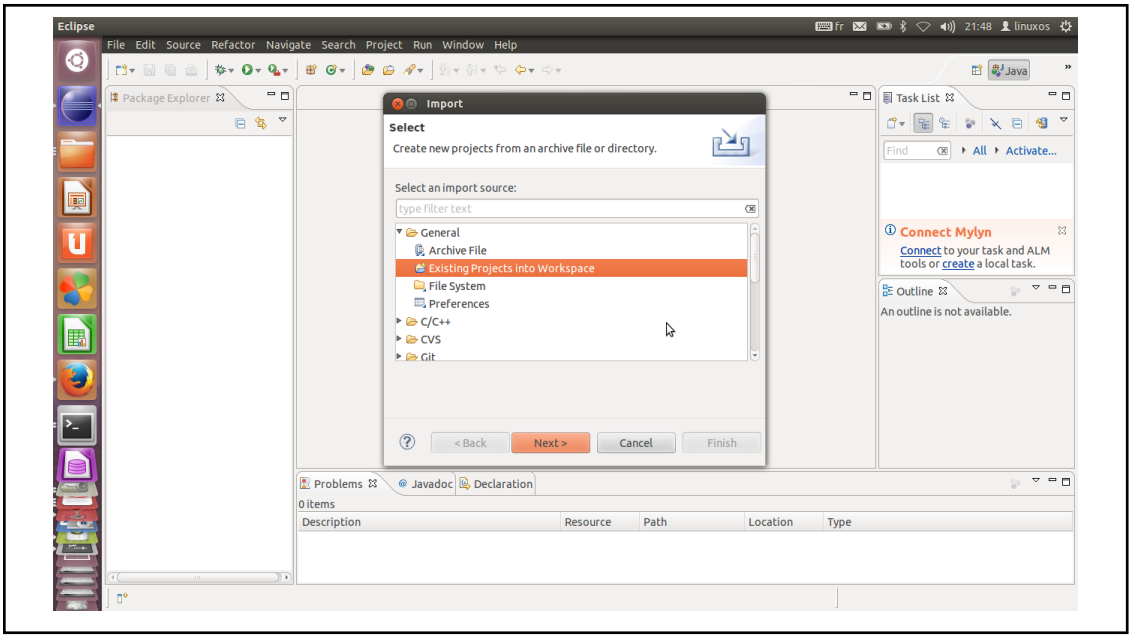
Dans notre cas c'est: /home/linuxos/JMETALHOME Sélectionner le répertoire

JMETALHOME et puis faire OK.

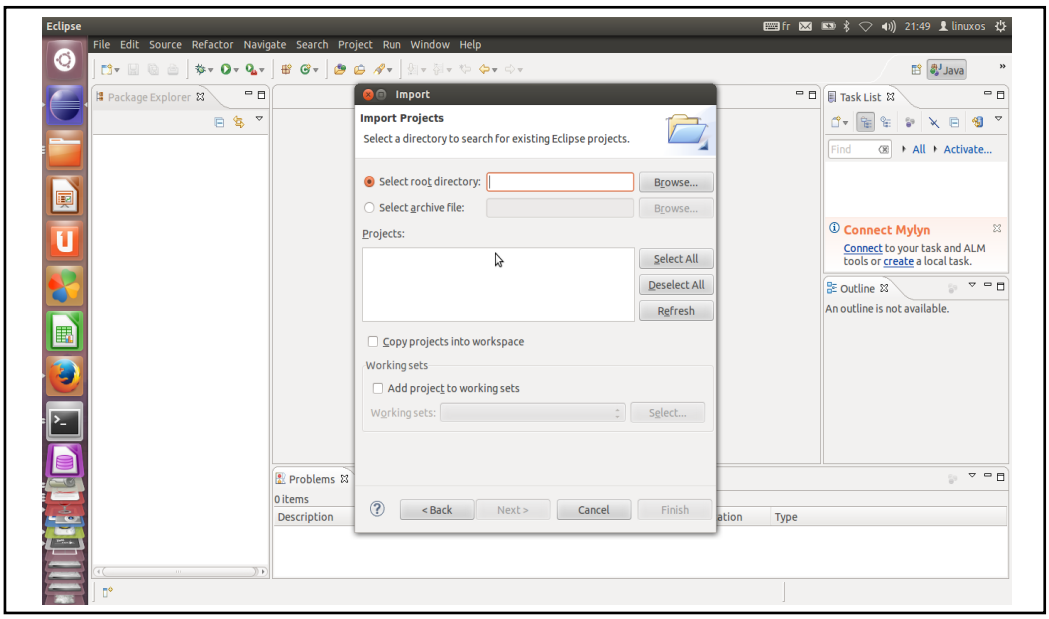
Les captures d'écran suivantes explicitent cette manipulation



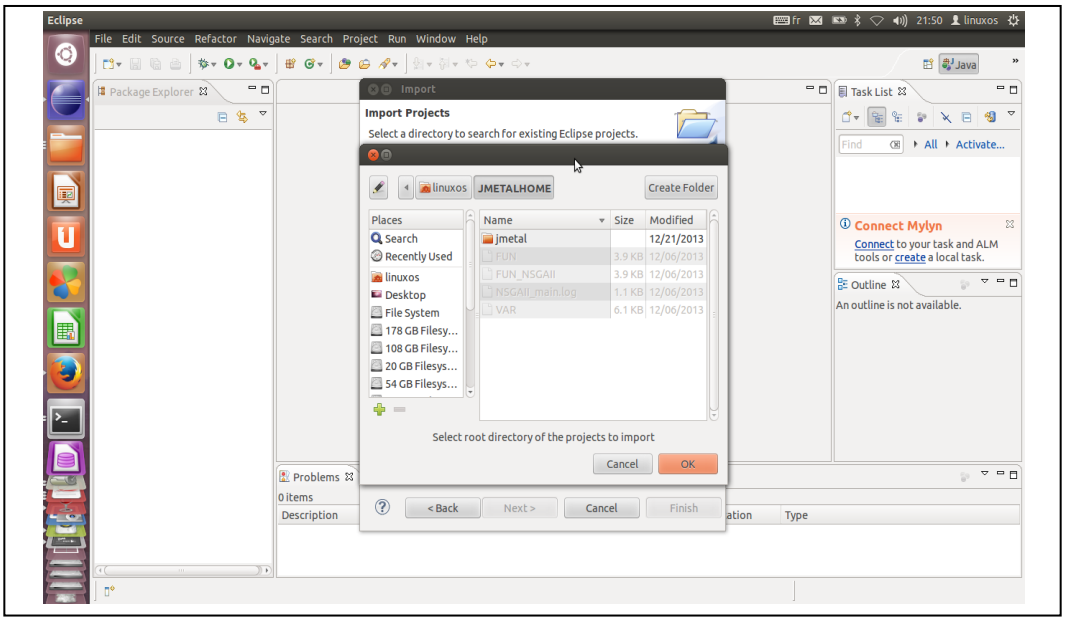
Cliquer alors sur le bouton File par la suite sur Import



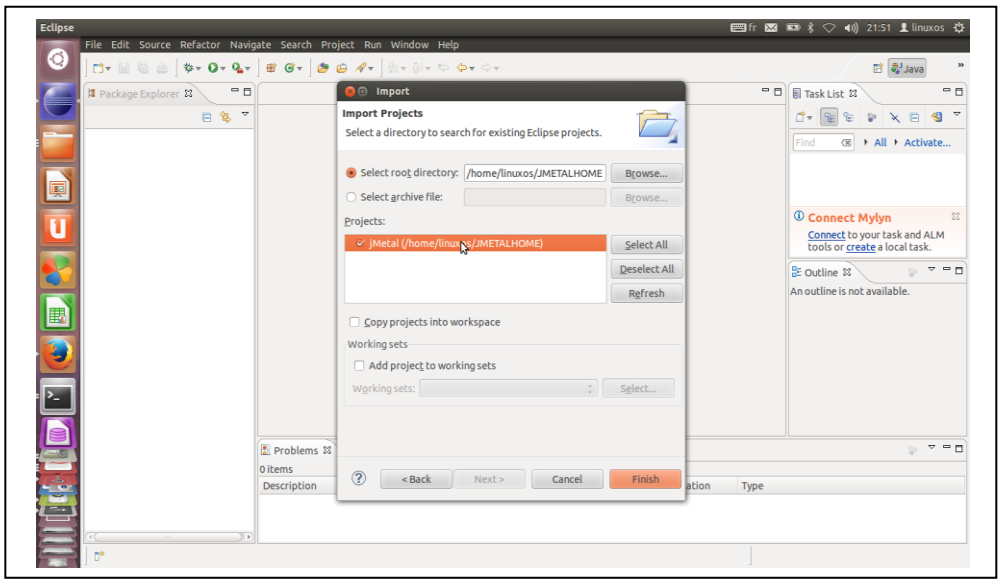
Cliquer alors sur le bouton ► devant General ⟶ Cliquer sur Existing Projects into Workspace



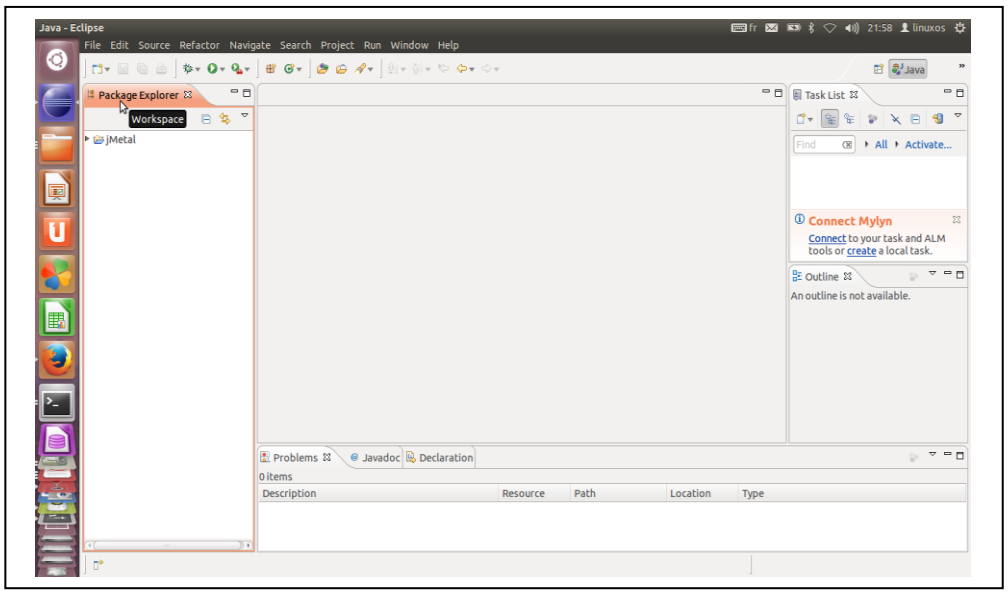
Cliquer alors sur le bouton Browse pour chercher le projet à importer.



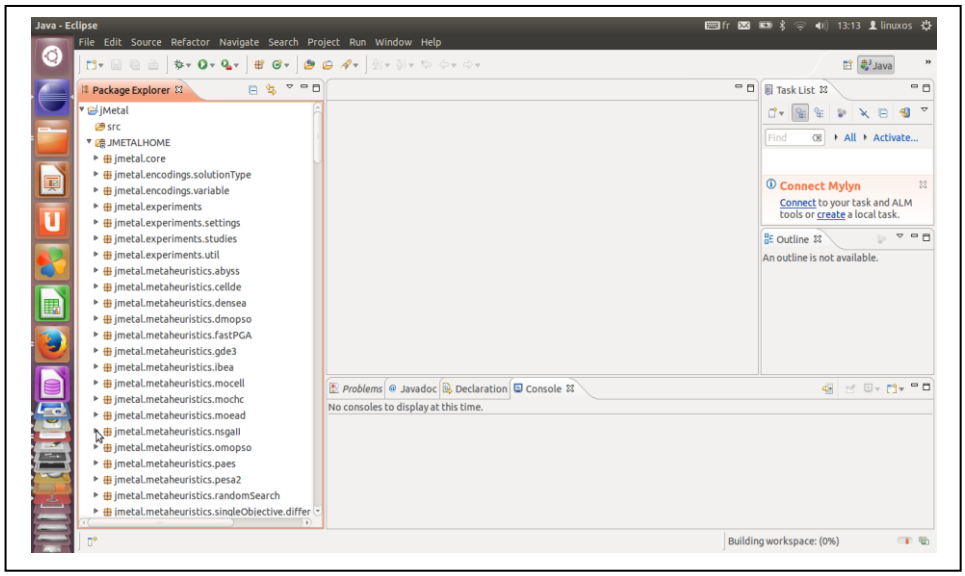
Sélectionner le répertoire JMETALHOME et puis faire OK.



Cliquer sur le bouton Finish



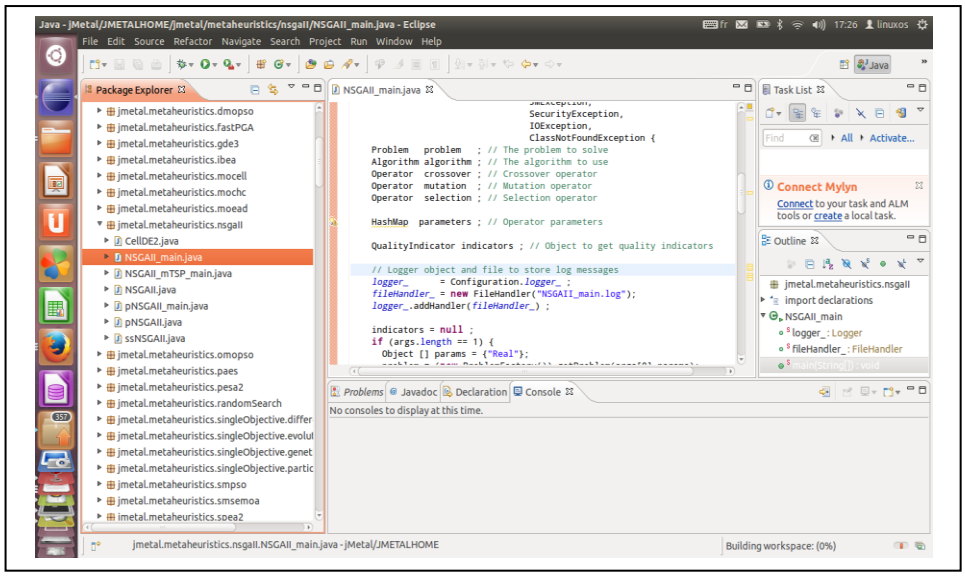
Vous devriez voir cette fenêtre, mais afin de visualiser les packages de ce projet cliquer sur « ►» devant jMetal



Vous devriez voir cette fenêtre

Configuration d'un algorithme

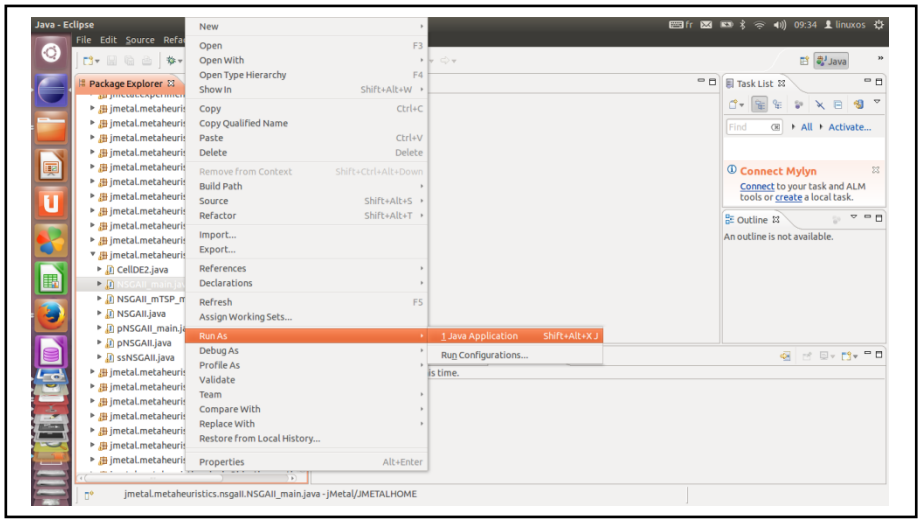
Nous utilisons NSGA-II comme un exemple. Pour configurer l'algorithme, cliquer sur le package jmetal.metaheuristics.nsgaII ⟶ cliquer sur NSGAII_main.java et modifier le fichier en fonction de vos préférences.



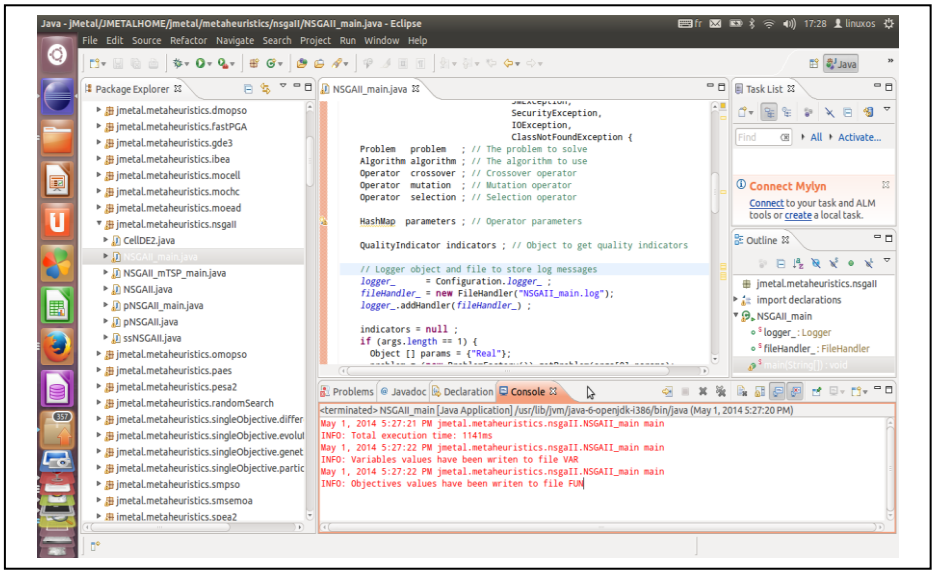
Compilation de l'algorithme

Afin de compiler l'algorithme, nous procédons comme suite:

- Cliquer sur le bouton droit sur le fichier NSGAII_main.java
- Sélection Run As
- Cliquer sur Java Application



Vous pourrez voir le résultat de votre programme dans la console Eclipse



Les opérateurs:

Les techniques méta-heuristiques sont basées sur la modification ou la création de nouvelles solutions à partir de celles qui Existent déjà, par le biais de l'application des différents opérateurs. Par exemple, les algorithmes évolutionnaires utilisent les opérateurs de croisement, mutation, et sélection pour modifier les solutions. Dans l'outil jMetal, toutes les opérations qui modifient ou génèrent des solutions (ou sous ensembles) sont hérités d'une classe "Operator".

Pour illustrer la façon dont les opérateurs sont utilisés et mis en œuvre dans l'outil jMetal en prenant comme exemple l'opérateur de croisement SBX. Avec les paramètres suivants : la probabilité de croisement et l'indice de distribution.

```
. . .
// Etape 1 : création d'un map pour les paramètres de l'opérateur
HashMap parameters = new HashMap ( ) ;

// Etape 2 : configure  l'opérateur
parameters.put ( "probability"  , 0 . 9 ) ;
parameters.put ( "distributionIndex" , 20.0 ) ;

// Etape 3 : créer l'opérateur
crossover = CrossoverFactory.getCrossoverOperator ( "SBXCrossover" , parameters ) ;

// Etape 4 : ajouter l'opérateur à l'algorithme
algorithm.addOperator ( "crossover" , crossover ) ;
 . . .
```

```
. . .
public class MOMetaheuristic extends Algorithm {
. . .
Operator crossover ;
crossover = operators.get ( " crossover " ) ; // L'algorithme doit obtenir l'opérateur précédemment créé
. . .
Solution [ ] parents = new Solution [ 2 ] ;
parents [ 0 ] = ( Solution ) selectionOperator.execute ( population ) ; // L'opérateur peut être exécuté après
  l'appel de la méthode execute () avec les paramètres correspondants
parents [ 1 ] = ( Solution ) selectionOperator.execute ( population ) ;
Solution [ ] offSpring = ( Solution [ ] ) crossoverOperator.execute ( parents ) ;
. . .
} // MOMetaheuristic
```

Les deux solutions précédemment obtenues, généralement après l'application d'un opérateur de sélection

REFERENCES BIBLIOGRAPHIQUES

Références Bibliographiques

[1] Jean-Baptiste, *"Architecture de simulation distribuée temps réel"*, thèse de doctorat, Université de Toulouse, janvier 2010.

[2] Emannuel Simeu, *"Test et surveillance intégré des systèmes embarqués"*, habilitation à diriger des recherches, université de Groneble, septembre 2005.

[3] F. Vahid and T. Givargis, *"Embedded System Design: A Unified Hardware/Software Introduction"*, John Wiley & Sons, ISBN: 0471386782, 2002..

[4] K. M. Krishna, R. Akl, A. R. Hurson, *"Real-Time Systems: An Introduction and the State-of-the-Art"*. Wiley Encyclopedia of Computer Science and Engineering, 2008.

[5] Vincent BARICHARD, *"Approches hybrides pour les problèmes multiobjectifs"*, Thèse de Doctorat, Ecole doctorale d'angres, 2003.

[6] I. Maalej, G. Gogniat, M. Abid, *"Jean Luc Philippe: Interface design approach for system on chip based on configuration"*. International Symposium on Circuit and Systems (ISCAS). Vol. 5, pp. 593-596, 2003:

[7] C. Tanougast, S. Jovanovic, F. Monteiro, C. Diou, A. Dandache, *"Initiation à la modélisation et co-simulation comportementale C-VHDL d'un réseau de communication sur Puce (Network on Chip)"*, 10es Journées Pédagogiques du CNFM, 26-28 novembre 2008, Saint-Malo.

[8] Qurat-ul-Ain Malik, M. Aqeel Iqbal, *"Next Generation High Speed Computing Using System-on-Chip (SoC) Technology"*, International Journal of Advancements in Technology (IJACT), Vol.2, No.2 , pp. 243-256, 2011.

[9] C. FLIPO-DHAENENS, *"Optimisation Combinatoire Multi-Objectif : Apport des Méthodes coopératives et contribution à l'extraction de connaissances"*, HDR, Université des Sciences et Technologies de Lille, Octobre 2005.

[10] A. A. Jerraya et W. Wayne, *"Multiprocessor Systems-on-Chips"*, Elseviere, United States of America: Morgan Kaufmann, 2005.

[11] Alexandre CHAGOYA-GARZON, *"Synthèse des communications dans un environnement de génération de logiciel embarqué pour des plateformes multi-tuiles hétérogènes"*, pour obtenir le grade de Docteur; Ecole Doctorale d'«Electronique, Electrotechnique, Automatique et Traitement du Signal» Institut National Polytechnique de Grenoble, Décembre 2010.

[12] H. Lee et al., *"On-Chip Communication Architecture Exploration: A Quantitative Evaluation of Point-to-Point, Bus, and Network-on-Chip Approaches"*, ACM Transactions on Design Automation of Electronic Systems, vol. 12, no. 3, Article 23, 2007.

[13] Wen-Chung Tsai, Ying-Cherng Lan, Yu-Hen Hu, and Sao-Jie Chen, *"Networks on Chips: Structure and Design Methodologies. Journal of Electrical and Computer Engineering"*, pp. 1-15, 2012, doi:10.1155/2012/509465.

[14] J.A. Rowson, A. Sangiovanni-Vincentelli, *"Interface-Based Design, In Proceeding of the 34th Design Automation Conference"*, pp. 178-183, 1997.

[15] Nicola Concer, *"Design and Performance Evaluation of Network-on-chip Communication Protocols and Architectures"*, PhD thesis, University of Bologna, Padova, March 2008.

[16] Hazem MOUSSA, *"Architecture de réseau sur puce pour décodeurs canal multiprocesseurs"*, Ecole National Supérieur des Télécommunications de Bretagne, Décembre 2009.

[17] Tobias BJERREGAARD, Shankar MAHADEVAN, *"A Survey of research and Practice of Network of Chip"*, Technical University of Denmark, ACM Computing Surveys, vol. 38, no. 1, pp.1-51, March 2006.

[18] T. N. K. Reddy, A. K. Swain, J. K. Singh and K. K. Mahapatra, *"Performance Assessment of Different Network-on-Chip Topologies"*, International Conference on Devices, Circuit and System (ICDCS), 2014.

[19] Jie Chen, Cheng Li and Paul Gillard, *"Network-on-Chip (NoC) Topologies and Performances: A Review"*, Memorial University of Newfoundland, 2008.

[20] Naveen Choudray, *"Migration of on-Chip Network from 2 Dimensional Plan to 3D Plan"*, International Journal of Engeering and Advanced Technology, vol. 2, no. 4, pp. 516-519, April 2013.

[21] Suyog K.Dahule, Sagar Soitkor, Vaishali Ingle, *"A Review Paper on Different Switching Techniques in NOC Router Architecture"*. International Journal of Advanced Research in Computer Science and Software Engineering. vol.4, no. 11, pp. 227-229, November 2014.

[22] J. J. Murilla, "*HW-SW component for parallel embedded computing on NoC based MPSoCs*", PhD thesis in computer science, University of Barcelona, 2009.

[23] E.Fleur, *"Communication de groupe : du parallélisme au ad hoc"*. Habilitation diriger des recherches de L'INSA de Lyon et de l'université Claude Bernard – lyon 1, 2002.

[24] M. Antagoziyev, *"Routing Algorithms for on Chip Network"*, Master of Science in Electrical and Electronics Engineering, Middle East Technical University, 2007.

[25] Maurizio Palesi, Masoud Daneshtalab, *"Routing Algorithms in Network-on-Chip"*, Springer New york Heidelberg Dordrecht London, ISBN 978-1-4614-8274-1(eBook), 2014.

[26] Nadia SMAIRI, *"Contribution à l'Optimisation par Essaim Particulaire : adaptation de tribes à l'optimisation multiobjectif"*, Thèse doctorat, Université de Manouba, Décembre 2013.

[27] Sergio Luciano AVILA, *"Optimisation multiobjectif et analyse de sensibilité appliquées à la conception de dispositifs"*, Thèse de doctorat, Université des sciences et technologie de Lyon, février 2006.

[28] Merdjaoui Brahim, *"Espace des objectifs **:** ensemble image de l'espace de recherche, déterminé par toutes les valeurs possibles des fonctions objectif"*, Mémoire de Magistère, université de Boumerdes, octobre 2006.

[29] Aroui Abdelkader, *"Ordonnancement multi-objectif d'applications intensives sur architectures régulières embarquées"*, Mémoire pour obtenir le diplôme magistère en informatique, Université d'oran ES-SENIA, Février 2012.

[30] Boumaaza Farid, *"Mapping multi-objectifs d'applications intensives sur architecture MPSoC"*, Mémoire pour obtenir le diplôme magistère en informatique, Université d'oran, Janvier 2013.

[31] Arnaud ZINFLOU, *"Système interactif d'aide à la décision basé sur des algorithmes génétiques pour l'optimisation multi-objectifs"*, Université de Québec à chicoutimi, Mémoire de maitrise, 2004.

[32] Julien LEMESRE. *"Méthodes exactes pour l'optimisation combinatoire multi-objectifs"*, Doctorat, Université des Sciences et Technologies de Lille, nov 2005.

[33] Metroplois, N., A.W. Rosenbluth, M.N Rosenbluth, A.H. Teller, and E.Teller, *"Equation of state calculation by fast computing machines"*, J. Chem. Phys. , Vol. 21, No. 6, pp. 1087 – 1092, 1953.

[34] Abdesslem LAYEB, *"Utilisation des Approches d'Optimisation Combinatoire pour la Vérification des Applications Temps Réel"*. Thèse de Doctorat, Université de Constantine, 2010.

[35] Souquet Amédée et Radet Francois-Gérard, *"Algorithmes génétiques"*, TE de fin d'année, 2004.

[36] D. E. Goldberg, *"Genetic Algorithms in search, optimization machine learning"*, Pearson Education, 1989.

[37] D. E. Goldberg, K. Deb, *"A Comparative Analysis of Selection Schemes Used in Genetic Algorithms"*, Foundations of Genetic Algorithms. pp. 69-96, Morgan Kaufamm, San Mateo, California, 1991.

[38] Rabah TAOUCHE, *"Prévision du comportement mécanique d'alliages biphases par algorithme génétique et réseaux de neurones",* Thèse de Doctorat, Université de Constantine, 2010.

[39] M-C. Portmann, *"Genetic algorithm and scheduling: a state of the art and some propositions"*, In Proceedings of the workshop on production planning and control, Mons, Belgium, pp. 1-14, 1996.

[40] K. Deb, *"Multi-Objective Optimization Using Evolutionary Algorithms"*, John Wiley & Sons LTD, 2001, ISBN: 0-471-87339-X.

[41] Mohamed Anouar Taleb, *"Parallélisme d'un algorithme pour le problème d'ordonnancement sur machine unique avec temps de réglages dépendants de la séquence"*, mémoire de maitrise en informatique, Université de Québec à chicoutimi, Avril 2008.

[42] D. Goldberg, R. Lingle, *"Alleles, Loci and the traveling salesman problem"*, Proceedings of the 1st International Conference on Genetic Algorithms, pp. 154-159, 1985.

[43] Mais Hadj-Rachid, Christelle Bloch , Wahiba Ramadane-Cherif, Pascal Chatonnay, *"Différents opérateurs évolutionnaires de permutation : sélections, croisements et mutations"*, Laboratoire d'Informatique de l'université de Franche-Comte EA 4269 (LIFC), juillet 2010.

[44] Ailsa H Land and Alison G Doig, *"An automatic method of solving discrete programming problems"*, Econometrica: Journal of the Econometric Society, pp. 497-520, 1960.

[45] László Pál, *"Global optimization algorithms for bound constrained problems"*, PhD Thesis. University of Szeged, 2010.

[46] Abou El Hassan BENYAMINA, *"Ordonnancement hiérarchique multi-objectif embarquées d'applications intensives"*, Thèse de doctorat, Université d'Oran, 2008.

[47] J. Hu and R. Marculescu, *"Communication and task scheduling of application specific networks-on-chip"*, IEEE Proceedings - Computers and Digital Techniques, vol. 152, no. 5, pp. 643, 2005.

[48] DELORME Julien, "Méthodologie de modélisation et d'exploration d'architecture de réseau sur puce appliqué aux télécommunications'', Thèse de doctorat, Institut national des sciences appliqués de Rennes, Février 2007.

[49] D.Shin and J.Kim, *"Power-aware communication optimization for networks-on-chips with voltage scalable links"*, In CODES+ISSS '04 : Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp. 170-175, September 2004.

[50] J. Hu and R. Marculescu, *"Energy-aware mapping for tile-based NoC architectures under performance constraints"*, in Proceedings of the Conference on Asia South Pacific Design Automation (ASP-DAC), Kitakyushu, Japan, pp. 233-239, January 2003.

[51] F.Vardi, S.Saeidi, A. Khademzadeh, *"Crinkle: A heuristic mapping algorithm for network on chip"*, IEICE Electronics Express, vol. 6, no.24, pp. 1737-1744, 2009.

[52] Y. Umit, Ogras and J. Hu, *"Key Research Problems in NoC Design : A Holistic Perspective"*, in Proceeding of the Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp. 69-74, Sept. 19-21, 2005.

[53] H. M. Harmanani and R. Farah, *"A Method for Efficient Mapping and Reliable Routing for NoC Architecture with Minimum Bandwidth and Area"*, IEEE Northeast Workshop on Circuit and System TAISA Conference, pp. 29-32, 22-25 June 2008.

[54] G. Ascia, V. Catania, and M. Palesi, *"A Multi-Objective Genetic Approach to Mapping Problem on Network-on-Chip"*, JUCS, vol. 12, no. 4, pp. 370-394, 2006.

[55] P. Sahu, N. Shah, K. Manna, and S. Chattopadhyay, *"A new Application mapping algorithm for mesh based Network-on-Chip design"*, in proceedings of India Conference (INDICON), pp. 1-4, 2010.

[56] B. Kernighan and S. Lin, *"An Efficient Heuristic Procedure for Partitioning Graphs"*, Bell System Technical Journal, vol. 49, no. 2, pp 291-307, 1970.

[57] L. Zonghai, X. Lai, A. Jantsch, *"Cluster-based simulated annealing for mapping cores onto 2D mesh networks on chip"*, In Proceeding of the 11th IEEE Workshop on Design and Diagnostics of Electronic Circuit and System, vol., no., pp.1-6, 16-18 April, 2008.

[58] C. Radu, *"Optimized Algorithms for Network-on-Chip application mapping"*. PhD Thesis, University of Sibiu, 2011.

[59] TGFF : Task Graphs For Free, http://ziyang.eecs.umich.edu/~dickrp/tgff/

[60] R. Dick, *"Embedded System Synthesis Benchmarks Suite"* http://ziyang.eecs.umich.edu/~dickrp/e3s/, 2011.

[61] J.J Durillo, A.J Nebro, F. Luna, B.Dorronsoro, and E. Alba, *"jMetal : A java Framework for developping multi-objective optimization metaheuristics"*, Technical Reports ITI-2006-10, University of Malaga, 2006.