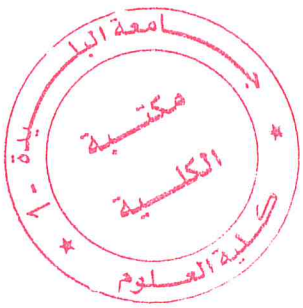
**Université Saad Dahleb Blida1**




**Faculté des sciences**

**Département d'Informatique**

Mémoire présenté par :

Souna Baya

En vue d'obtenir le diplôme de Master

**Domaine** : Mathématique et informatique

**Filière** : Informatique

**Spécialité** : Génie Systèmes Informatiques

**Sujet :** Placement optimal ou pré-optimal des composants d'un système sur puce implémenté par une architecture 3D.

**Soutenu le :** 24/06/2015

**Devant le jury :**

| | | |
|---|---|---|
| M. | Toubaline | présidente. |
| M. | Cherfa | rapporteur. |
| Mr. | Kameche | examinateurs. |
| Mr. | Mahdoum | promoteur. |

# Résumé

Avec l'avènement du développement technologique, il est actuellement possible d'intégrer des millions de transistors sur une seule et même puce. Ceci a donné naissance à des systèmes électroniques communément appelés les Systèmes sur Puce (SoC).

Les systèmes actuels se caractérisent par un fort degré de communication, les architectures précédentes à base de bus ou de barres croisées ne peuvent être utilisées à cause de la largeur de bande limitée qu'elles offrent. Il est donc nécessaire d'opter pour un réseau sur puce répondant aux contraintes de largeur de bande, de surface et de consommation de puissance.

Le présent sujet consiste alors à placer les composants du système en vue de concevoir un réseau sur puce dans le cadre des architectures 3D. Ce placement est un problème d'optimisation multicritères dont la solution exacte ne peut être obtenue en un temps polynomial. Il s'agira donc d'opter pour une heuristique afin d'obtenir une solution intéressante en un temps CPU raisonnable. La qualité de la solution sera fortement dépendante de l'emplacement de ces composants sur l'architecture 3D.

Mots clés : SoC, communication, réseau sur puce, architectures 3D, placement, optimisation, heuristique.

# Abstract

With the advent of technology scaling, it is now possible to integrate millions of transistors on the same chip. This has led to design systems on chip (SoC). Because such systems feature a high degree of communications, both bus or cross-bare-based architectures do not efficiently deal with those systems because of the limited bandwidth they offer. Thus, one has to design a Network on Chip (NoC) such that to meet the bandwidth, area and power constraints.

The present work aims at placing the IP components on a 3D-architecture, wich is a task among others in designing on NoC. Such a placement is a multi-criteria optimizationproblem whose exact solution can not be obtained in a polynomial time. Therefore, one has to develop a heuristic-based method so that to obtain a good solution in reasonable CPU time. Note that the quality of the solution will be strongly dependent on the location of these components on the 3D-architecture.

Keywords : SoC, communication, NoC, 3D-architecture, placement, optimization, heuristic.

شهد العصر الحالي ثورة هائلة في عالم التكنولوجيا، حيث أصبح من الممكن، في وقتنا هذا، دمج الملايين من المقاحل (الترانزستورات) على شريحة واحدة . هذه الشرائح تسمى أنظمة إلكترونية على رقاقة. تتميز هذه الأنظمة الإلكترونية بدرجة عالية من التواصل بين مكوناتها، مما جعل البنيات القديمة التي كانت تعتمد عليها مثل الموصل المشترك أو الموصلات المتقاطعة غير مجدية بسبب محدودية عرض النطاق الترددي التي تقدمها. و لذلك أصبح من الضروري أن نستعمل تكنولوجيا جديدة وهي طريقة التواصل من خلال الشبكة التي تقدم عدة مزايا و فوائد فهي تجعل المساحة المستعملة أقل، و استعمال الطاقة كذلك.

هذا الموضوع سيعمل على مشكلة تحديد أماكن وضع المكونات في مكانها لتصميم رقاقة خاصة تعتمد على الشبكات و البنيات ثلاثية الأبعاد. هذه المشكلة تعتبر مشكلة معقدة وسنحاول حلها عن طريق وسائل تقريبية. لان الحصول على الحل عن طريق طريقة نموذجية. بالطبع سيكون لمواقع المكونات تأثير كبير على نوعية التصميم.

Il me serait impossible de citer nommément toutes les personnes qui m'ont aidé, encouragé et soutenu afin que ce travail puisse voir le jour. Que toutes ces personnes trouvent ici l'expression de ma sincère reconnaissance.

Je tiens tout d'abord à remercier très chaleureusement tous les enseignants qui ont contribué de près ou de loin à ma formation.

Je tiens également, à remercier Mr. Mahdoum Ali pour avoir assuré l'encadrement de ce projet, qui n'a pas toujours été de tout repos. Je le remercie pour nos séances de travail agréables et fructueuses, ses remarques pertinentes, mais aussi pour son écoute et son discours bienveillants.

Il m'est difficile de ne pas remercier Mr. Mahdoum Ali tout particulièrement de m'avoir donné l'opportunité de travailler sur un projet d'une telle envergure et de m'avoir aidé à comprendre et à exposer le plus justement possible ce travail.

Je me dois de mentionner la précieuse et totale collaboration que j'ai reçu au sein de CDTA, de part les moyens mis à ma disposition et d'autres parts l'aide et le support apporté par l'ensemble des employés et des cadres.

Je remercie vivement Mesdames et Messieurs les membres du jury d'avoir accepté d'évaluer ce travail.

Ces remerciements ne seraient pas complets sans une pensée sincère à mes amies proches et ma famille et plus particulièrement mes parents pour leur soutien constant.

Enfin, je ne pourrai clore ces remerciements sans remercier du fond du cœur ma moitié pour son soutien, sa patience et son écoute. Cette thèse est le fruit d'un effort vécu à deux dans la complexité de l'adéquation entre réussite personnelle et professionnelle. Tu as toujours su être là quand et comme il le faut : tu es merveilleux ... !

*SOUNA*

# DEDICACES

Je commence par rendre grâce à dieu « soubhânahou wa ta'âlâ» qui m'a guidé sur le droit chemin tout au long du travail, m'a inspiré les bons pas et les justes réflexes et m'a donné la capacité d'écrire et de réfléchir, la force d'y croire, la patience d'aller jusqu'au bout. Sans sa miséricorde, ce travail n'aura pas abouti. Merci Allah !

Avec tout mon amour éternel et avec l'intensité de mes émotions. Je dédie ce modeste travail à :

Mon très cher père « Hacene » qui m'a beaucoup aidé pour accomplir ce travail. il est très difficile de choisir les termes adéquats pour t'exprimer mon amour et mon respect que Dieu te garde en bonne santé pour nous.

Mon adorable mère « Wassila » voudrais-je des profondeurs de mon amour et de ma reconnaissance te dédier ce mémoire, dont tous les mots que je pourrais utiliser seraient insuffisants pour te témoigner l'amour que je te porte.

Mon « CR Abdelkader » qui a cru en moi et su apprécier mes idées. Merci d'avoir parcouru avec moi le chemin menant à ce stade et de m'avoir encouragé à aller toujours plus loin.

Mon cher frère « Chakib » et mes sœurs « Yassmine », « Ibtisseme » et « Hadjer ». En témoignage de mon amour éternel que Dieu vous garde, vous protège et vous offre une vie pleine de joie.

Mes copines « Dalila », « Djamila », « Lamia », « Aicha » et à leurs familles.

Toutes les personnes que j'aime, à ceux qui m'ont soutenue dans les moments difficiles, en témoignage des liens qui nous unissent, je vous dédie ce travail, je vous souhaite beaucoup de succès, de courage et de bonheur tant professionnel que familial.

**Baya**

# Liste des symboles

ASIC Application Spécific Integrated Circuit = HW (circuit intégré pour application spécifique)

CAD Computer Aided Design (Conception assistée par ordinateur (CAO))

CI Circuit Intégré

FPGA Field Programmable Gate Array (réseau de portes programmables)

HW Hard Ware

IP Intellectual Property (bloc IP ou « IP core » ou Composant virtuel, propriété intellectuelle)

IR Interface Réseau

MIV Monolithic Inertier Via

MOS Métal Oxyde Semi-conductor

NOC Network On Chip

SoCs Systems on Chip = = HW et SW (Système sur puce)

SW Soft Ware

TSVs Through Silicon Via

# Table des figures

# Sommaire

# Introduction générale

# Contexte général

L'évolution rapide de la technologie informatique et l'émergence de nouvelles applications dans plusieurs domaines (télécommunication, aéronautique, automobile, ...) avec des contraintes fonctionnelles de plus en plus complexes et critiques (puissance de calcul, consommation d'énergie, surface, embarquement) et non fonctionnelles (temps de mise sur le marché, rétrécissement de la durée de vie du produit, coût, . . .) a conduit à l'augmentation de la capacité d'intégration des circuits intégrés et de leurs architectures afin de répondre à ces exigences. Ceci a donné naissance à des systèmes électroniques composés de millions de transistors sur une seule et même puce, communément appelés les Systèmes sur Puce ou SoC.

Outre cet avantage, les délais des portes logiques deviennent de plus en plus courts à cause de la commutation rapide des transistors due à des tensions de seuil des transistors de plus en plus réduites. A côté de ces avantages, il existe malheureusement de nombreux problèmes auxquels les concepteurs doivent y faire face. Nous nous contentons uniquement de citer le problème en relation avec le contexte de ce projet. Les courbes de l'ITRS (The International Technology Roadmap for Semiconductors) concernant les délais des portes logiques et des interconnexions montrent clairement que le premier type de délais ne cesse de s'améliorer au fur et à mesure que la longueur du canal du transistor se réduit. Malheureusement, ceci n'est pas le cas pour les interconnexions : la communication devient de plus en plus lente avec le développement technologique. Bien que des solutions partielles aient été trouvées telles que l'insertion d'amplificateurs, le remplacement de l'Aluminium par du Cuivre (IBM 2001), le problème se pose toujours.

En effet, les systèmes actuels se caractérisent par un fort degré de communication. Ainsi les solutions telles que le bus partagé ou point à point trouvent leurs limites en termes de bande passante et d'extension à mesure que le nombre d'éléments communicants augmente. C'est dans ce contexte que le concept des Réseaux sur Puce ou NoC a vu le jour. Ce paradigme d'interconnexion s'inspire des réseaux informatiques classiques et offre une structure de communication évolutive et flexible répondant aux contraintes de largeur de bande, de consommation de puissance et de surface. Plusieurs méthodologies de conception ont été proposées, dont les principales sont celles reposant sur les Fat Trees, les Mesh Networks et d'autres variantes de ces méthodologies. Dans le cadre d'un projet mené au CDTA, une autre méthodologie [17] a été proposée et pour laquelle des tâches la réalisant sont en cours de développement. Dans le cadre de ce projet, l'une des tâches porte sur le placement des composants sur une architecture 3D. Cette tâche est l'objet du travail décrit dans ce mémoire.

La phase de placement est l'une des phases de conception des NoCs, dont le résultat a un impact direct sur les performances du système. Elle consiste à placer chaque élément de l'application sur l'architecture matérielle qui compose la puce afin de minimiser certains critères tel que la surface la consommation d'énergie et le temps de calcul. C'est dans ce cadre que s'insère notre travail. Il consiste à étudier différentes techniques de placement sur une architecture SoC 3D basée NoC et implémenter une de ces techniques qui semble la plus adaptée à notre problème.

## Problématique et objectifs du projet

Les caractéristiques d'un réseau sur puce en termes de largeur de bande, de surface et de consommation de puissance sont fortement dépendantes de l'emplacement des composants (IPs) sur l'architecture. Ce problème se présente comme un problème d'optimisation non polynomial. Il doit être donc résolu moyennant une méthode à base d'heuristique ou de métaheuristique donnant une solution de qualité en un temps CPU raisonnable.

## Planification et conduite du projet

L'initiation de tout projet nécessite une phase de planification. Celle-ci permet de définir les tâches à réaliser, maîtriser les risques et rendre compte de l'état d'avancement du projet.

« Planifier optimise ainsi les chances de réussite d'un projet en améliorant la productivité grâce à une meilleure maîtrise de la qualité.» [48].

Pour garantir le bon déroulement du projet, tout en respectant les délais, nous avons élaboré une planification globale de conduite du projet. Le diagramme suivant décrit cette planification ainsi que l'ordonnancement prévu des phases du projet.

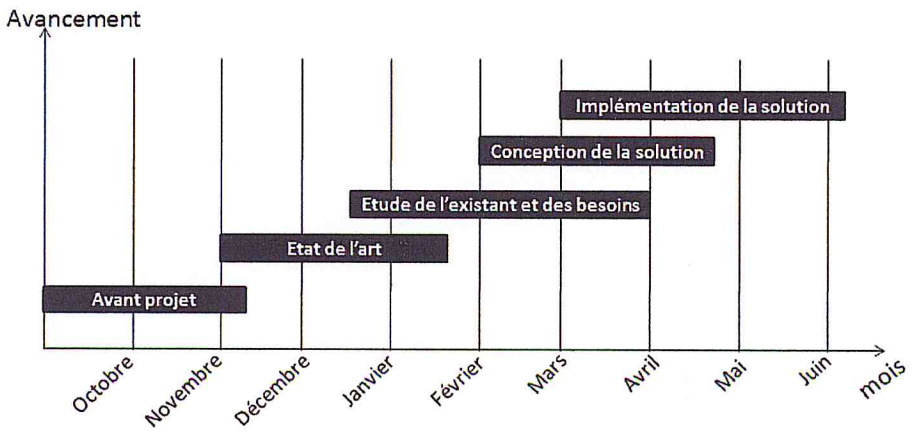



FIGURE 1 – *Planification et conduite du projet.*

Afin de présenter notre travail, le présent mémoire est organisé en deux parties et se présente comme suit :

La première partie présente les aspects théoriques du domaine des systèmes sur puce, en évoquant leurs définitions et les concepts de bases relatifs aux réseaux sur puce et à l'optimisation multicritères.

Dans la deuxième partie, nous présentons le travail réalisé au sein du CDTA à travers les trois chapitres suivants :

Le chapitre un présente une synthèse de la collecte des besoins, ainsi que la solution proposée.

Le chapitre deux décrit l'architecture globale de la solution, et cela en présentant les différents outils utilisés.

Le chapitre trois a pour vocation de présenter les tests que nous avons effectués et les résultats obtenus.

Une conclusion générale est proposée afin de synthétiser le travail réalisé et de citer les perspectives du projet.

# Première partie

# Etude bibliographique

# Chapitre 1

# Les systèmes sur puce SoCs

## 1.1 Introduction

Les systèmes sur puce font de plus en plus partie de notre vie quotidienne sans pour autant s'en rendre compte. Pour la mise en œuvre de ces systèmes les concepteurs se basent sur les communications au sein de la puce car cette partie devient de plus en plus lente avec l'avènement technologique. Dans ce chapitre nous allons définir ces systèmes et voir les moyens d'interconnexion dans ces structures.

## 1.2 Evolution des semi-conducteurs

Les composants électroniques ont envahi nos appareils domestiques, nos voitures, nos outils de communication et désormais tous les objets de nos vies privées et professionnelles [46]. Non contents de se multiplier, ils doivent être toujours plus petits, consommer moins d'énergie, dissiper le moins de chaleur possible et coûter moins cher. Pour les fabriquer, les industriels des semi-conducteurs ont perfectionné la technologie du silicium.

Les fabricants de semi-conducteurs sont confrontés à toujours plus de défis avec la multiplication des composants électroniques. Ils doivent concilier la course à la miniaturisation et à la sobriété énergétique avec une croissance exponentielle du coût des infrastructures de production.

Certes, ils sont aidés par la loi de Moore [37] qui affirme que le nombre de transistors par circuit est multiplié par deux tous les dix-huit à vingt-quatre mois à prix constants. Cette loi s'applique aux puces des circuits intégrés. Mais pour parvenir à augmenter la densité des transistors, il faut réduire leur taille et graver de plus en plus finement. La taille de la gravure est passée sous la barre du micron puis elle a progressivement baissé jusqu'à avoisiner l'échelle du nanomètre.

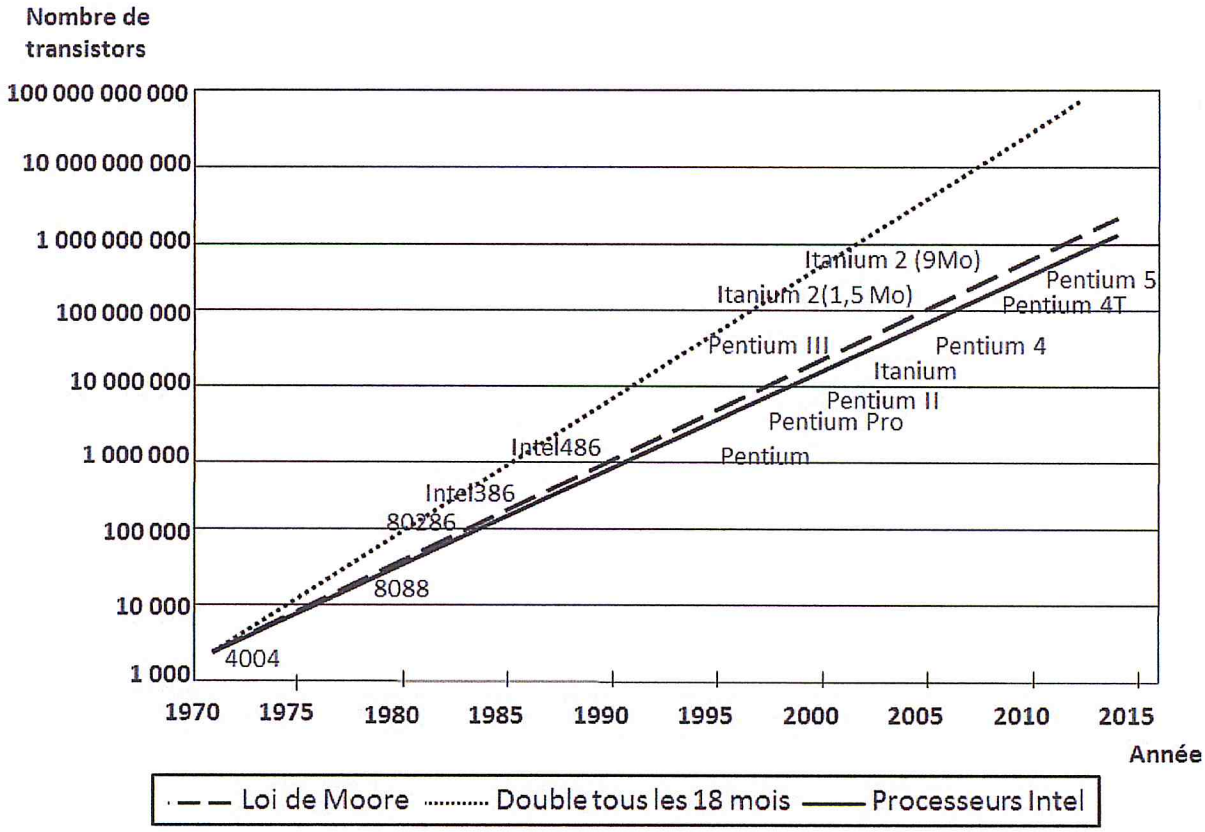



FIGURE 2 – *Courbe de densité d'intégration des transistors dans les processeurs Intel.*

Cette course à la miniaturisation pose un sérieux défi aux industriels. « Depuis 50 ans, toute l'industrie du semi-conducteur s'est bâtie sur des transistors qu'on appelle MOS, pour Métal Oxyde Semi-conductor. La combinaison des différents transistors permet de réaliser des circuits intégrés plus ou moins denses, plus ou moins complexes. Cette diminution de taille de plus en plus poussée a permis d'augmenter la densité d'intégration qui se définit comme le rapport du nombre de transistors par unité de surface.

- **Échelle d'intégration :** Les circuits intégrés peuvent être classés suivant le nombre de transistors qui les composent. On distingue ainsi les circuits :

| Catégorie | Taux d'intégration | Nombre de transistors (n) |
|---|---|---|
| SSI (small scale integration) | faible | quelques dizaines |
| MSI (medium scale integration) | moyen | quelques centaines |
| LSI (large scale integration) | grand | quelques milliers |
| VLSI (very large scale integration) | très grand | quelques millions |
| ULSI (Ultra large scale integration) | ultra grand | quelques milliards |

TABLE 1 – *Tableau comparatif entre les différentes catégories d'intégration.*

Ces distinctions ont peu à peu perdu de leur utilité avec la croissance exponentielle du nombre de portes. Aujourd'hui plusieurs centaines de millions de transistors (plusieurs

dizaines de millions de portes) représentent un chiffre normal (pour un microprocesseur ou un circuit intégré graphique haut de gamme).

## 1.3 Qu'est-ce qu'un SoC ?

Un système mono puce, appelé encore SoC ou système sur puce, désigne un système complet sur une seule pièce de silicium [16, 41], résultant de la cohabitation sur silicium de nombreuses fonctions déjà complexes en elles-mêmes : processeurs, DSP, mémoires, bus, convertisseurs, etc.

Néanmoins, une telle implémentation nécessite des méthodes de conception adéquates pour répondre aux spécifications du client. Ainsi, certains peuvent être conçus pour implémenter des applications temps réel qui nécessitent que chaque tâche s'exécute dans les délais [29]. Pour d'autres SoCs, c'est plutôt la consommation de la puissance qui a la priorité, ce qui amène à concevoir des systèmes ou circuits à faible consommation de puissance assujettis aux contraintes du temps et de surface [47]. L'estimation du temps et de la consommation de la puissance à un niveau de conception donné se fait par des outils de CAO industriels ou académiques [26, 27, 28].

Un circuit de telle complexité peut être implémenté par une partie opérative et une partie de contrôle.

- La partie opérative exécute le comportement de l'application selon un ordonnancement établi [31, 30, 34, 33, 32].
- La partie de contrôle gère le séquencement de l'exécution des différentes opérations (tâches). Un type de synthèse qui s'opère sur une partie de contrôle consiste à optimiser la taille des codes des états de la machine implémentant cette partie. Ceci, afin de diminuer le nombre de transistors et la longueur des interconnexions, ce qui est bénéfique pour la surface, la vitesse et la consommation de la puissance [25, 24]. Un exemple de partie de contrôle est indiqué dans (la figure 3). Les signaux de commande permettent de connecter/ déconnecter les ports des IPs aux interconnexions. La partie de contrôle mentionnée dans (la figure 3) joue le rôle d'interface du réseau dans la topologie du réseau proposée au CDTA. A cette interface s'ajoute une autre interface réglant le problème éventuel de métastabilité.

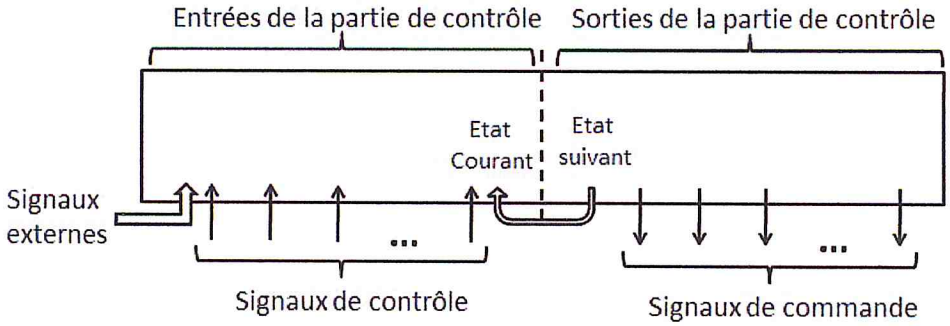



FIGURE 3 – *Partie de contrôle gérant les transferts de données.*

Bien souvent, l'utilisation d'une seule partie opérative et une seule partie de contrôle ne peut pas implémenter de manière efficace sur un SoC, nous somme alors amenés à implémenter un tel système à l'aide de plusieurs sous systèmes utilisant un bus global et une mémoire globale pour les faibles communications qu'ils s'échangent. Les importantes communications se font, quant à elles, localement, au niveau de chaque sous système. Ceci nécessite une décomposition fonctionnelle du système répondant aux contraintes de vitesse et de consommation de puissance [22].

## 1.4 Exigence des futures structures d'interconnexion

Pour faire face aux évolutions que nous venons de présenter, les architectures de communication devront répondre aux différentes exigences que nous allons détailler :

**Flexible et extensible :** L'architecture de communication doit être conçue pour interconnecter de nombreux blocs d'IP de nature hétérogène, de plus en plus complexes intégrant des fonctions ayant des caractéristiques très diverses. Le protocole d'échange des données doit donc être suffisamment flexible pour s'adapter à ces modules qui peuvent être développés par différentes équipes, avec différents outils, dans différents contextes et qui sont destinés à être intégrés dans le même circuit. Cette flexibilité peut être obtenue en spécifiant de manière rigoureuse des interfaces qui permettent de découpler les fonctions de traitement et les fonctions de communication [20]. On utilise le terme d'interface-based design [44]. Une architecture de communication flexible permet aussi de gagner en productivité.

D'un autre côté, La notion d'extensibilité est également un élément important. Il s'agit de la possibilité d'augmenter le nombre de blocs interconnectés tout en maintenant le même niveau de performance dans les communications.

- **Simplicité :** désigne la simplicité de mise en œuvre de cette structure, en minimisant le nombre de ressources nécessaires à sa réalisation.

## 1.5 Variété des systèmes d'interconnexions

Plusieurs standards de communication existent ou émergent avec leurs avantages et leurs inconvénients. Nous allons dresser dans cette section une liste des principaux modes d'interconnexions disponibles pour les SoC.

### 1.5.1 La connexion point à point

Il s'agit de la connexion la plus simple qui soit pour connecter deux IPs. Les blocs fonctionnels sont donc reliés entre eux directement sans aucun protocole de gestion de communications (Figure 4).

Chaque connexion est dédiée et spécifique ce qui permet d'avoir un parallélisme contrôlé et des connexions à haut débit [8]. Malheureusement, ce type d'interconnexions peut poser un problème d'extensibilité (scalability) pour de nombreux IPs présentant de nombreuses interconnexions entre-eux [42].


FIGURE 4 – *La connexion point à point [8].*

### 1.5.2 Le bus standard

La topologie bus (Figure 5)contrairement à la connexion point à point (Figure 4) permet de connecter toutes les IPs de l'application à un seul et même média de communication appelé bus [40]. Cette solution permet d'avoir une encapsulation des blocs fonctionnels identique et d'avoir un protocole de communication plus flexible, contrairement à la connexion point à point. Ceci permet de faire évoluer un SoC beaucoup plus rapidement car il suffit simplement d'étendre le bus et de rajouter d'autres IP pour intégrer de nouvelles normes ou applications à un SoC. La figure 5 montre un exemple de topologie bus standard.

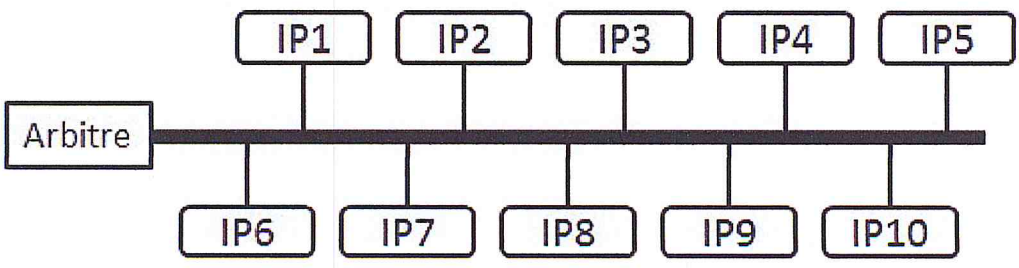



FIGURE 5 – *Topologie bus [8].*

L'inconvénient majeur d'une topologie bus standard est la contrainte de ne pouvoir réaliser qu'une seule communication à la fois (les taches peuvent se dérouler en parallèle mais les communications ne se font que de manière séquentielle). Ces communications étant toutes gérées par l'arbitre du bus, on crée un goulet d'étranglement lorsque le nombre de communications croît, mais également lorsque les contraintes de bande passante de plusieurs communications deviennent importantes. Cet arbitrage implique donc une limitation sur le nombre d'IP connectées au bus à une dizaine d'éléments [8].

De plus, ce mode d'interconnexion physique devient un facteur limitant en performance pour les SoC car les lignes du bus deviennent de plus en plus longues à mesure que le nombre d'IP connectées augmente. On voit ainsi apparaître des capacités parasites qui engendrent une augmentation du temps de charge (proportionnel au nombre d'éléments raccordés).

### 1.5.3 Les bus hiérarchiques

Pour solutionner les problèmes du bus standard énoncés précédemment, une solution permettant de contourner les limites de cette topologie est de créer un bus hiérarchique (Figure 6). Le concept de ces bus hiérarchiques est de faire communiquer plusieurs bus entre eux par l'intermédiaire d'un pont (Bridge) reliant deux bus entre eux [8].

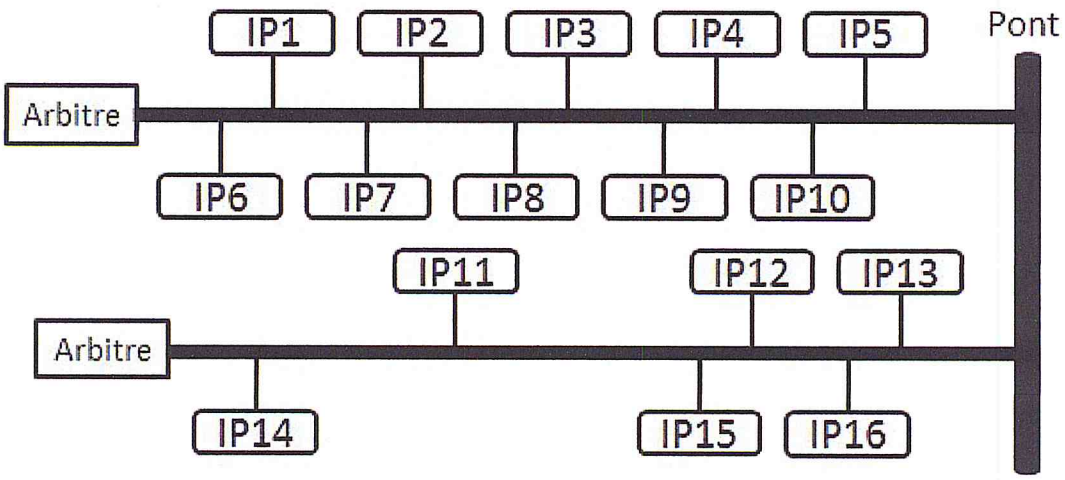



FIGURE 6 – *Topologie bus hiérarchique [8].*

L'intérêt de cette topologie hiérarchique est de permettre d'ordonnancer les différentes tâches qui composent le SoC en les répartissant sur les différents bus qui la composent de façon à ne pas surcharger un seul et même bus. Les différents segments du bus hiérarchique possèdent donc des longueurs courtes avec peu d'IP connectées, offrant une faible capacité

sur chacun de ces segments et permettant d'utiliser chaque bus à des fréquences élevées. Ainsi, les transactions peuvent se faire en parallèle sur les différents segments de bus et à fréquence élevée. De cette façon, nous offrons une qualité de service plus élevée à l'application et une meilleure bande passante globale.

Cependant, l'accès d'un bus à un autre au travers d'un pont, implique un coût en latence d'où l'importance de bien répartir les différentes IP communicantes pour limiter l'utilisation du pont et ainsi favoriser le fonctionnement parallèle des bus.

Enfin, le bus hiérarchique consomme en principe moins que le bus partagé puisque la capacité des éléments connectés au bus est plus faible sur chaque segment du bus. Ce découpage en segment est un premier pas qui tend vers l'approche réseau mais le goulot d'étranglement que l'on peut observer au niveau des ponts devient un facteur limitant compte tenu des besoins en bande passante des futures applications. C'est pourquoi les topologies réseaux ou "cross-bar" offrent une alternative intéressante pour les futurs SoC.

### 1.5.4 Le crossbar

Une alternative offrant un compromis intéressant entre les topologies bus et celles des réseaux est le crossbar [14]. Dans ce cas, tous les blocs fonctionnels de l'application sont reliés les uns aux autres par l'intermédiaire du crossbar (Figure 7). Celui-ci a l'avantage de permettre des communications parallèles (contrairement au bus) et d'offrir une grande bande passante pour chaque communication car celles-ci ne sont plus partagées avec les autres communications comme c'est le cas dans la topologie bus standard ou hiérarchique.

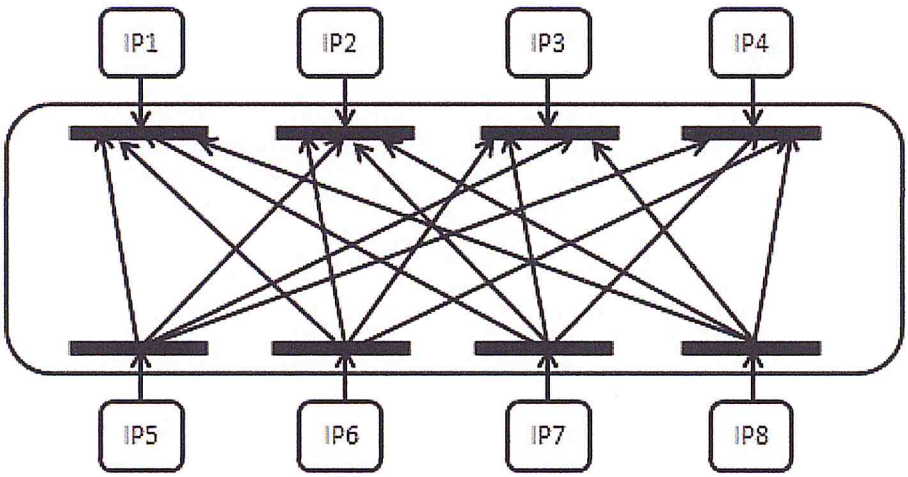



FIGURE 7 – *Topologie d'un crossbar 4 vers 4 [8].*

Cependant, la complexité de câblage d'une telle architecture croit en fonction du nombre d'IP qui composent l'application. Celle-ci augmente avec le carré du nombre d'éléments communicants soit une complexité de $O(n^2)$, ce qui devient vite exorbitant.

### 1.5.5 Le réseau

Depuis une dizaine d'années, nous avons vu une évolution rapide de la technologie d'interconnexion des systèmes sur puce, en particulier, depuis [50] qui a proposé de remplacer les bus par des réseaux sur puce avec des interconnexions à base de routeurs.

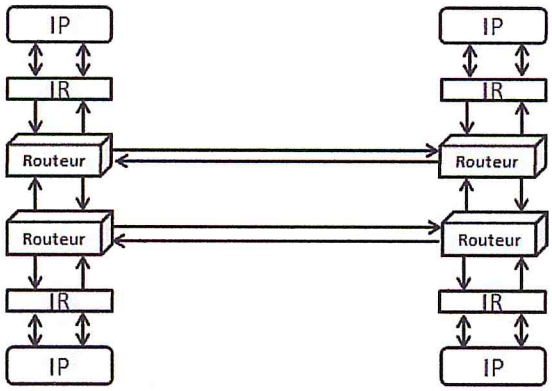



FIGURE 8 – *Topologie Réseau [8].*

## 1.6 Comparaison entre les types d'interconnexion

Etant donné l'évolution rapide et de plus en plus complexe des SoCs énoncée précédemment, l'interconnexion de communication des IPs constituant ces systèmes, a subi l'évolution topologique et structurelle suivante :

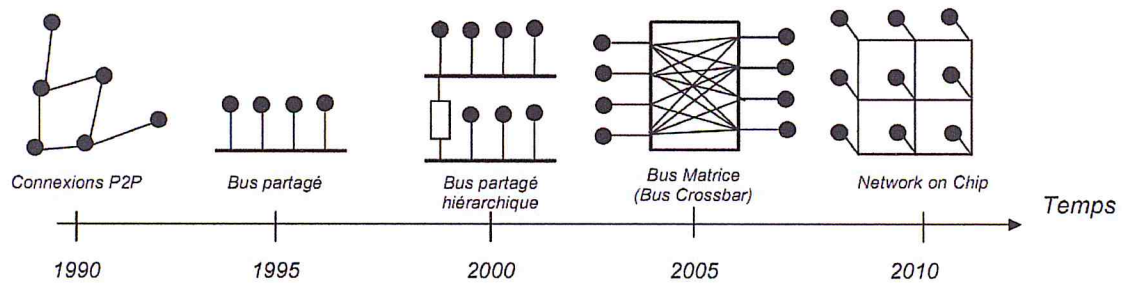



FIGURE 9 – *Evolution des interconnexions dans les SoC* [49].

Dans ce chapitre nous avons cité plusieurs types d'interconnexion dans les systèmes sur puce avec leurs avantages et leurs limites qui ont mené aux solutions actuelles, notamment aux NoCs.

Le tableau ci-dessous représente une comparaison entres ces différents structures d'interconnexions [42].

| Connexion | Parallélisme | Consommation | Scalabilité | Réutilisation |
|---|---|---|---|---|
| Point à point | ++ | ++ | – – | – – |
| Bus partagé | – – | – – | – – | ++ |
| Bus hiérarchique | + | – | – | ++ |
| NoC | ++ | ++ | ++ | ++ |

++ : Très bon.

+ : Bon.

– : Mauvais.

– – : Très mauvais.

TABLE 2 – *Tableau comparatif entre les différentes structures d'interconnexions dans les SoCs [42].*

## 1.7 Conclusion

Nous avons présenté dans ce premier chapitre les facteurs participant à l'évolution de la technologie des semi-conducteurs. Cette évolution a permis d'intégrer un système complet sur une seule et même puce, c'est ce que l'on appelle le système sur puce, aussi appelé SoC (System on Chip).

Nous avons aussi présenté les différents modes d'interconnexions, les plus utilisés de nos jours, qui sont basés fondamentalement sur les bus partagés, mais qui sont limités en termes de bande passante et d'extension à cause de l'augmentation du nombre de composants communicants. C'est pour cette raison que ce type de structure ne semble plus adapté aux applications futures et c'est dans ce contexte que les réseaux sur puce ont été introduits. Dans le chapitre suivant nous allons détailler la structure de ces derniers.

# Chapitre 2

# Réseaux sur puce (NoCs)

## 2.1 Introduction

Après avoir introduit les réseaux sur puce dans le contexte de la conception des SoCs, nous entamons ce deuxième chapitre dont l'objectif est de présenter un certain nombre de concepts relatifs aux NoCs. Ces différentes notions sont issues des publications qui ont été faite dans ce domaine. Nous présenterons dans ce chapitre certaines topologies de réseaux sur puce, dont celle proposée dans [21].

## 2.2 Définition d'un réseau sur puce

Un réseau sur puce NoC est une technique de conception du système de communication entre les IPs dans un SoC [7]. La tâche essentielle d'un NoC est d'échanger des informations d'un point vers un autre tout en améliorant les performances par rapport aux interconnexions de bus et cela au niveau de la bande passante et de l'extensibilité [8].

## 2.3 Composants d'un réseau sur puce de base

Un réseau sur puce est composé de routeurs, d'adaptateurs réseau (interfaces), d'IPs (Intellectual Property), et de lien comme le montre la figure suivante :

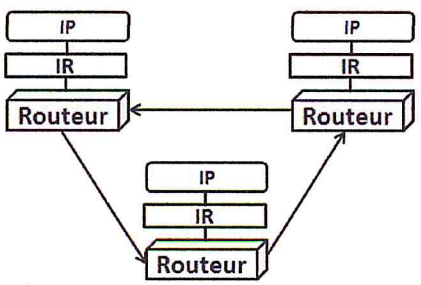



FIGURE 10 – *Structure d'un réseau sur puce[11].*

1. Les nœuds de routage sont chargés de diriger les données qu'ils reçoivent d'une source vers leurs destinataires.
2. Les interfaces réseau sont les éléments qui séparent le traitement (qui s'effectue au niveau des IPs) des communications (acheminés par le réseau)c'est à dire elles permettent aux IPs d'accéder au réseau sur puce.
3. les IPs sont des composants matériels ou logiciels utilisés selon le contexte. Dans notre cas un IP est un composant matériel "HW : Hard Ware" qui peut être soit un : ASIC, Mémoire, Module d'entrées/sorties ou autres.
4. Les liens de communication relient les routeurs entre eux ou les routeurs avec les interfaces réseau pour assurer le transfert des données entre deux composants du réseau sur puce.

## 2.4 Caractéristiques des réseaux sur puce

Un réseau sur puce pourra être caractérisé par plusieurs propriétés : débit, surface, énergie consommée, fiabilité, etc. Ces caractéristiques sont directement reliées aux choix de commutation, de routage et à la topologie [3].

### 2.4.1 Les techniques de commutation

Les techniques de commutation définissent comment sont utilisées les différentes ressources d'un réseau sur puce afin d'acheminer des données. La commutation dans les réseaux sur puce prend principalement deux formes : la commutation de circuits et la commutation par paquets.

#### La commutation de circuits

Dans ce mode de commutation, un chemin (circuit, lien unique) est établi entre l'émetteur et le récepteur à partir des liaisons du réseau commuté. Ce circuit est "temporaire" et n'appartient qu'aux deux entités qui communiquent.

Lorsque ce lien est réservé, l'émetteur envoie le message entier par ce même chemin. Ceci garantit une large bande passante entre les deux ressources et augmente les performances du système lorsque les données échangées sont de taille importante. Si, lors de l'établissement du lien, il y a conflit avec un chemin déjà réservé, l'envoi du message est retardé jusqu'à la libération du chemin en conflit [17].

Ce chemin est ensuite libéré de manière à ce que les liaisons puissent être utilisées dans le cadre d'une autre communication.

Le principal avantage de ce mécanisme de réservation est d'assurer que deux flots de communication ne rentreront jamais en conflit. Cependant, ce mode de commutation est très pénalisant en termes de ressources car celles-ci sont réquisitionnées tout au long du transfert. Un autre désavantage important de cette méthode est d'avoir une latence importante à cause du temps nécessaire à l'établissement de la connexion.

#### La commutation par paquets

La commutation par paquets, quant à elle, ne fait aucune réservation de liens. Ce mécanisme de commutation consiste à découper un message en plusieurs paquets avant d'être envoyé et tous les paquets fragmentés du message sont indépendants selon leur chemin. Dans les réseaux sur puce, un paquet est décomposé en plusieurs FLITs : « Flow control unIT ». Chaque flit est stocké dans une file d'attente puis transmis sur la voie appropriée. Pour que chaque flit puisse atteindre sa destination, des informations dites de routage leurs sont ajoutées. Ces informations de routage sont utilisées par les nœuds de routage pour aiguiller les flits qu'ils reçoivent vers leurs destinations. Cependant tous les flits d'un paquet doivent suivre le même chemin et les flits de différents paquets ne peuvent pas être entrelacés aux cours de leur parcours. A la réception, le paquet est reconstitué à partir des flits reçus [39].

Étant donné qu'il n'y a pas de réservation de chemin, si plusieurs paquets veulent obtenir le même lien, un seul pourra l'avoir et les autres devront attendre.

Ce mode de commutation permet un meilleur partage des éléments du réseau car les voies sont libérées dès qu'un flit est envoyé. Ceci réduit la latence et améliore les performances du système. En revanche, ce mode de commutation de paquets doit être employé avec précaution à cause des risques importants d'interblocage qui lui sont associés.

Les modes de commutations de paquets les plus utilisés dans les réseaux sur puce sont :

- Store and Forward (Stocker et propager) : avec cette stratégie, tous les flits constituant un paquet sont stockés avant d'être transmis. Le but de ce stockage est le contrôle des paquets envoyés [8].
- Wormhole (Trou de ver) : ce mode de commutation réduit la latence du système car il n'exige pas que tout le paquet soit stocké avant d'être envoyé. Dès qu'une voie est libre, un flit est transmis au routeur destinataire [20].

### 2.4.2 Les techniques de routage

Un algorithme de routage est utilisé par un nœud de routage pour déterminer le chemin (la suite des liens de communication à utiliser) que doit emprunter un paquet pour

atteindre sa destination en fonction des caractéristiques d'un réseau sur puce et de ses besoins (qualité de service, robustesse, . . .). Il existe cependant différents types d'algorithmes de routage qui peuvent être utilisés. Or, ces algorithmes sont classés en deux classes d'algorithmes de routage :

### Routage déterministe

Un algorithme de routage est dit déterministe [2] (ou statique) lorsqu'il ne dépend que de la localisation de l'expéditeur et du récepteur d'un paquet. C'est-à-dire la suite des liens de communication à utiliser est indiquée dans les informations de routage de chaque paquet. Le rôle d'un nœud de routage se limite donc à extraire et à appliquer les informations de routage contenues dans un paquet. Cette classe est caractérisée par sa grande simplicité d'implémentation.

### Routage adaptatif

A l'opposé, un algorithme de routage adaptatif (ou dynamique) prend en compte l'état du réseau (charge des liens de communication, liens de communication défectueux, . . .) pour prendre ses décisions [38].

Dans un algorithme adaptatif, la route empruntée par un message n'est pas connue à priori (un paquet ne contient que l'adresse de sa destination), mais est déterminée au fur et à mesure de sa propagation dans le réseau. Au niveau de chaque nœud traversé la fonction de routage (non déterministe ici) sélectionne un port de sortie en fonction d'un certain nombre de critères. Ces critères sont principalement liés à la charge du réseau. L'idée principale derrière ce concept consiste à éviter/contourner les régions du réseau déjà fortement chargées. Donc ce type d'algorithme offre une meilleure tolérance aux pannes, car il existe plusieurs chemins à emprunter pour un seul message.

En plus de définir le chemin que doit suivre un paquet pour atteindre sa destination, un algorithme de routage doit aussi assurer que l'ensemble des chemins suivis par les paquets ne crée pas des situations d'interblocage statique (ou deadlock). Ce phénomène survient lorsque plusieurs paquets se bloquent mutuellement à cause d'une dépendance cyclique.

Bien que le routage dynamique présente plus d'avantages que le routage déterministe, ce dernier est beaucoup plus utilisé pour les NoCs à cause de sa simplicité contrairement à l'autre qui complique la conception de ces systèmes et alourdit leur soft, ce qui est très déconseillé pour ce genre de systèmes.

#### Contrôle de flux

Le contrôle de flux est un ensemble de mécanismes qui évitent la surcharge du réseau et régulent le trafic. Afin de réaliser ces fonctions, des signaux de requêtes et d'acquittement sont utilisées par les routeurs. Il existe plusieurs techniques de contrôle de flux tel que le Handshake et le Credit-Based.

### 2.4.3 Les différentes topologies d'un NoC

La topologie spécifie l'organisation physique du réseau et définit la disposition structurelle de ses éléments. Il existe plusieurs topologies, les plus répandues étant :

#### 2D maillé

La maille à 2 dimensions [18] est une topologie matricielle et est la plus employée dans les réseaux sur puce. Ceci est dû à la facilité de son implémentation (Figure 11).

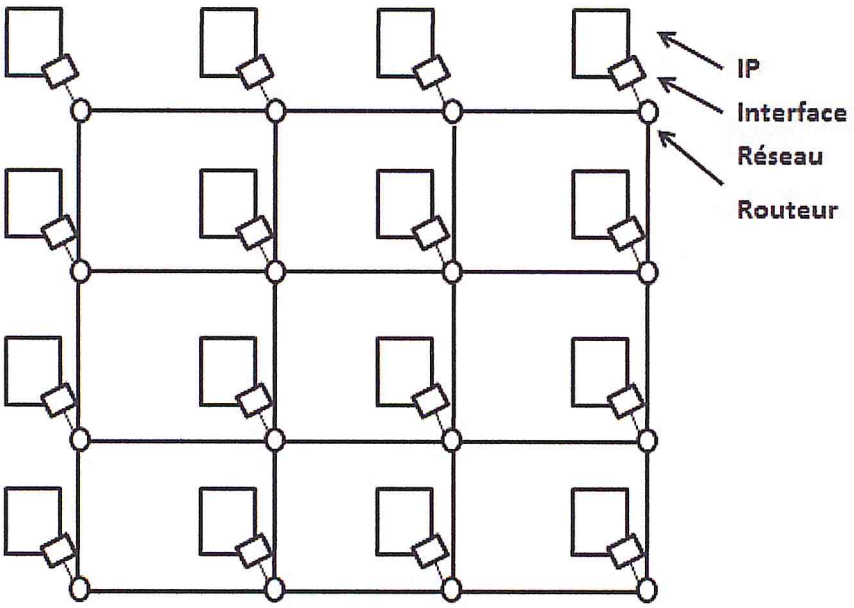



FIGURE 11 – *Topologies 2D maillée [18].*

#### Tore

C'est une topologie dérivée de la structure 2D possédant la particularité d'un repliement des bords extérieurs sur eux-mêmes [52]. Elle offre ainsi une bande passante légèrement supérieure à celle de la 2D maillée (Figure 12).

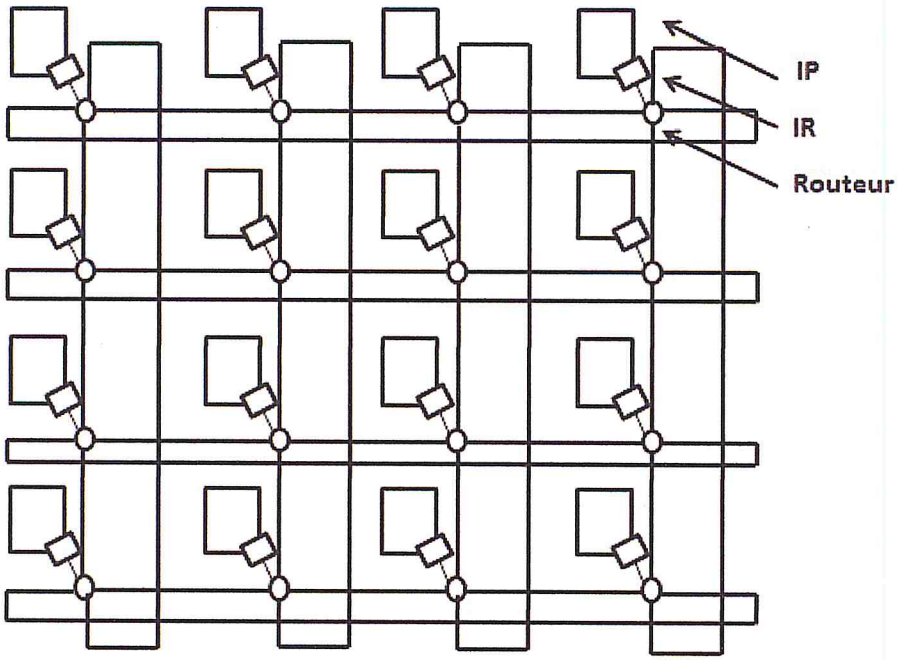



FIGURE 12 – *Topologies Tore [52].*

### Anneau

La topologie en anneau est une structure facilement intégrable sur silicium (Figure 13). Cependant, elle n'est pas extensible.

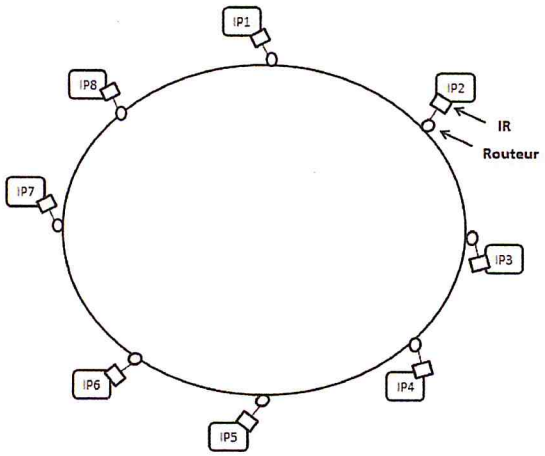



FIGURE 13 – *Topologies en anneau [8].*

### Fat tree

la structure (Figure 14) de cette topologie est sous forme d'arborescence, où chaque feuille représente un IP [13].

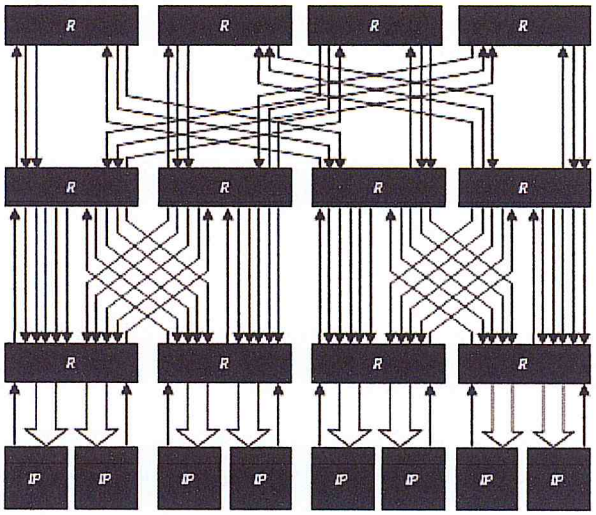



FIGURE 14 – *Fat tree [13].*

**Topologie proposée dans [21]**

Cette topologie consiste à exploiter les avantages des topologies qui offrent une très large bande passante, tout en assurant le critère d'extensibilité. Elle consiste aussi à réduire le temps de routage et les paramètres parasites présents notamment dans les Fat trees (Figure 14).

Puisque, le routage est déterminé lors de la conception (en exécution, le routage ne consomme pas beaucoup de temps). Ainsi, le budget de temps est presque consommé par l'application elle-même, et non pas par le routage, ce qui sera utile pour la conception des systèmes temps réel.

Pour plus d'avantages, les IPs qui communiquent fréquemment les uns avec les autres sont placés côte à côte et une insertion de répéteurs peut être employée pour les éventuelles longues interconnexions. L'extensibilité ne pose pas plus de problèmes qu'avec les autres méthodologies.

Dans la (Figure 15), il est indiqué que les transferts $T_{14}$ et $T_{42}$ par exemple, peuvent se faire sur la même interconnexion $S_1$ sans possibilité de conflit.

Dans ce contexte, nous optons dans le cadre de ce projet réaliser une tâche spécifique aux architectures 3D.

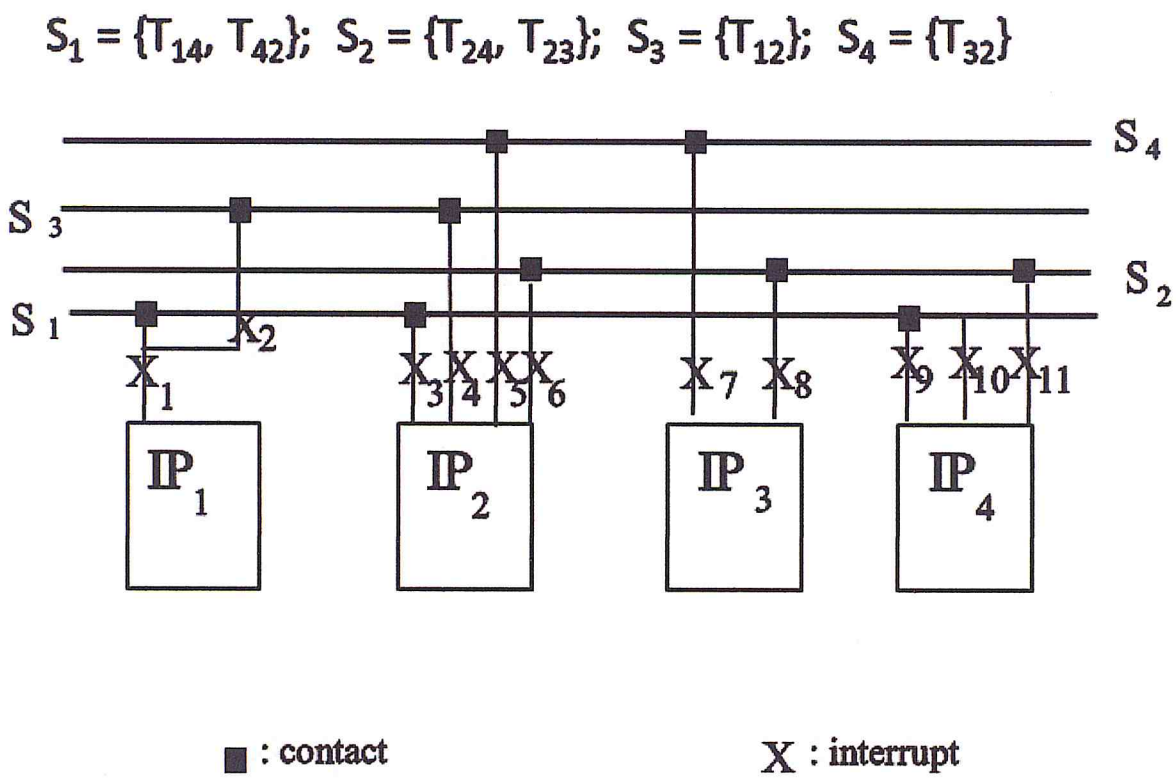



FIGURE 15 – *Topologie proposée dans [21].*

## 2.5 Conclusion

Dans ce chapitre, nous avons exposé dans un premier temps une brève introduction aux NoCs (qui sont venus pour remédier et résoudre les problèmes rencontrés dans les structures d'interconnexions des SoC) ainsi qu'une petite définition de ces derniers. Ensuite, nous avons cité un par un les différents composants d'un NoC, ainsi que ses caractéristiques.

Dans le prochain chapitre nous présenterons notre problème de placement des IPs dans le cadre des architectures 3D basées sur les réseaux sur puce.

# Chapitre 3

# Analyse du Problème de placement 3D

## 3.1 Introduction

Tout Circuit intégré doit être en mesure de répondre aux attentes des utilisateurs dont : une augmentation de la puissance de calcul, une réduction de la surface et une faible consommation d'énergie. Cela ne peut, évidemment, se faire sans une étude approfondie sur les enjeux qui influencent la fabrication des puces électroniques.

Ainsi, il existe deux acteurs principaux qui seront cités lors de la section suivante et qui sont : « les transistors et « les interconnexions ». S'intéresser à l'un de ces deux facteurs ne produit nullement des résultats satisfaisants.

Néanmoins, ce chapitre permet de présenter la situation actuelle et les tendances liées à la problématique d'intégration en microélectronique, de constater clairement que le délai généré par les interconnexions devient le facteur limitant des performances des circuits intégrés actuels, ainsi que de présenter une solution potentielle à ce problème en utilisant les architectures 3D et le placement des IPs.

## 3.2 La microélectronique 2D

Le motif de base de la microélectronique est le transistor, construit sur un matériau solide semi-conducteur, la plupart du temps des MOS, pour Métal Oxyde Semi-conducteur. Grâce à des progrès constants dans la finesse de gravure des circuits, la combinaison des différents transistors permet de réaliser des circuits intégrés plus ou moins denses, plus ou moins complexes. Concrètement, les puces électroniques qui sont fabriquées en série sur les wafers [1] peuvent être constituées de plusieurs centaines de millions de transistors

1. des plaques de silicium

(jusqu'à plus d'un milliard !), Or, ces progrès sont de plus en plus difficiles à maintenir en raison d'obstacles majeurs d'ordre économique et physique.

Gravées dans la couche de silicium, les interconnexions étant elles-mêmes fabriquées par dépôt de couches métalliques sur cette surface. Généralement, les transistors sont fabriqués sur une même surface plane, les uns à côtés des autres, en deux dimensions donc (technique dite Planar).

## 3.3 Limitations des circuits intégrés planaires

L'histoire de la microélectronique remonte à plus de 60 ans. Son évolution a été marquée, comme nous l'avons vu précédemment,par une course vers la complexité et l'hétérogénéité des systèmes. Cette croissance technologique et économique des plus agressives seulement ponctuée de quelques crises mineures provoque des augmentations continuelles de la fonctionnalité (en termes de performances et de diversité).

### 3.3.1 Un problème d'interconnexions

D'un point de vue volontairement simpliste, l'enjeu des interconnexions consiste à concevoir les fils reliant les différents éléments d'un circuit de la manière la plus efficace possible. Dans un cas strictement idéal, les interconnexions n'engendrent aucun délai de propagation du signal, ne prennent pas de place dans la puce, sont bon marché et toujours fiables. L'expérience acquise depuis 40 ans renvoie une toute autre réalité. L'enjeu perpétuel des interconnexions consiste à atteindre des valeurs de délai, de dimensionnement, de coût et de fiabilité qui constituent un compromis acceptable pour la réalisation d'interconnexions utilisables par l'industrie microélectronique [43].

### 3.3.2 Historique des interconnexions

Depuis la création de l'industrie microélectronique dans les années 50 jusqu'à la fin des années 90, les transistors ont dominé les performances et les coûts des puces, alors que les interconnexions n'ont joué qu'un rôle second dans ces domaines. Le XXIe siècle a vu cette hégémonie s'inverser significativement. Une bonne illustration de ce qu'est le problème des interconnexions consiste à faire le parallèle entre ce qui se faisait il y a une vingtaine d'années et ce qui se fait maintenant. Ainsi, pour une technologie 1 $\mu m$ (fin des années 80), le délai intrinsèque d'un MOSFET approchait les 10 ps, alors que le temps de réponse d'une interconnexion de longueur 1 mm avoisinait 1 ps. Une technologie de 0.1 Em typique du début des années 2000 présente des performances bien différentes. Le temps de commutation du MOSFET chute à 1 ps alors que le temps de réponse de la même interconnexion s'envole à 100 ps. On remarque ainsi que, sur une période d'environ

dix ans, l'augmentation du délai dans les interconnexions a été dix fois plus importante que la diminution du temps de commutation des transistors. A partir de cette simple constatation, il apparaît clairement que le délai généré par les interconnexions devient le facteur limitant des performances des circuits intégrés actuels.

Par une complexification croissante des circuits, la longueur totale des interconnexions a également augmenté d'un facteur 50 par puce. Cette augmentation des longueurs accroît la valeur de la capacité totale C inhérente au réseau d'interconnexions. Or elle constitue un paramètre qui joue directement sur la puissance dynamique consommée, exprimée par :

$$P = \frac{1}{2}\alpha.C.V_{dd}^2.f_c$$

où $V_{dd}$ est la tension d'alimentation, $f_c$ la fréquence d'horloge et $\alpha$ un facteur d'activité. Une grande partie de l'énergie dissipée dans une puce est ainsi due au réseau d'interconnexions [43].

Toutes ces évolutions pèsent sur les performances des interconnexions, qui imposent désormais des limitations sur la dissipation d'énergie, le délai et l'intégrité du signal dans un circuit intégré.

Les limites théoriques sont engendrées par les lois physiques et les innovations technologiques. Les limites pratiques couvrent les contraintes liées aux coûts de fabrication et aux tendances des marchés.

### 3.3.3 Des solutions temporaires pour dépasser ces limitations

Diminuer le délai dans l'interconnexion consiste, au premier ordre, à minimiser la valeur de son produit résistance\capacité (RC). Des solutions technologiques ont été mises en place dans ce sens. À titre d'exemple, l'aluminium, métal auparavant utilisé pour les interconnexions, a été remplacé par le cuivre, moins résistif.

En réalité, ces améliorations technologiques atteignent leur limite physique, rendant difficile la poursuite de la miniaturisation des circuits 2D classiques telle que prévue par la loi de Moore.

Toutes ces solutions technologiques et conceptuelles possèdent un potentiel intéressant pour ce qui est de réduire le délai propre aux interconnexions et la puissance dynamique associée. Néanmoins, nous venons de voir en quoi ces solutions sont aujourd'hui limitées dans les architectures classiques. Pour comprendre tout l'enjeu que représente le réseau d'interconnexions, il est nécessaire de décrire précisément sa hiérarchisation, et de

comprendre comment la répartition des différentes longueurs d'interconnexions peut être anticipée.

Toutes ces problématiques sont aujourd'hui de premier plan car les circuits atteignent un niveau de complexification extrême, et les intégrations monolithiques classiques risquent d'être trop lourdes à mettre en œuvre dans un avenir proche, aussi bien techniquement que financièrement.

### 3.3.4 Évolution et limitations des systèmes électroniques intégrés

Ces limitations ont pour conséquence de freiner l'augmentation de la densité d'intégration, rendant la politique industrielle du More Moore plus coûteuse et incertaine que jamais.

Ainsi, sans vraiment caricaturer la dynamique actuelle inhérente à la création d'un nouveau produit, la majorité des applications se doivent d'offrir plus de puissance de calcul, une réduction des dimensions, un faible coût et consommer moins d'énergie.

Leur conception doit aussi s'effectuer en un temps réduit (notion de délai de lancement importante) et dans un souci d'évolutivité. On a donc vu apparaître des systèmes complets intégrés sur une même puce, incluant soit des technologies similaires, soit des technologies hétérogènes. Ces systèmes sont connus sous les appellations anglophones System on Chip (SoC) (voir chapitre Les systèmes sur puce SoCs) et System in Package (SiP). Ils doivent être compris comme étant deux systèmes d'intégration complémentaires. Leur important développement leur fait jouer un rôle prépondérant dans les applications grand public actuelles. Néanmoins, les systèmes sur puce (SoC) se trouvent limités par leur grande complexité de conception, une des principales problématiques actuelles du domaine More Moore, et les systèmes en boîtier (SiP) doivent faire face à une mise en boîtier (packaging) de plus en plus coûteuse et complexe à réaliser, ce qui constitue la limite principale des systèmes hétérogènes du More than Moore.

#### Système en boîtier (SiP)

Le système en boîtier associe au sein d'un même boîtier les éléments les plus hétérogènes : circuits intégrés, MEMS, batteries, composants RF, modules de traitement biologique, etc. Les procédés de fabrication de ces différents composants sont trop hétérogènes pour être compatibles. , mais le coût de production unitaire est plus élevé et l'ajout a posteriori de la connectique pénalise la fiabilité et la vitesse des circuits.

Généralement, le terme SiP désigne l'ensemble des solutions permettant de réaliser une mise en boîtier de plusieurs puces indépendantes . Au-delà de cette signification, le véritable avantage du SiP réside dans le fait qu'il permet l'intégration de composants de nature réellement différente, ce que ne permet pas le système sur puce (où la conception est limitée à des systèmes de même nature technologique). Comparé au SoC, le système en boîtier se veut donc plus complet et représente un véritable système électronique intégré.

## 3.4 Exploitation de la troisième dimension dans les circuits intégrés

Nous venons de voir en quoi les architectures traditionnelles de circuits intégrés sont aujourd'hui limitées, à la fois d'un point de vue technologique pour la réalisation des futurs générations, que d'un point de vue conceptuel par la complexité croissante des circuits. Ces architectures de circuit vont devoir évoluer vers d'autres modèles. Les tous prochains défis technologiques vont pousser l'industrie microélectronique à faire des choix cruciaux pour mettre en œuvre de nouveaux types de circuits et peut être une autre façon d'envisager la conception et la production. Dans ce sens, l'intégration 3D de circuits est une des nouvelles architectures les plus prometteuses à l'heure actuelle.

A ce titre, en parallèle à toutes les solutions et innovations technologiques et conceptuelles proposées jusqu'à présent, l'intégration tridimensionnelle, dans sa forme la plus générale, représente une solution prometteuse aux problèmes de la miniaturisation et de la fonctionnalisation des circuits [43]. Ce concept présente un fort potentiel pour jouer un rôle déterminant dans la résolution des problèmes des interconnexions pour le noeud technologique 32 nm et au-delà. En ce sens, il s'inscrit à la lisière entre la dynamique de recherche « More than Moore », désignant les technologies émergentes en marge du CMOS (non digitales), et celle appelée « More Moore », qui s'attache à poursuivre la miniaturisation des dispositifs CMOS (Figure 16).

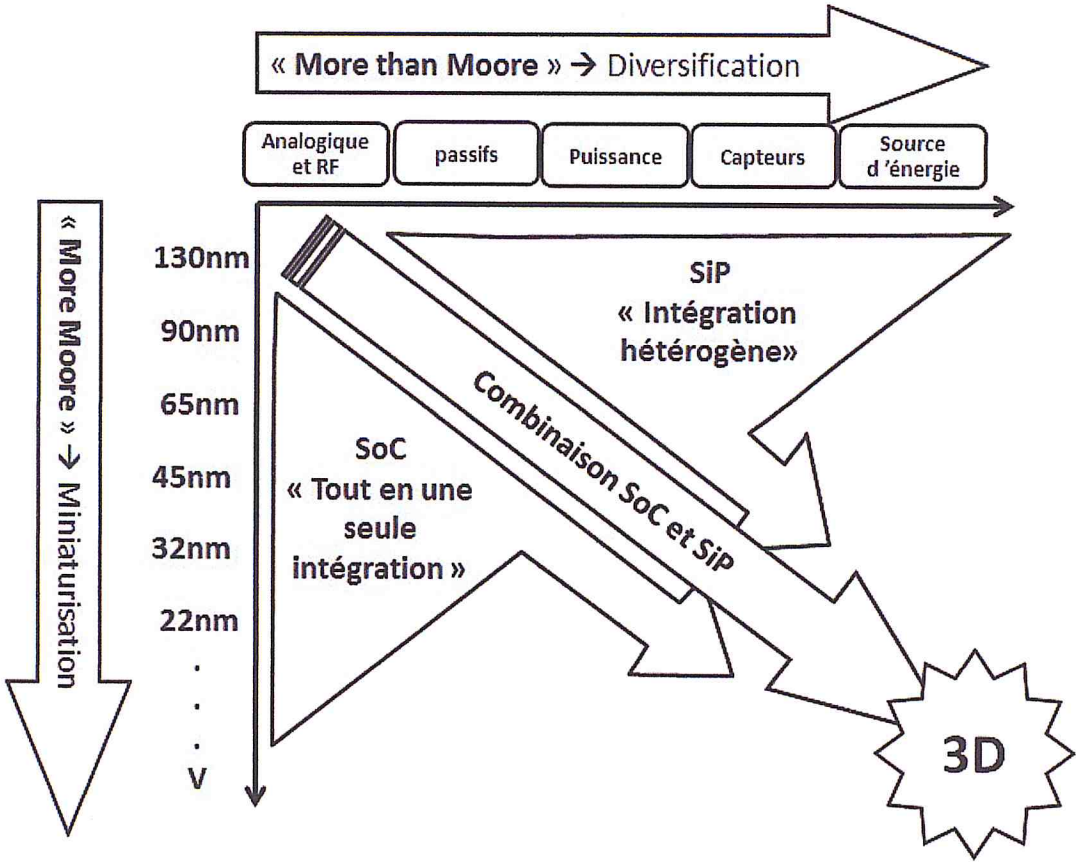



FIGURE 16 – *Positionnement de la thématique de l'intégration 3D dans le contexte présent et futur de l'industrie des semi-conducteurs [43].*

## 3.5 Les avantages d'intégration tridimensionnelle

Nous venons de voir que l'intégration classique (planaire) des blocs logiques d'une puce commence à montrer certaines limites, notamment en terme de temps de propagation du signal dans les interconnexions inter blocs.

Jusqu'à présent peu utilisée encore dans le domaine de la microélectronique, la troisième dimension apporte des avantages au niveau des performances du circuit pour tout ce qui concerne l'intégration homogène (domaine du More Moore), ainsi que de multiples possibilités au niveau de l'intégration hétérogène (domaine du More than Moore).

Le principal avantage est de pouvoir concevoir des systèmes hautement complexes par l'intégration de différents types de puces ou fonctionnalités dans un système compact afin de profiter des bénéfices de technologies hautement spécialisées.

Les bénéfices potentiels avancés par l'intégration 3D associe les concepts de multifonctionnalité, d'augmentation des performances des interconnexions, de diminution de la puissance consommée, le retard et les pertes d'intégrité des signaux grâce à des longueurs de connectique fortement réduites pour le transport des signaux et la distribution de puissance, faible facteur de forme, packaging réduit, augmentation du rendement et de la fiabilité, intégration hétérogène flexible et réduction des coût globaux.

## 3.6 Définition du placement

Il s'agit de trouver l'emplacement de chaque IP dans la puce électronique en fonction de son degré de communication avec les autres composants, et ce, en plaçant les IPs fortement communiquant côte à côte.

La fonction objectif à minimiser dans ce cas est la longueur des interconnexions, et les inconnus (encore appelés variables de décision) sont les emplacements des composants du circuit. Ce positionnement influencera sûrement les caractéristiques du circuit en termes de surface, de largeur de bande et de consommation de puissance.

L'allure de la fonction objectif de ce problème peut être schématiquement représentée comme sur la figure 17, en fonction de la « configuration » : chaque configuration est un placement particulier, associé à un choix de valeur pour chacune des variables de décision. Lorsque lespace des configurations possible présente une structure aussi tourmentée, il est difficile de repérer le minimum global $C^*$. Il est ce que nous appelons un problème non polynomial.

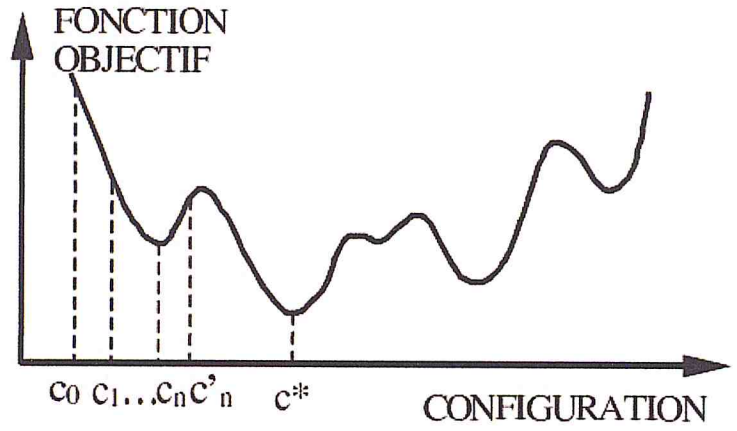



FIGURE 17 – *Allure de la fonction objectif d'un probème d'optimisation en fonction de la configuration.*

## 3.7 Conclusion

Dans ce chapitre, nous avons fait une petite synthèse quant aux tendances actuelles et futures des technologies des semi-conducteurs.

Enfin, n'oublions pas que le but premier de ce travail est de savoir donc comment placer les IP's de la manière la plus optimale que possible, sachant par ailleurs que les architectures 3D des SOC basées NOC croulent d'une part sous une masse considérable de composants et que d'autre part la communication entre ces composants influence grandement cette architecture.

A ce sujet, une réflexion émerge et une question trouve alors sa légitimité :
"Quelle méthode utiliser pour optimiser les interconnexions horizontales et verticales ?"

Pour trouver la réponse de cette question cruciale, nous devons faire recours au chapitre suivant.

# Chapitre 4

# Optimisation multicritères

## 4.1 Introduction

L'optimisation est une discipline en plein essor qui couvre une large gamme de techniques et fait toujours l'objet de recherches intensives. Elle touche plusieurs domaines telle que la conception de circuits électroniques, la recherche opérationnelle, la biologie, l'industrie, ... etc.

Pour certains problèmes dits d'optimisation difficile, utiliser une méthode d'optimisation exhaustive pour trouver la solution optimale s'avère limité -au regard des temps d'exécutions prohibitifs que nécessitent les problèmes de taille réelle-, voire inapte à fournir de telles solutions. L'optimisation de ces problèmes permet de trouver une configuration idéale ou pré-idéale.

Ce chapitre rassemble différents éléments concernant l'optimisation qui joue un rôle très important pour accomplir ce mémoire. Nous définissons tout d'abord ce que l'on entend par «Complexité théorique d'un problème», et nous fournissons par la suite une définition des problèmes d'optimisation. Les prochaines sections font ensuite un panorama des principales approches pour résoudre en pratique ces problèmes et citent quelques exemples de ces méthodes. Enfin, les deux dernières sections décrivent plus particulièrement l'optimisation multi objectif ainsi qu'une brève introduction au problème des graphes colorés.

## 4.2 Complexité théorique d'un problème

« Entre différents algorithmes réalisant une même tâche, quel est le plus performant ?». Pour répondre à cette question fondamentale, nous pourrons estimer le nombre d'instructions nécessaires pour que l'algorithme fournisse la bonne solution.

Les travaux théoriques dans ce domaine ont permis d'identifier différentes classes de problèmes en fonction de la complexité de leur résolution. Il existe en fait un très grand nombre de classes différentes, et nous nous limiterons ici à une présentation succincte nous permettant de caractériser formellement la notion de problème d'optimisation combinatoire.

La classe NP (Non-déterministe polynomial) inclut les problèmes qui peuvent être résolus, en temps polynomial, par une machine de Turing non déterministe[1]. Parmi ces problèmes, certains peuvent être résolu de manière déterministe[2] et ce, en un temps polynomial : ils appartiennent à la classe P ($P \subset NP$).

Nous qualifions de NP-complets les problèmes décisionnels. Pour montrer qu'un problème de décision $\prod$ est NP-complet, il suffit donc de montrer que $\prod$ appartient à NP, de choisir un problème $\prod'$ connu pour être NP-complet et d'établir la relation $\prod' \prec \prod$, où $\prec$ dénote la transformation polynomiale. Les problèmes NP-complets possèdent deux propriétés intéressantes. Tout d'abord, aucun d'entre eux n'a pu être résolu, à ce jour, par un algorithme déterministe polynomial. Ensuite, si l'on trouvait un algorithme déterministe polynomial permettant de résoudre un seul problème NP-complet, on pourrait en déduire un autre pour chacun des autres problèmes NP-complets, ce qui n'est pas encore le cas [1].

Nous qualifions de NP-difficiles les problèmes d'optimisation, c'est-à-dire ceux dont la réponse est de type numérique. À un problème d'optimisation NP-difficile on peut associer un problème de décision NP-complet dont le but est de déterminer s'il existe une solution pour laquelle la fonction objectif soit supérieure (respectivement inférieure) ou égale à une valeur donnée, mais dire qu'un problème NP-difficile est aussi NP-complet est un abus de langage. De nombreux problèmes d'optimisation combinatoire ont été prouvés NP-difficiles. Cette difficulté n'est pas seulement théorique, mais se confirme aussi dans la pratique.

## 4.3 C'est quoi un problème d'optimisation ?

Un problème d'optimisation se définit comme la recherche, parmi un ensemble de solutions possibles S (appelé aussi espace de recherche), de la (ou des) solution(s) $x^*$ qui rend

1. Un algorithme est dit non déterministe lorsque, appliqué sur une instance donnée du problème, donne des résultats qui peuvent varier d'une exécution à l'autre où dont le temps d'exécution varie.

2. Une machine de Turing déterministe a au plus une transition possible, contrairement à une machine de Turing non déterministe, qui peut en avoir plusieurs. Cela signifie qu'alors que les calculs d'une machine de Turing déterministe forment une suite, ceux d'une machine de Turing non déterministe forment un arbre, dans lequel chaque chemin correspond à une suite de calculs possibles.

(ent) minimale (ou maximale) une fonction mesurant la qualité de cette solution. Cette fonction est appelée fonction objectif ou fonction coût. Si l'on pose $f : S \rightarrow R$ la fonction objectif à minimiser (respectivement à maximiser) à valeurs dans R, le problème revient alors à trouver l'optimum $x^* \in S$ tel que f($x^*$) soit minimal (respectivement maximal).

Cependant, il peut exister des solutions intermédiaires, qui sont également des optimums, mais uniquement pour un sous-espace restreint de l'espace de recherche : nous parlons alors d'optimums locaux ( $x^*$ de S telle que $f(x^*) \leq f(x); \forall x \in V(x^*)$ ). Cette notion est illustrée dans la figure 18.

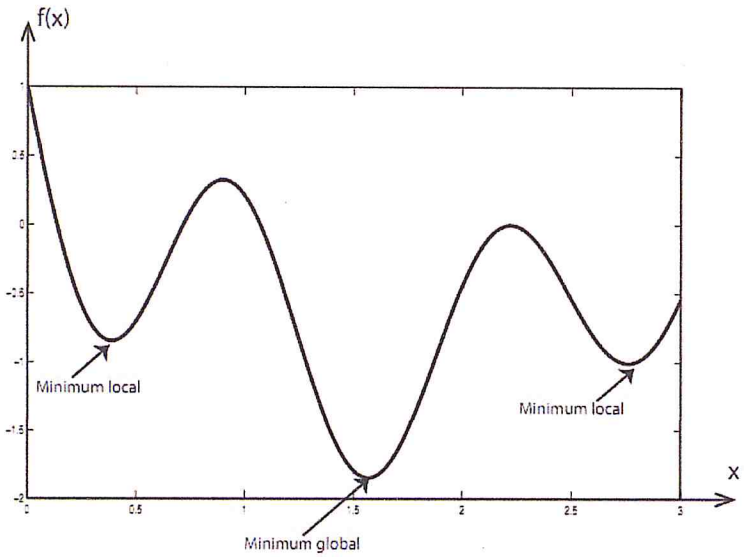



FIGURE 18 – *Différence entre un optimum global et des optima locaux.*

### 4.3.1 Optimisation sous contraintes

Les problèmes posés dans le monde réel sont souvent soumis à des contraintes. Le problème général d'optimisation sous contraintes peut être formulé comme suit :
Trouver x qui optimise f(x) sous les contraintes :

— $g_i(x) \leq 0, i = 1, 2, ..., p$

— $h_j(x) = 0, j = 1, 2, ..., q$

Le vecteur $x = (x_1, ..., x_n)$ est le vecteur des n variables de décision ($x \in F \subseteq S \subseteq R_n$), avec F l'espace des solutions réalisables du problème et S l'espace de recherche. Les contraintes sont des fonctions qui vont restreindre l'espace de recherche. Elles vérifieront si la solution est réalisable, mais elles ne mesureront pas la qualité de cette solution. Il en existe de deux types : contraintes d'inégalité $g_i(x) \leq 0, i = 1, 2, ..., p$ et contraintes d'égalité $h_j(x) = 0, j = 1, 2, ..., q$. Si une solution x ne satisfait pas au moins une des contraintes, elle est dite solution infaisable, contrairement aux solutions faisables, qui vérifient l'ensemble des $p + q$ contraintes.

### 4.3.2 Optimisation multicritère

La plupart des problèmes réels sont en fait des problèmes multi-objectifs, c'est-à-dire que nous cherchons à optimiser simultanément plusieurs critères qui sont généralement conflictuels. L'optimisation multiobjectif (dites aussi multicritère) consiste donc à optimiser simultanément plusieurs fonctions. Nous pouvons formaliser un problème d'optimisation multiobjectif comme suit :

- $minF(x) = (f_1(x), f_2(x), ..., f_m(x))$
- sous la contrainte $x \in C$

où $x = (x_1, ..., x_n)$ est le vecteur représentant les variables de décision ; C représente l'ensemble des solutions réalisables associé à des contraintes d'égalité, d'inégalité et des bornes explicites (espace de décision), et $F(x) = (f_1(x), f_2(x), ..., f_m(x))$ est le vecteur de fonctions objectifs à optimiser (ou critères de décision) avec $m \geq 2$ le nombre de fonctions objectifs. L'image de l'espace de décision par la fonction objectif F (ou ensemble des points réalisables $Y = F(C)$) est appelée espace des objectifs (ou espace des critères). On impose sur cet ensemble une relation d'ordre partiel appelée relation de dominance.

## 4.4 Les différents algorithmes d'optimisation

La classification des méthodes d'optimisation sera illustrée dans la figure 19. Nous distinguons en premier lieu l'optimisation continue de l'optimisation discrète (ou combinatoire), c'est la dichotomie « discret-contenu » qui conditionne évidemment beaucoup les possibilités de recourir à certaines méthodes.

Pour les problèmes d'optimisation combinatoire [3] qui font l'objet de notre étude, nous séparons sommairement les méthodes exactes (qui trouvent des solutions optimales) des méthodes approchées (qui donnent des solutions optimale ou pré-optimale) qui sont à leurs tours divisées en deux catégories les heuristiques spécifiques et les métaheuristiques génériques.

Pour les métaheuristiques, une subdivision parmi d'autres, distingue les méthodes à solution uniques de celles à population de solutions. Or, cette dernière approche utilise des techniques basées soit sur des algorithmes évolutionnaires ou sur l'intelligence en essaim.

3. L'optimisation combinatoire vise à déterminer parmi un ensemble fini de solutions potentielles une solution particulière qui satisfait un ou plusieurs critère(s) de minimisation ou maximisation.

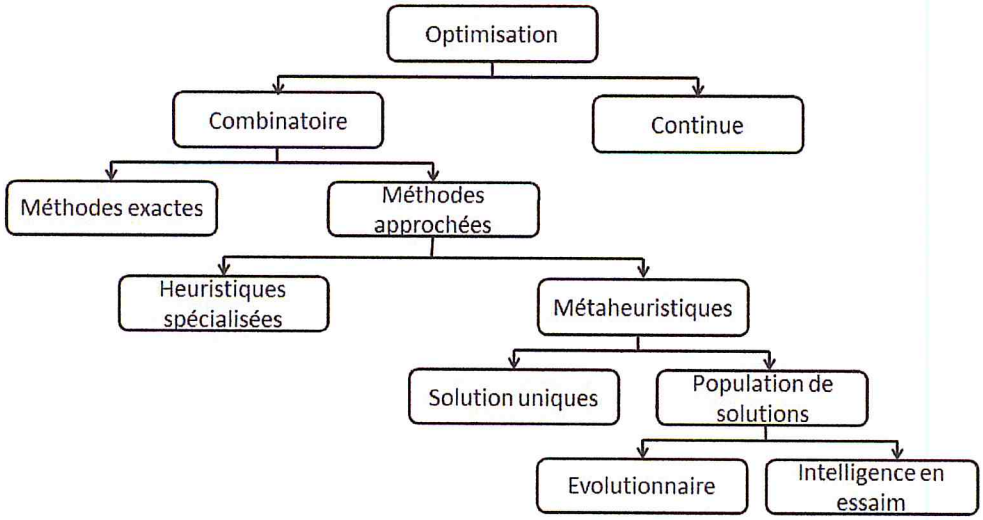



FIGURE 19 – *Classification des méthodes d'optimisation [9].*

### 4.4.1 Les méthodes exactes

Les méthodes exactes garantissent de trouver l'ensemble des solutions optimales. Pour ce faire, ces méthodes (déterministes) explorent systématiquement l'espace de toutes les combinaisons possibles et retournent l'optimum théorique.

Parmi les méthodes exactes, on peut citer la programmation dynamique et la méthode Little.

Cependant, leur approche par énumération accroit leur temps d'exécution de manière exponentielle en fonction de la taille du problème, ce qui fait qu'elles ne sont appropriées que lorsque la taille du problème est raisonnable, c'est à dire quand |S| n'est pas trop grand tel que S représente l'ensemble des solutions. Ce qui nous pousse à s'intéresser qu'aux méthodes approchées.

### 4.4.2 Les heuristiques

Issu du grec ancien eurisko qui signifie « je trouve », l'heuristique désigne l'art de la découverte. En informatique, une heuristique est une méthode de calcul approchée qui vise à atteindre une solution réalisable en temps polynomial, mais sans garantie d'optimalité, pour un problème d'optimisation NP-difficile.

Les heuristiques évitent le problème de l'explosion exponentielle en n'explorant qu'une partie de l'espace des combinaisons. Une heuristique n'est pas fondée sur un modèle formel et elle est entièrement dédiée à un problème dans un domaine donné, autrement dit, conçue

pour un problème particulier, en s'appuyant sur sa structure propre pour conduire souvent à la solution, d'où l'existence d'un très grand nombre d'heuristiques.

Qu'est-ce qu'une bonne heuristique ? Une bonne heuristique est de complexité raisonnable (idéalement polynomiale), robuste : fournit le plus souvent une solution proche de l'optimum global et a une faible probabilité d'obtenir une solution de mauvaise qualité.

### 4.4.3 Les métaheuristiques

On parle de méta, du grec $\mu\varepsilon\tau\alpha$ « au-delà », dans notre contexte à un plus haut niveau qu'une heuristique, quand nous visons à résoudre des problèmes d'optimisation difficile, en utilisant des heuristiques génériques qui se caractérisent cependant par un haut niveau d'abstraction, leur permettant d'être adaptées (sans nécessiter de changements profonds dans l'algorithme) à une large gamme de problèmes différents.

Elles s'inspirent généralement d'analogies avec la physique (recuit simulé), avec la biologie (algorithmes évolutionnaires), avec la biogéographie (BBO : Biogeography-Based Optimization) ou encore l'éthologie (colonies de fourmis, essaims particulaires). Leurs domaines d'application sont vastes et s'étendent souvent bien au-delà des problèmes pour lesquels elles ont été initialement conçues.

Par opposition aux méthodes exactes, les métaheuristiques sont exploitées quand la taille de l'espace de solution |S| est trop grande et permettant d'obtenir une valeur approchée de la solution optimale en un temps polynomial. Elles sont globalement des algorithmes stochastiques itératifs. Les itérations successives doivent normalement permettre de passer d'une solution de mauvaise qualité à la solution optimale. L'algorithme s'arrête après avoir atteint un critère d'arrêt, consistant généralement en une précision demandée. Si une nouvelle solution est construite à partir d'une solution existante, elle est sa voisine. Le choix du voisinage et de la structure de donnée le représentant peut être crucial.

En effet, ces algorithmes sont en général non-déterministes[4], ils peuvent ne pas trouver la solution optimale, et encore moins prouver l'optimalité de la solution trouvée. Nous en verrons quelques-unes par la suite.

4. Un algorithme est dit non déterministe lorsque, appliqué sur une instance donnée du problème, donne des résultats qui peuvent varier d'une exécution à l'autre où dont le temps d'exécution varie, contrairement aux algorithmes déterministes, qui se comportent toujours de la même façon et produisent toujours le même résultat (heuristique gloutonne).

## 4.5 Tentatives de classification des méta-heuristiques

Nous allons présenter les différentes classifications possibles des métaheuristiques selon le ou les critères de regroupement choisis.

- Une classification selon le type : naturel (algorithme génétique, colonies de fourmis ou d'abeilles ... etc.) ou artificiel (recuit simulé, méthode tabou... etc.).
- Une classification selon le mode de traitement de la fonction objectif Z(S) : Z(S) statique qui demeure inchangée d'un bout à l'autre de l'algorithme (recherche tabou) ou bien dynamique, quand la fonction objectif est modifiée en fonction des informations collectées au cours de l'exploration de l'espace de recherche (recherche locale guidée), l'objectif de cette modification est d'augmenter les chances de s'échapper de l'optimum local afin d'atteindre l'optimum global.
- Une autre manière de classer les métaheuristiques consiste à regrouper les métaheuristiques comme celles qui ont une mémoire (la plus connue étant la méthode tabou) et celles qui n'ont pas de mémoire (avancent en aveugle, autrement dit quand l'algorithme s'échappe de l'optimum local il peut revenir à ce dernier car il ne se souvient pas de son dernier mouvement). Un bon exemple en est le recuit simulé que nous pouvons améliorer en lui donnant une mémoire.
- Un autre système de classification distingue les métaheuristiques en fonction de la façon dont elles évoluent : celles qui manipulent en parallèle toute une population de solutions (on peut citer les algorithmes évolutionnaires et l'intelligence en essaim.) et les autres qui se basent sur l'évolution itérative d'une solution unique (la méthode Tabou et le Recuit Simulé sont des exemples typiques de ces méthodes). Cette classification est la plus réputée dans la littérature et elle sera traitée plus en détails dans la section suivante.

## 4.6 Quelques métaheuristiques connues

De très nombreuses métaheuristiques existent dans la littérature. Nous allons détailler celles qui sont illustrées dans la figure suivante.

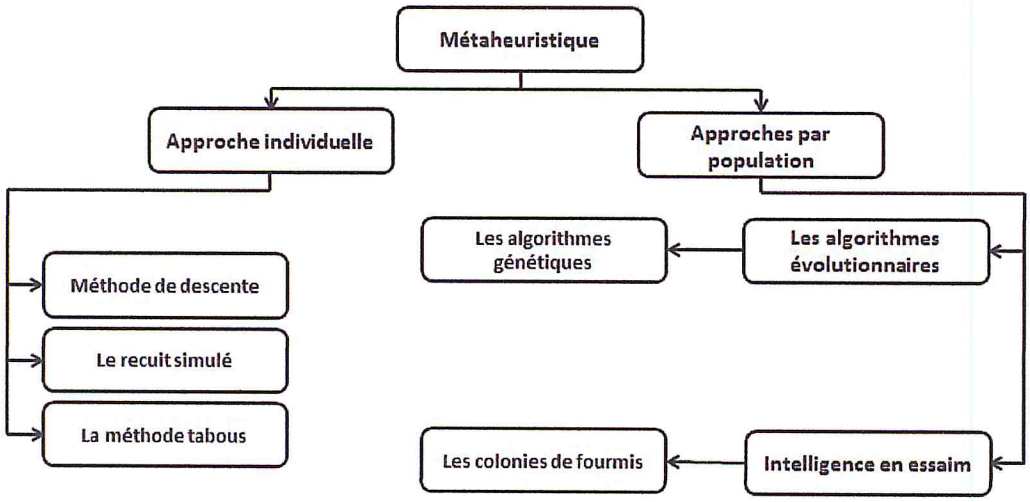



FIGURE 20 – *Classification des métaheuristiques.*

### 4.6.1 Approche individuelle (trajectoire)

A partir d'une seule et unique solution initiale $S_0$ de l'espace des solutions admissibles (S) du problème d'optimisation combinatoire, l'algorithme construit de proche en proche une trajectoire qui converge, on l'espère, vers ce que nous pouvons appeler l'optimum globale du problème. Nous disons trajectoire parce que l'algorithme va construire progressivement un itinéraire de telle façon qu'à chaque étape la solution trouvée est meilleure que la solution précédente ainsi de suite jusqu'à ce que nous trouvons une solution qui sera, après un certain nombre d'itérations jugé suffisant et selon la métaheuristique utilisée, la solution optimale ou la plus proche de la solution optimale.

Les méthodes de trajectoire englobent essentiellement la méthode de descente, la méthode du recuit simulé, la recherche tabou, la méthode GRASP, la recherche à voisinage variable, et leurs variantes.

**i. Méthode de descente**

- **Principe** Le principe est simple : à partir d'une solution existante, chercher aléatoirement une solution dans le voisinage et accepter cette solution si elle améliore la solution courante. Cette recherche s'arrête quand une solution localement optimale est trouvée, c'est à dire quand il n'existe pas de meilleure solution dans le voisinage. La plus part du temps, les opérateurs de recherche locale restent enfermés dans un optimum local. Il existe cependant plusieurs variantes de cette méthode : descente stochastique, descente profonde, descente kangourou, ..., qui tentent à échapper de cet optimum local.

- **Organigramme :**

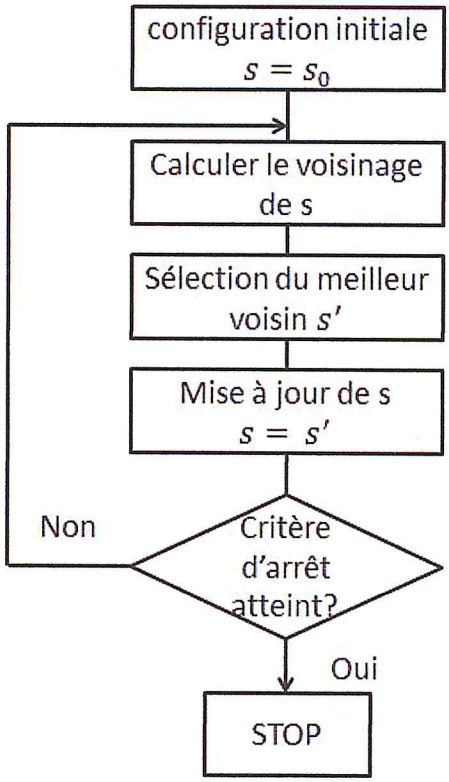



FIGURE 21 – *Organigramme de la méthode de descente [15].*

- **Algorithme :**

**Algorithm 1** Descente simple ()

Choisir une solution de départ $s_0$ ;
Initialiser s à $s_0$ $s = s_0$ ;
**repeat**
  Calculer le voisinage de s noté V(s) ;
  Pour tout $s'$ de V(s)
  **if** $(Z(s') <= Z(s))$ **then**
    $s = s'$ ;
    Fin ;
  **end if**
**until** (critère d'arrêt n'est pas atteint)
$s^* = s$ ;
Retourner la meilleure solution trouvée $s^*$ ;

### ii. Le recuit simulé

- **Principe :** Le recuit simulé a été Introduit par Kirkpatrick en 1982. Il tire son origine de la mécanique statistique en utilisant les critères de la méthode de Métropolis 1953. Son principe de fonctionnement repose sur une imitation du phénomène de recuit en science des matériaux basé sur les principes d'équilibre énergétique lors de la cristallisation des métaux. L'analogie avec une méthode d'optimisation est

trouvée en associant une solution à un état du métal, l'équilibre thermodynamique est la valeur de la fonction objectif de cette solution. Passer d'un état du métal à un autre correspond à passer d'une solution à une solution voisine. Le recuit simulé est une technique stochastique de type Monte-Carlo généralisé pour la résolution approchée de problèmes NP-difficile de l'optimisation combinatoire basé sur un paramètre appelé température, qui sera ajusté au cours de la recherche. A partir d'une solution initiale il recherche dans son voisinage une autre solution de façon aléatoire qui peut être de moins bonne qualité permettant d'échapper aux optima locaux en acceptant temporairement une dégradation de la fonction objectif. Initialement, la température est élevée autorisant une forte dégradation de qualité puis décroît pour diminuer la probabilité des dégradations importante.

- **Organigramme :**

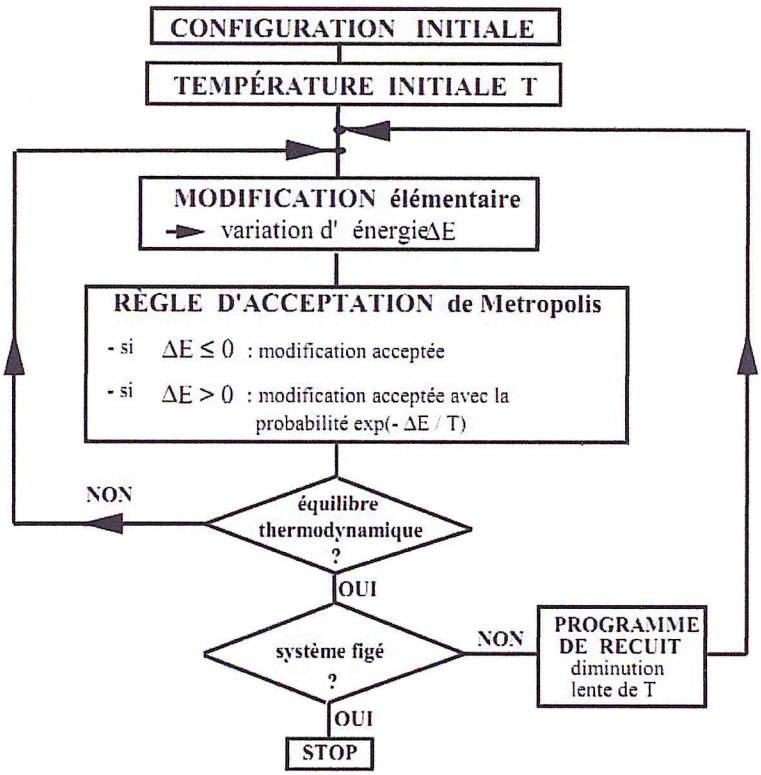



FIGURE 22 – *Organigramme de l'algorithme recuit simulé [45, 5].*

- **Algorithme :**

**Algorithm 2** Recuit Simulé ()

Choisir une solution de départ $s_0$ ;
Initialiser s à $s_0$ $s = s_0$ ;
Initialiser T en fonction de schéma de refroidissement ;
**repeat**
  Engendrer un voisin aléatoire $s^{'}$ de s ;
  **if** (CritMetropolis (D,T)) **then**
    $s = s^{'}$ ;
    Mettre T à jour en fonction du schéma de refroidissement ;
  **end if**
**until** (critère d'arrêt n'est pas atteint)
$s^* = s$ ;
Retourner la meilleure solution trouvée $s^*$ ;

### iii. La méthode de recherche avec tabous

- **Principe :** Le principe de base est de poursuivre la recherche de solutions même lorsqu'un optimum local est rencontré en permettant des déplacements qui n'améliorent pas la solution et en utilisant le principe de mémoire pour éviter les retours en arrière (mouvements cycliques). Contrairement au recuit simulé qui ne génère qu'une seule solution $s^{'}$ aléatoirement dans le voisinage V(s) de la solution courante s, la méthode tabou, dans sa forme la plus simple, examine le voisinage V(s) de la solution courante s. La nouvelle solution $s^{'}$ est la meilleure solution de ce voisinage (dont l'évaluation est parfois moins bonne que s elle-même). Pour éviter de cycler, une liste taboue (qui a donné le nom à l'algorithme) est tenue à jour et interdit de revenir à des solutions déjà explorées.
- **Organigramme :**

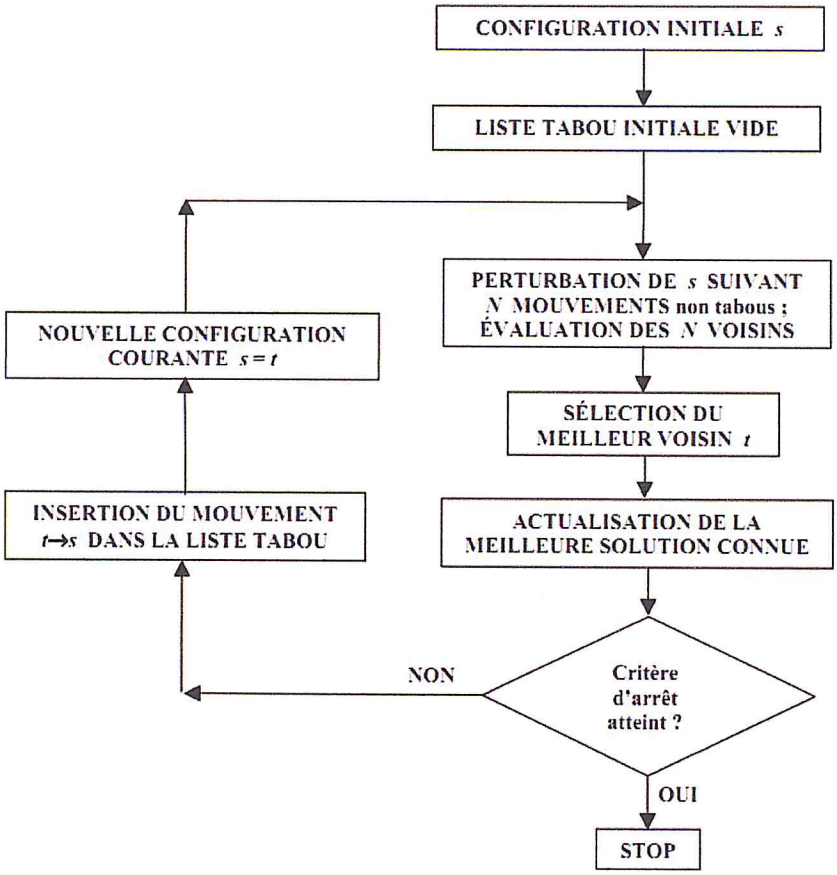



FIGURE 23 – *Organigramme de l'algorithme tabou simple [12].*

- **Algorithme :**

**Algorithm 3** Tabou Search()

Choisir une solution de départ $s_0$ ;
Initialiser s à $s_0$ $s = s_0$ ;
Créer une liste Tabou vide ;
**repeat**
  Perturbation de s suivant N mouvements non tabou ;
  Évaluation des N voisins ;
  Sélection du meilleur voisin t ;
  Actualisation de la meilleure solution connue ;
  Insertion du mouvement t $\rightarrow$ s dans la liste tabou ;
  Nouvelle configuration t (si t est meilleure que s) s=t ;
  s=t ;
**until** (critère d'arrêt n'est pas atteint)
$s^* = s$ ;
Retourner la meilleure solution trouvée $s^*$ ;

### 4.6.2 Approches par population

Ces méthodes partent d'une population (plusieurs solutions initiales) plutôt que d'individu, donc au lieu d'avoir une solution initiale $S_0$ nous aurons $S_0^1, S_0^2, S_0^3, \ldots, S_0^n$ avec lesquels nous travaillions dès le départ. A partir de cette population l'algorithme construit, au fur et à mesure des itérations, des générations en espérant qu'à long terme la génération finale soit relativement homogène et donc contient des solutions dont les coûts seront pratiquement identiques ou très proche les uns des autres. Ces méthodes sont très gourmandes en calcul.

On distingue dans cette catégorie, les algorithmes évolutionnaires, qui sont une famille d'algorithmes issus de la théorie de l'évolution par la sélection naturelle, énoncée par Charles Darwin [6] et les algorithmes d'intelligence en essaim qui, de la même manière que les algorithmes évolutionnaires, proviennent d'analogies avec des phénomènes biologiques naturels.

#### 1. Les algorithmes évolutionnaires

Les phénomènes biologiques ont été à la source de nombreux algorithmes dits évolutionnistes ou algorithmes évolutionnaires s'en inspirant plus ou moins librement. Ainsi les algorithmes génétiques s'inspirent de l'évolution darwinienne pour résoudre des problèmes divers.

Mais les algorithmes évolutionnaires sont avant tout des méthodes stochastiques d'optimisation globale. Et la souplesse d'utilisation de ces algorithmes pour des fonctions objectifs non régulières, permet leur utilisation pour des problèmes qui sont pour le moment hors d'atteinte des méthodes déterministes plus classiques.

La conception des algorithmes évolutionnaires AEs est basée sur l'idée que l'apparition d'espèces adaptées au milieu est la conséquence de la conjonction de deux phénomènes : d'une part la sélection naturelle imposée par le milieu –les individus les plus adaptés survivent et se reproduisent – et d'autre part des variations non dirigées du matériel génétique des espèces (mutations).

Les AEs englobent une classe assez large de métaheuristiques telles que les algorithmes génétiques et les algorithmes à estimation de distribution. La figure 24 décrit le squelette d'un algorithme évolutionnaire type, commun à la plupart des instances classiques d'AEs.

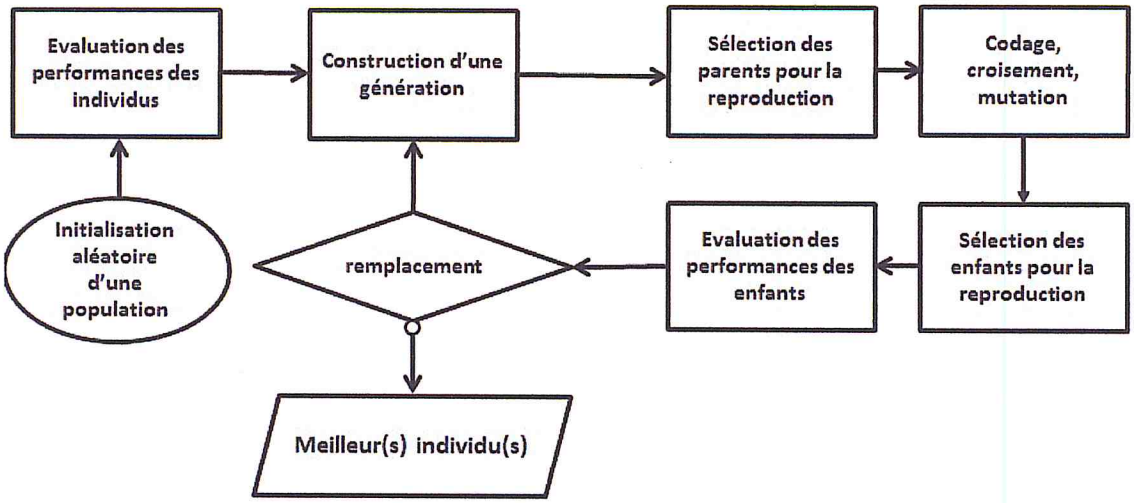



FIGURE 24 – *Squelette d'un algorithme évolutionnaire.*

Chaque génération est sujette à des variations génétiques et à la pression de la fonction d'adaptation, ce qui provoque la sélection naturelle. Les opérateurs de variation sont appliqués aux individus parents sélectionnés, ce qui génère de nouveaux descendants appelés enfants. Nous parlerons de mutation [5] pour les opérateurs unaires, et de croisement pour les opérateurs binaires (ou n-aires). Les individus issus de ces opérations sont alors insérés dans la population. Le processus d'évolution est itéré, de génération en génération, jusqu'à ce qu'une condition d'arrêt soit vérifiée.

### 1.i. Les algorithmes génétiques

- **Principe :** Les variables définissent chacune un gène du chromosome. Ces chromosomes évoluent grâce à différentes opérations calquées sur les lois de la génétique. Pour déterminer quels individus sont plus enclins à obtenir les meilleurs résultats, une sélection est opérée. Dans la première étape, les individus les plus adaptés gagnent la compétition de reproduction tandis que les moins adaptés meurent avant la reproduction, ce qui améliore globalement l'adaptation [4]. La seconde étape est de générer une nouvelle population à partir des individus sélectionnés en utilisant l'opérateur de croisement. Il s'agit tout simplement d'un échange de fragments de chromosomes. La reproduction alliée au croisement, donnent aux algorithmes génétiques une grande part de leur puissance, mais l'opérateur de mutation joue cependant un rôle dans le fonctionnement des algorithmes génétiques : il permet d'éviter les pertes qui auraient pu être irréparables et permet d'introduire de nouveaux gênes éventuellement non encore explorés. La mutation permet d'éviter de converger vers des extremums locaux en agrandissant l'espace de recherche. La fréquence des mutations doit néanmoins rester assez basse [23].

5. L'idée directrice de la mutation est de permettre de corriger une solution non réalisable et d'étudier des éléments représentatifs.

- **Organigramme :**

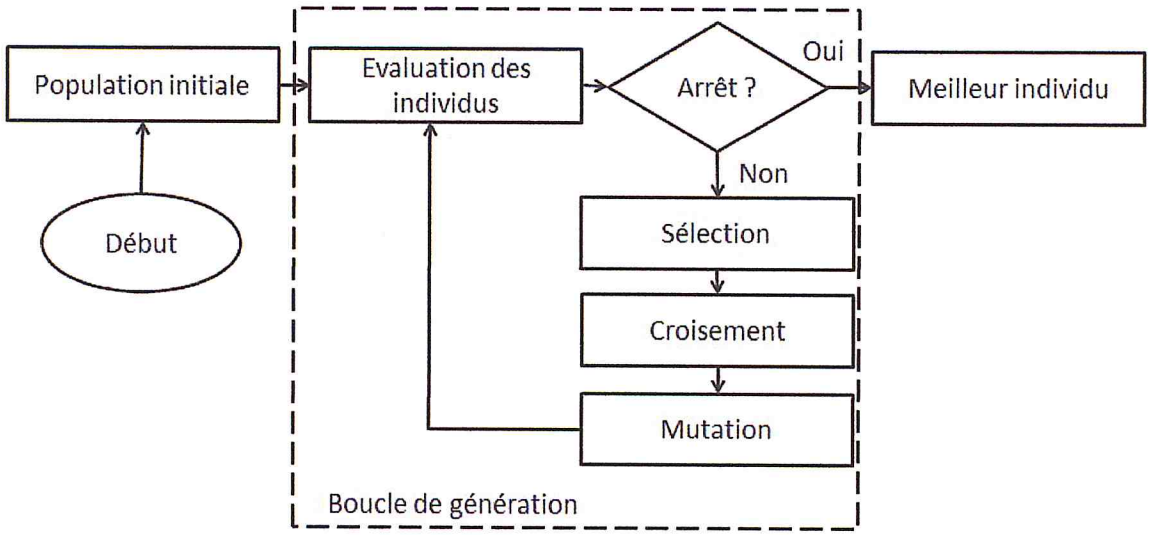



FIGURE 25 – *Organigramme de l'algorithme génétique simple [4].*

- **Algorithme :**

**Algorithm 4** Agorithme génétique()

Génération d'une population de base aléatoirement $G_0$ ;
Poser $G \longleftarrow G_0$ ;
Évaluation ; //valeur correspond à l'adaptation de chaque individu au problème
Poser $X^*(meilleuresolutiondeG_0)$ ;
**repeat**
  Sélection ; // Chaque individu a une probabilité d'être tirée proportionnelle à son adaptation au problème.
  Croisement et mutation ; // Chaque couple donne 2 individus fils.
  Mettre à jour $X^*$si nécessaire ;
**until** (critère d'arrêt n'est pas atteint)

### 2. Intelligence en essaim

L'intelligence en essaim (Swarm Intelligence) est née à partir des phénomènes biologiques. Elle recouvre un ensemble d'algorithmes, à base de population d'agents simples, qui interagissent localement les uns avec les autres. Ces entités, dont la capacité individuelle est très limitée, peuvent conjointement effectuer de nombreuses tâches complexes nécessaires à leur survie. Bien qu'il n'y ait pas de structure de contrôle centralisée qui dicte la façon dont les agents individuels devraient se comporter, les interactions locales entre les agents conduisent souvent à l'émergence d'un comportement collectif global et auto-organisée.

Un exemple phare d'algorithmes de l'intelligence en essaim est les algorithmes de colonies de fourmis. D'autres algorithmes d'optimisation qui proviennent d'analogies avec des phénomènes biologiques naturels ont été proposés. Parmi les plus significatifs d'entre eux figure l'algorithme d'optimisation basée sur la biogéographie (Biogeography-Based Optimization) et les algorithmes d'optimisation par essaim particulaire.

**2.i. Les colonies de fourmis**

- **Principe :** Les algorithmes à base de colonies de fourmis ont été introduits par Dorigo. Une des applications principales de la méthode originale était le problème du voyageur de commerce et depuis elle a considérablement évolué. Cette nouvelle métaheuristique imite le comportement de fourmis cherchant de la nourriture. A chaque fois qu'une fourmi se déplace, elle laisse sur la trace de son passage une odeur (la phéromone). Avec plusieurs de ses congénères, elle explore une région en quête de nourriture. Face à un obstacle, le groupe des fourmis explore les deux côtés de l'obstacle et se retrouvent, puis elles reviennent au nid avec de la nourriture. Les autres fourmis qui veulent obtenir de la nourriture elles aussi vont emprunter le même chemin. Si celui-ci se sépare face à l'obstacle, les fourmis vont alors emprunter préférablement le chemin sur lequel la phéromone sera la plus forte. Mais la phéromone étant une odeur elle s'évapore. Si peu de fourmis empruntent une trace, il est possible que ce chemin ne soit plus valable au bout d'un moment, il en est de même si des fourmis exploratrices empruntent un chemin plus long. Par contre, si le chemin est fortement emprunté, chaque nouvelle fourmi qui passe redépose un peu de phéromone et renforce ainsi la trace, donnant alors à ce chemin une plus grande probabilité d'être emprunté [10].
- **Organigramme :**

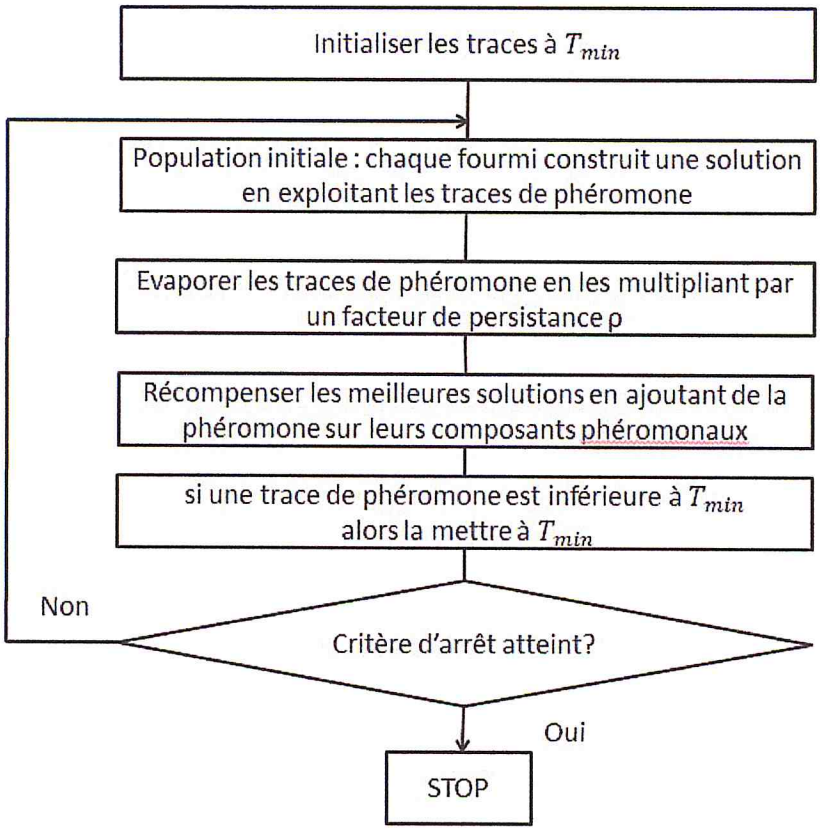



FIGURE 26 – *Organigramme de l'algorithme de colonie de fourmis [10].*

AntClnie

- **Algorithme :**

**Algorithm 5** Colonie de fourmis()

Initialiser les traces de phéromone à $t_{min}$ ;
**repeat**
  Chaque fourmi construit une solution en exploitant les traces de phéromone ;
  Evaporer les traces de phéromone en les multipliant par un facteur de persistance p ;

  Récompenser les meilleures solutions en ajoutant de la phéromone sur leurs composants phéromonaux ;
  **if** (une trace de phéromone est inférieure à $t_{min}$) **then**
    la mettre à $t_{min}$ ;
  **end if**
**until** (critère d'arrêt n'est pas atteint)
Retourner la meilleure solution trouvée ;

## 4.7 Synthèse

La difficulté que nous rencontrons en la matière est multiforme, la plus importante est la diversité des variantes d'un même algorithme.

Nous avons vu précédemment que nous ne connaissons pas d'algorithmes polynomiaux pour les problèmes NP-difficiles. Or, de nombreux problèmes d'optimisation combinatoire sont NP-difficiles et ne pourront donc pas être résolus de manière exacte dans un temps raisonnable.

C'est dans ce contexte que les méthodes approchées ont vu le jour. Elles offrent de bons résultats pas nécessairement optimaux mais de qualité suffisante sur lesquels nous pouvons s'appuyer pour concevoir notre solution.

Cependant, il existe de nombreux algorithmes d'optimisation dans des domaines très variés. Une question fondamentale qui se pose ici est : « Comment pouvons-nous comparer différents algorithmes d'optimisation de manière à pouvoir choisir le plus performant pour un problème d'optimisation donné ? ».

Des protocoles de tests génériques pour la comparaison d'algorithmes existent et nous fournissent des éléments de réponse. En effet, plusieurs fonctions analytiques de tests, regroupées sous l'appellation de « benchmarks », ont été mises en place pour évaluer les performances et les capacités de convergence des algorithmes d'optimisation. Dans ces benchmarks, les optimums globaux sont connus à l'avance. À la fin d'une simulation, il est donc possible d'évaluer l'erreur commise par l'algorithme.

Demeurent toutefois des questions relatives à l'utilité d'une métaheuristique particulière pour résoudre un large éventail de problèmes. Nous pouvons dire qu'aucun algorithme d'optimisation n'est plus adapté que les autres pour résoudre tous les types de problèmes. Néanmoins, cela n'empêche pas certains algorithmes d'être mieux adaptés que d'autres sur des classes particulières de problèmes.

## 4.8 Problème des graphes colorés

En théorie des graphes, colorer un graphe signifie attribuer une couleur à chacun de ses sommets de manière que deux sommets reliés par une arête soient de couleur différente, la notion de coloration n'est définie que pour les graphes sans boucle [51].

En optimisation, nous cherchons à minimiser le nombre de couleurs utilisées c'est à dire étant donné G un graphe donné, quel est le nombre minimum de couleurs nécessaires pour avoir une coloration valide de G ?. De façon plus formelle, pour un graphe défini par un couple G = (V,E) tel que :

— V est un ensemble fini de sommets,

— $E \subseteq V * V$ est un ensemble d'arêtes ;

— f : V $\rightarrow \{1, 2, ..., k\}$ est la fonction de coloration.

Trouver la valeur minimale de $k \in N$ telle que :$f(u) \neq f(v) \forall (u, v) \in E$

### 4.8.1 Complexité

Déterminer le nombre chromatique d'un graphe est un problème NP-difficile dans le cas général. Ainsi, à moins que P=NP, il n'existe pas d'algorithme polynomial déterminant le nombre chromatique d'un graphe arbitraire.

L'importance du problème a donné lieu à l'élaboration de nombreuses heuristiques spécifiques au problème, spécialement des algorithmes séquentiels de coloration sommet par sommet (DSATUR, cosine, maya, etc.). Elle a aussi suscité de nombreuses expérimentations des méthodes approchées générales : méta-heuristique (recuit simulé, recherche tabou), ou encore algorithme génétique. De même une heuristique [35, 36] à été développé au sein de CDTA.

### 4.8.2 Application à notre problématique

Nous avons développé un algorithme pour adapter la technique de coloration de graphes à notre problématique. Ceci sera détaillé dans la deuxième partie de ce mémoire.

## 4.9 Conclusion

Ce chapitre a été consacré aux différentes approches couramment employées pour résoudre un problème d'optimisation. Bien évidemment, ce chapitre n'a pas prétendu à l'exhaustivité mais il a espéré aborder et présenter les principales notions.

Nous avons présenté donc l'état de l'art des méthodes d'optimisation. Après avoir rappelé quelques notions préliminaires, nous avons passé en revue les principaux algorithmes utilisés à nos jours.

Un intérêt particulier a été porté au problème basé sur les graphes colorés comme précédemment dit il nous servira de support au long de ce mémoire pour présenter le travail que nous avons réalisé.

Après avoir détaillé les algorithmes proposés dans la littérature ainsi que dégagé les notions de bases, nous abordons la partie suivante qui représente la partie pratique.

# Deuxième partie

# Conception et implémentation de la solution

# Chapitre 5

# Etude des besoins et solution proposée



## 5.1 Introduction

Tout projet englobant une application informatique, que ce soit dans le domaine de la recherche ou dans le domaine professionnel, doit être en mesure de répondre aux attentes de ses utilisateurs. Ainsi, le développement d'une application nécessite plusieurs phases. L'analyse des besoins étant la première phase, elle permet une meilleure compréhension de ce que nous devons faire pour une meilleure organisation lors des phases de conception et de réalisation.

Cela ne peut, évidemment, se faire sans une étude approfondie des besoins dans un souci de prise en considération des concepts avancés dans le domaine étudié, sans perdre de vue toute possibilité et opportunité offerte par les idées qui peuvent être ajoutées par une personne qui n'a pas une vue globale sur tout le domaine.

L'objectif de ce chapitre s'est donc articulé autour des besoins du projet afin de proposer un programme modélisable. Mais modéliser pour quoi faire ? l'utilisation de schémas et d'illustrations rend quelque chose de complexe plus compréhensible. En général, peu de personnes ont envie de lire un document contiendrait plusieurs dizaines de pages de texte décrivant de façon précise ce qui doit être réalisé. De plus, un long texte de plusieurs pages est source d'interprétations et d'incompréhension.

Partant de cet objectif, nous allons commencer par l'analyse et la spécification des besoins, puis nous allons définir la méthode à employer pour solutionner notre problème. Ensuite, nous présenterons les critères qui argumentent notre étude et enfin nous terminerons par une conclusion.

## 5.2 Analyse des besoins

La phase d'analyse permet de lister les résultats attendus, en termes de fonctionnalités, de performance, de robustesse, de maintenance, d'extensibilité, etc. Elle décrit l'objet à développer en termes de fonctionnalité. En ce sens, elle répond à la question "quoi ?".

Dans la gestion de projets, nous pouvons citer deux approches permettant de définir les besoins :

- La décomposition fonctionnelle (ou l'approche procédurale)
- L'approche objet (sur laquelle est basée UML)

Les applications qui seront réalisées dans un langage basique comme le C doivent subir une décomposition fonctionnelle. L'approche par décomposition fonctionnelle considère que l'application est composée d'une hiérarchie de fonctions et de données. Les fonctions fournissent les services désirés et les données représentent les informations manipulées. Pour réaliser une fonction de notre application, nous pouvons utiliser un ensemble d'autres fonctions, déjà disponibles.

## 5.3 Positionnement du travail dans le projet principal

Nous rappelons que notre tâche s'intègre dans le cadre de la réalisation d'un réseau sur puce. Elle consiste principalement à générer une disposition adéquate des IPs pour leur emplacement ultérieur dans une architecture 3D. La figure 27 indique clairement l'agencement de notre tâche dans la réalisation du réseau sur puce.

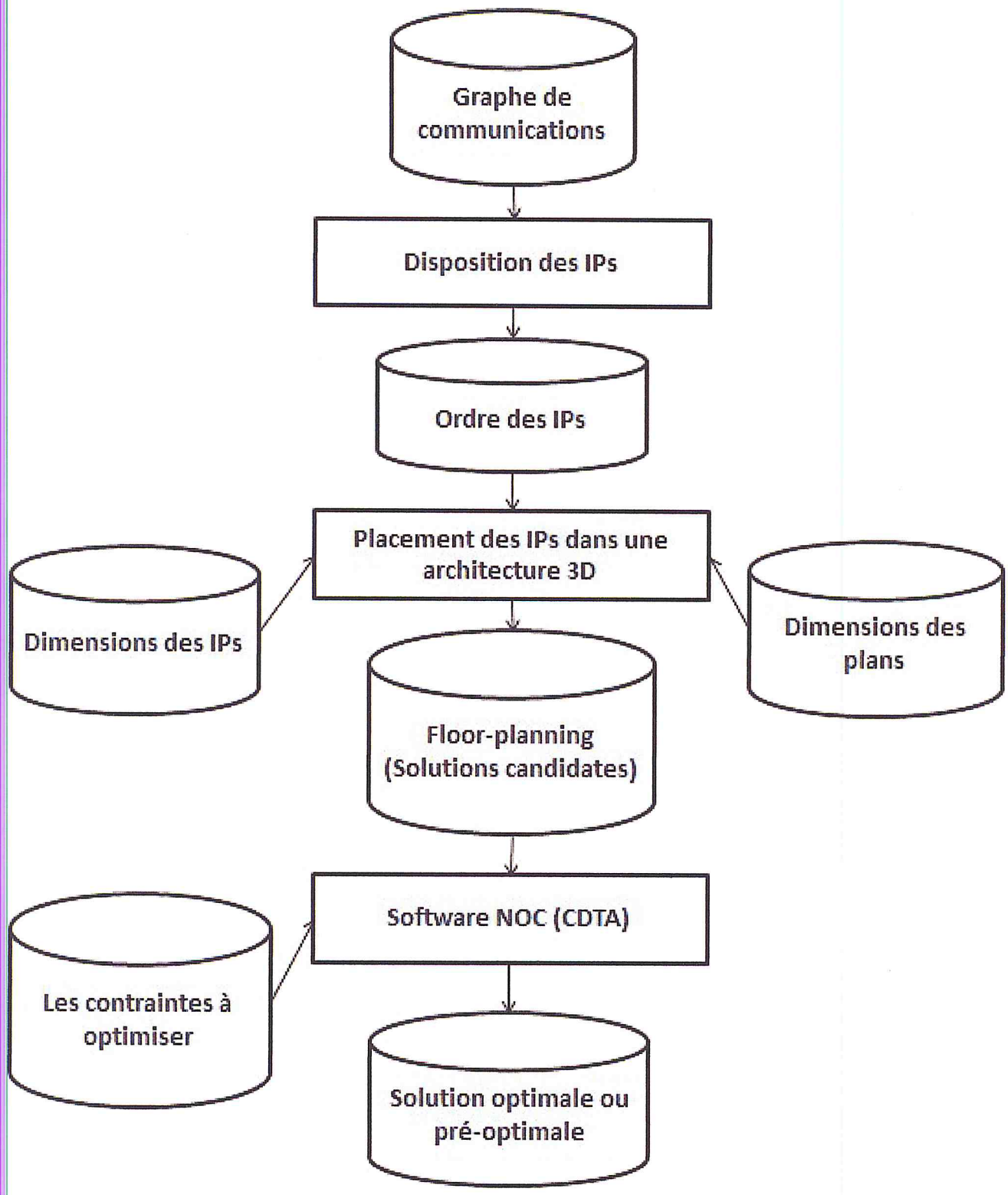



FIGURE 27 – *Description du problème de placement 3D.*

### 5.3.1 Disposition des IPs

La disposition des IPs a un impact important sur les caractéristiques du réseau sur puce en matière de largeur de bande, de surface et de consommation de puissance. L'objectif est alors de placer, autant que possible, les IPs les plus communicants entre eux, côte à côte. Pour cela, nous utilisons un graphe de communication qui sera défini comme suit :

## Graphe de communication

Le graphe de communication G(V, E) (voir figure 28) est un graphe non orienté où chaque sommet (vertex) $v_i \in V$ numéroté correspond à un IP. Deux sommets sont liés entre eux par une arête $e_{ij} \in E$ qui désigne le taux de communication entre les sommets $v_i$ et $v_j$.

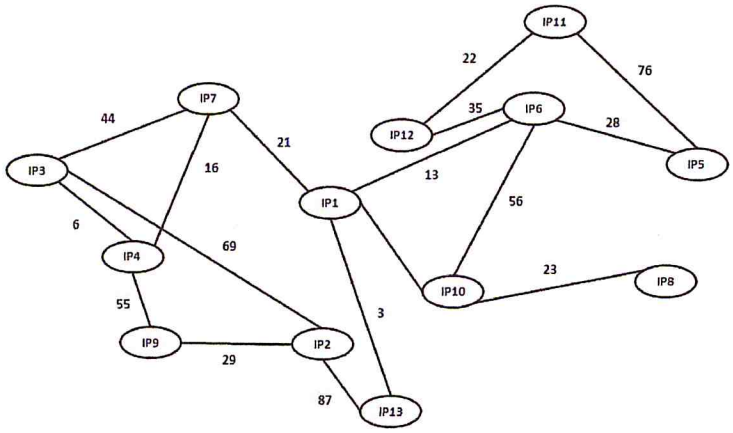



FIGURE 28 – *Graphe de communication.*

A partir du graphe de communication nous construisons la disposition des IPs (Figure 29).

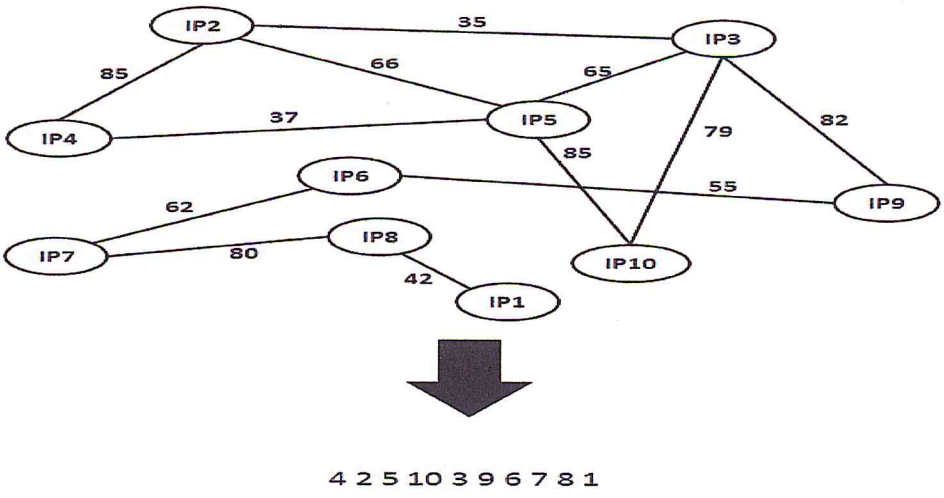



FIGURE 29 – *Disposition des IPs à partir du graphe de communication.*

Il est facile de montrer que ce problème n'est pas de complexité polynomiale, celle-ci étant O (N!), N étant le nombre d'IPs. Ceci nous amène à utiliser une méthode à base d'une heuristique ou d'une métaheuristique.

Dans notre cas, nous avons adapté une heuristique de coloration de graphes à notre problématique. Ceci, en faisant des appels récursifs à la technique utilisée. Ceci sera plus détaillé dans la sous-section 5.4.2.

### 5.3.2 Problématique de placement

Le problème à résoudre dans sa globalité est un problème de placement d'IPs (Composants matériels) dans un circuit intégré 3D de telle façon que les critères (objectifs) soient atteints.

Pour présenter notre problème de manière plus détaillée, nous décrivons dans ce qui suit l'architecture ciblée.

#### Architecture 3D ciblée

La disposition des IPs étant obtenue, il s'agit de les placer sur une architecture 3D (Figure 30). Ce placement obéit aux contraintes imposées par les dimensions des trois plans et celles des IPs.

Afin d'obtenir de bonnes solutions, deux mesures sont prises en considération :

- les IPs sont placés en serpent pour ne pas fausser l'ordre obtenu des IPs.
- utilisation d'une architecture 3D mettant en œuvre des contacts MIV (Monolithic Inertier Via) qui présentent de loin de meilleurs caractéristiques électriques (résistance, capacité et inductance) que celles de TSVs (Through Silicon Via) [19], en plus de l'occupation d'une surface plus faible.

La figure 31 illustre, par un exemple ce qui vient d'être dit.

FIGURE 30 – *Modèle d'architecture 3D.*

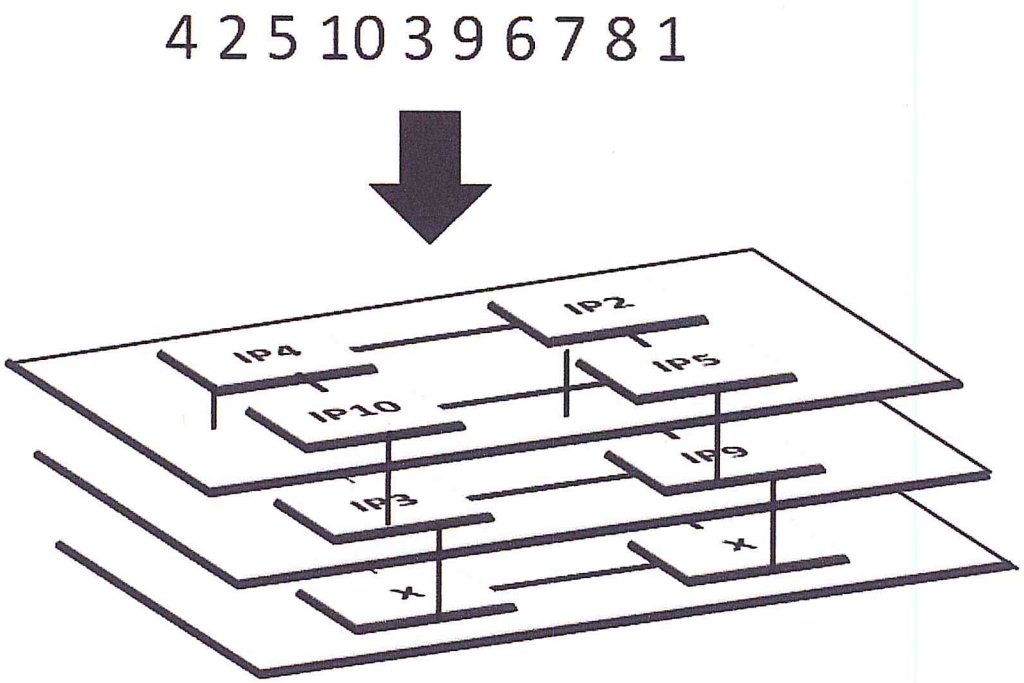



FIGURE 31 – *Placement des IPs dans une architecture 3D.*

## 5.4 Démarche globale proposée

Pour une optimisation efficace d'un problème non polynomial, le mieux serait d'opter pour une méthodologie basée sur une bonne compréhension de la problématique posée. Ainsi, ceci nous permettrait de mieux initialiser la solution, mieux passer d'une solution à une autre, et éviter, autant que possible, la génération aléatoire.

Notre démarche, comme l'indique la figure 32, s'articule sur deux étapes principales :

- Générer les IPs dans un ordre telque les IPs les plus communicants entre eux soient, autant que possible, placés les uns à coté des autres.
- En respectant l'ordre obtenu dans la première étape, placer les IPs sur 1,2 ou 3 plans selon la contrainte de surface posée ultérieurement.

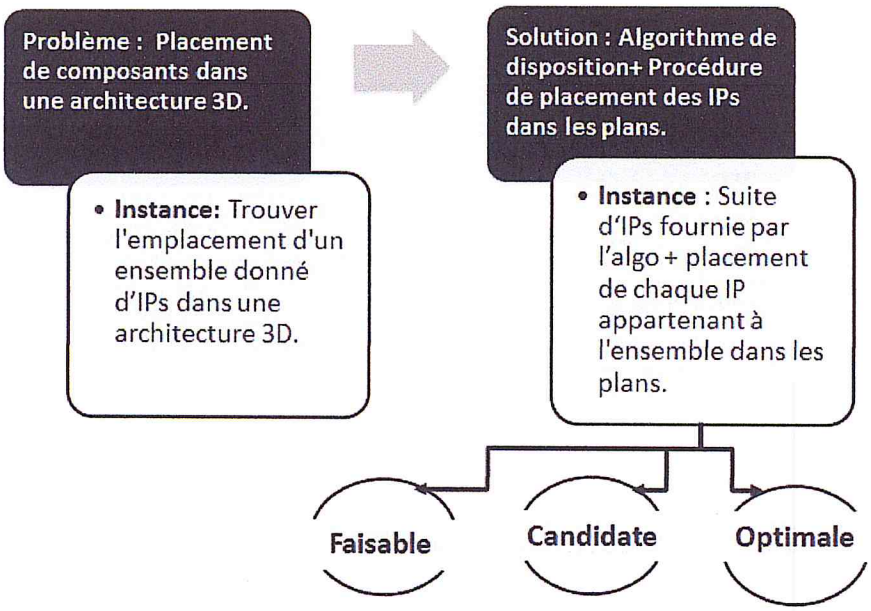



FIGURE 32 – *Contexte de placement 3D.*

La solution qui répond au mieux aux contraintes de largeur de bande, de surface et de consommation de puissance sera retenue.

### 5.4.1 Utilisation des graphes colorés

Comme il a été indiqué auparavant, l'emplacement des IPs par rapport les uns aux autres, a un impact sur les caractéristiques du réseau sur puce. Déterminer le meilleur ordre de manière exhaustive nécessite une complexité algorithmique en O( $N!$), N étant le nombre d'IPs. Pour une valeur conséquente de N, le temps CPU deviendrait exorbitant.

Ceci nous amène alors à développer un algorithme à base d'une heuristique ou d'une métaheuristique. Dans notre cas, nous avons adopté une technique de coloration de graphes à notre problématique pour les raisons suivantes :

- Notre problématique repose sur un problème d'optimisation dont le problème de décision est NP-complet.
- Un outil d'optimisation de ce problème NP-dur a été développé au CDTA [21] et a été testée sur des benchmarks DIMACS pour lesquels il a donné une erreur moyenne de 1.92%.

Les détails d'utilisation de cette heuristique de coloration de graphe sont donnés dans la sous-section suivante.

### 5.4.2 Adaptation des graphes colorés à notre problème

Placer deux IPs côte à côte est assujetti au fait qu'ils sont fortement communicants entre eux. Pour utiliser la technique des graphes colorés à cette problématique, il suffit :

- D'associer un nœud à chaque IP.
- Relier deux nœuds dont les IPs correspondants sont moyennement ou faiblement communicants entre eux, ou ne se communiquent pas.
- Colorer le graphe.
- Les nœuds ayant obtenu la même couleur seront placés dans le même voisinage puisqu'ils présentent la caractéristique d'être fortement communicants entre eux.

Il est clair que si une couleur n'est affectée qu'à 2 nœuds au maximum, le traitement s'arrête. Toutefois, dans le cas général, plusieurs nœuds peuvent avoir la même couleur. Dans ce cas, comment ordonner, de manière efficace, les IPs correspondants à ces nœuds ?

Pour ce faire, il suffit d'opter pour une granularité plus fine que celle définie à l'étape précédente. Plus précisément, si p nœuds ($p > 2$) obtiennent la même couleur, un graphe associé est constitué de la manière suivante :

- Les nœuds sont ceux ayant obtenus une couleur donnée.
- Recalculer la moyenne des degrés de communication concernant ces p nœuds.
- Relier deux nœuds par une arrête si leur degré de communication est inférieur ou égale à cette moyenne.

Ainsi, à chaque étape i, il y'aura Mi graphes à colorer, Mi étant le nombre de couleurs associées à plus de deux nœuds.

L'appel récursif à la technique de coloration de graphes s'arrête lorsque chaque couleur colore au maximum deux nœuds.

A titre illustratif, nous donnons l'exemple indiqué à la figure 33, nous supposons, que deux nœuds donnés sont reliés par une arrête ont un degré de communication inférieur ou égal à la moyenne des degrés de communication.

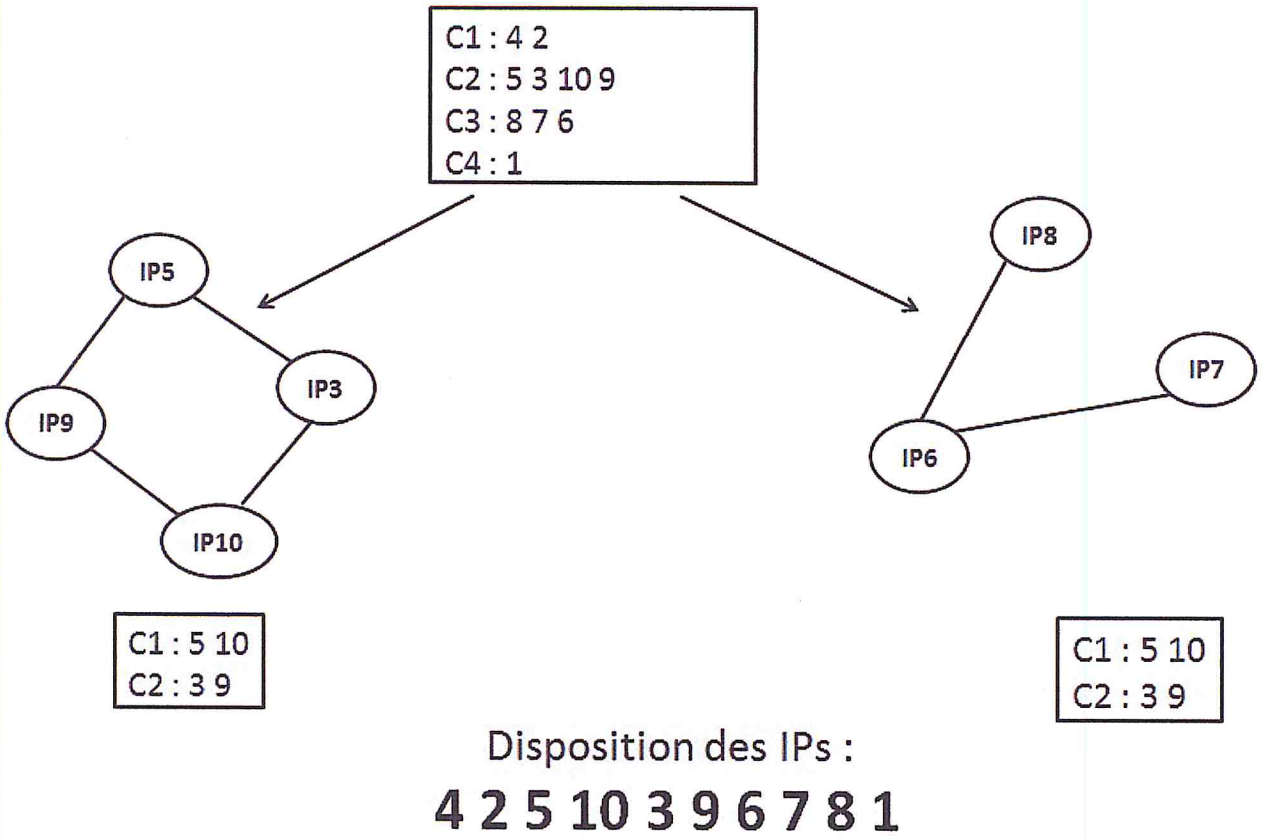



FIGURE 33 – *Disposition des IPs en utilisant l'appel récurssif de la technique.*

## 5.5 Synthèse ou Discussion de choix

Lorsque nous faisons face à un problème d'optimisation, nous somme confrontés à des problèmes difficiles et de taille importante. Nous pouvons alors avoir recours aux méthodes approchées, en se contentant de rechercher une solution de « bonne qualité ».

Dans ce cas, un choix s'offre à nous : voulons-nous opter pour une technique d'optimisation qui convient au problème (une heuristique spécialisée entièrement dédiée au problème considéré) ou au contraire, une optimisation aléatoire (en utilisant une métaheuristique générique) ? Les critères pour faire notre choix tenaient au nombre de trois : le contexte général, les principales méthodes existantes, et les facteurs d'initialisation.

Les métaheuristiques utilisées en lieu et place d'heuristiques spécialisées montrent généralement de moins bonnes performances.

Les métaheuristiques peuvent ne pas trouver la solution optimale, et encore moins prouver l'optimalité de la solution trouvée.

## 5.6 Conclusion

Cette étude a constituée le point de départ pour définir le périmètre du projet. Elle nous a permis d'avoir une vision générale sur la démarche à suivre, du fait qu'elle a ressorti les besoins du système en soulignant les étapes principales de la réalisation de ce projet.

Nous avons alors essentiellement décrit, dans ce chapitre la démarche globale adoptée.Celle-ci sera détaillée dans le chapitre suivant.

# Chapitre 6

# Conception et mise en œuvre de la solution

## 6.1 Introduction

Afin de faciliter la compréhension de l'application, d'éviter les anomalies et de simplifier une éventuelle mise à jour, il est important que l'application soit bien conçue et définie de manière claire pour une utilisation future.

Dans la phase de conception, nous apportons généralement plus de détails à la solution et nous cherchons à clarifier les aspects techniques, pour cela, avant de passer à la réalisation et la mise en place de la solution, il a été nécessaire de recourir à un certain nombre d'outils, ainsi qu'à la mise en place des environnements d'exécution.

Ce chapitre a pour objectif de décrire l'environnement mis en place et les outils utilisés, ainsi que de décrire l'environnement existant (matériels et logiciels) dans lequel évoluera notre outil.

## 6.2 Conception de la solution

La phase de conception permet de décrire de manière non ambiguë le fonctionnement futur du système, afin d'en faciliter la réalisation. Elle décrit l'ensemble des moyens et procédures permettant de produire cette fonctionnalité, et répond en ce sens à la question "comment ?".

Dès qu'elle dépasse un certain nombre de lignes de code, une application informatique, surtout si elle sera intégrée dans un outil plus général, doit obligatoirement être structurée sous peine d'être inutilisable dans un futur proche. Pour cela, nous avons découpé notre application en deux niveaux :

— en plusieurs modules.
— en plusieurs fonctions à l'intérieur d'un module donné.

Nous donnons ici quelques indications sur la méthode que nous avons adoptée pour décomposer notre application en plusieurs modules.

Tout d'abord, nous devons respecter un certain nombre de règles :

1. une fonction ne doit pas être de longueur trop importante.
2. une fonction doit, comme son nom l'indique, avoir un traitement bien précis dans le programme et non plusieurs traitements différents.
3. un module (fichier .c) ne doit pas être de longueur démesurée lui non plus. Il est beaucoup plus facile de gérer 10 modules d'une centaine de lignes chacun qu'un seul module de 1000 lignes. Cette organisation nous permet du gagner du temps à tous les niveaux : temps de mise au point, temps de compilation, ...
4. un module doit, lui-aussi, avoir un rôle précis : il doit regrouper des fonctions correspondant à des fonctionnalités proches.

Pour illustrer ces règles, dans le cadre de notre projet, Nous avons défini les modules suivants :

— un module spécialisé dans la génération de graphe, appelé "principale.c". Celui-ci contient les fonctions primitives de manipulation d'un graphe telles que cal_ poids_ moy() et gen_ graphe().
— un module spécialisé dans le placement des IPs, nommé "placement.c". Celui-ci contient les fonctions primitives suivantes : remplir_ vecteur(), taille_ Boite(), estimation(int nb_ elts), refine_ floorplanning(FILE* ptr, int nbr_ ips, int nb_ elts_ par_ ligne), lire_ graphe_ coloree(char chaine_ fichier[25]) et ordonner(int niveau,char chaine_ fichier[25]).
— un module contenant les appels des opérations déjà définies et la manipulation de ces dernières (main.c).

Ces modules et fonctions seront présentés ultérieurement.

## 6.3 Périmètre technique et fonctionnel

Dans cette partie nous allons décrire les infrastructures mises en place. En effet, cette étape est à ne pas négliger, car la diversité des plateformes techniques pourra engendrer des problèmes de compatibilité.

### 6.3.1 Matériel

Les systèmes sources sont installés sur différentes plateformes :

— Machines **SAMSUNG** :

- **Processeur :** Core(TM)i5 CPU 2.67 GHz.
- **Mémoire :** 4.00 Go.

— Machines **HP** :

- **Processeur :** 2 Core CPU 2 GHz.
- **Mémoire :** 1.00 Go.

### 6.3.2 Systèmes d'exploitation

Lors de notre étude, nous avons utilisé le système d'exploitation suivant :

— LINUX : OpenSUSE 11.1.

### 6.3.3 Logiciel

Pour la réalisation de l'interface graphique nous avons utilisés :

— L'IDE : CodeBlocks.

— La libririe : wxWidgets (GUI[1]).

## 6.4 Architecture technique de la solution

La figure suivante illustre la structure et l'architecture technique de la solution proposée :

1. Graphicals Users Interface.

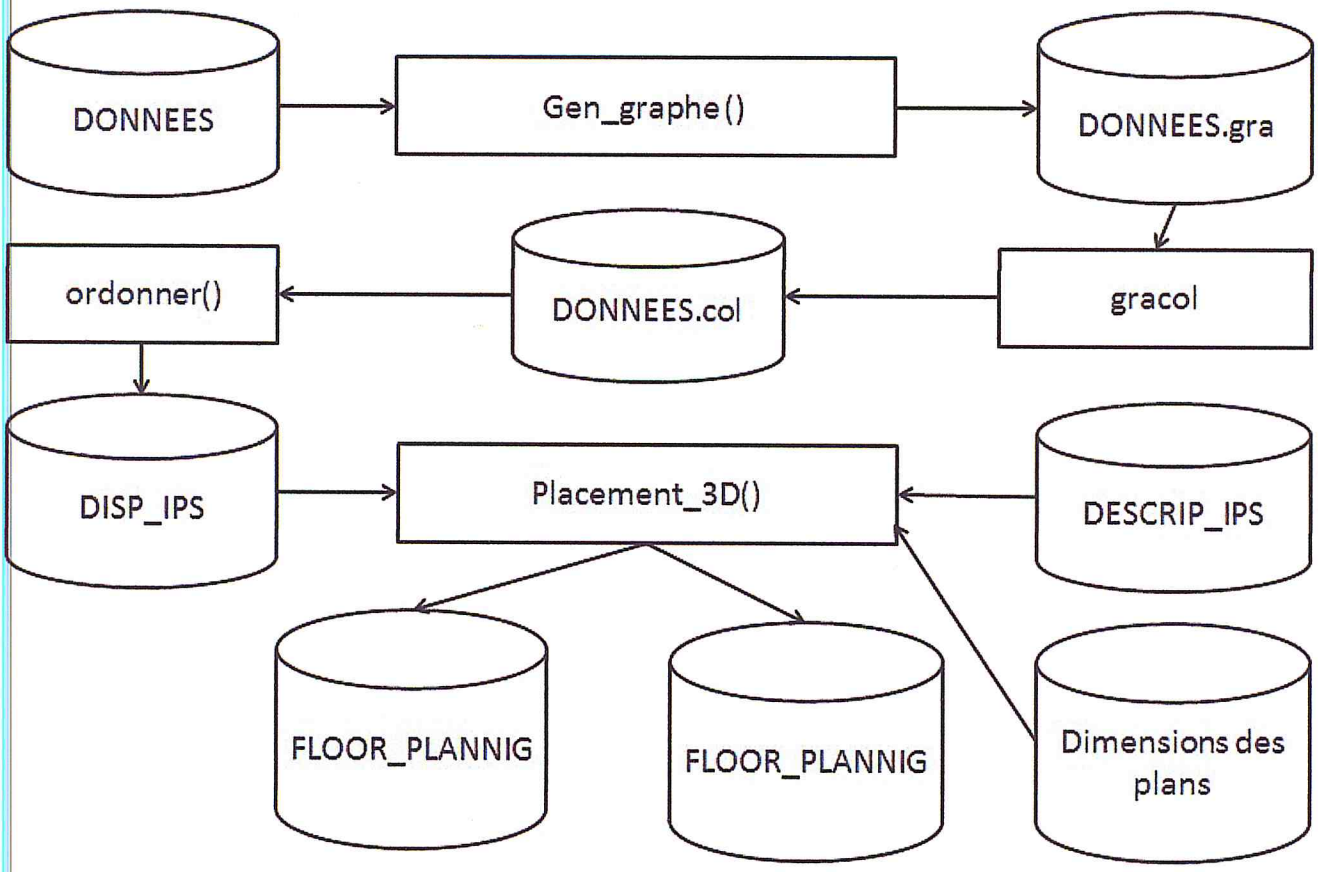



FIGURE 34 – *Diagramme de l'algorithme principal.*

- **Gen-graphe () :** relie deux nœuds dont les IPs correspondants sont moyennement ou faiblement communicants entre eux, ou ne se communiquent pas.
- **gracol :** SOFTWARE développé au CDTA [35, 36] pour Colorer le graphe.
- **ordonner() :** Squelette dans la figure 35.
- **Placement 3D :** donne la disposition visuelle des IPs.
- **DONNEES, DISP_IPS, ...etc :** fichiers d'enregistrement qui seront détaillés dans la section (Mise en œuvre).

### 6.4.1 Diagramme de disposition

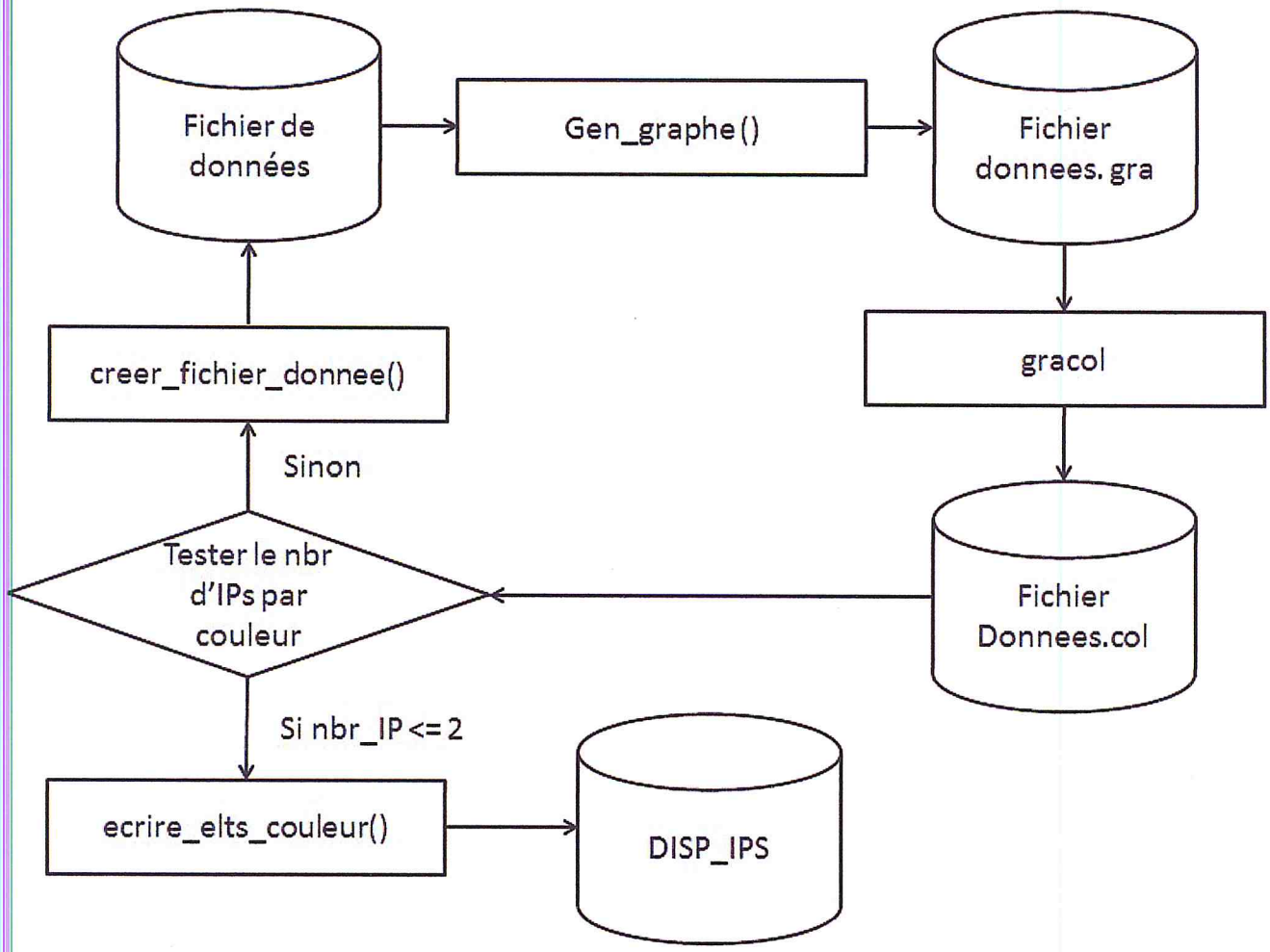



FIGURE 35 – *Diagramme de disposition correspondant à la fonction ordonnee().*

- **creer_fichier_donnee() :** si p nœuds (p > 2) obtiennent la même couleur, un graphe associé aux p nœuds ayant obtenus une couleur donnée sera constitué.
- **ecrire_elts_couleur :** si une couleur n'est affectée qu'à 2 nœuds au maximum, le traitement s'arrête. Les nœuds qui correspondent a cette couleur seront écris dans un fichier.

### 6.4.2 Diagramme de placement

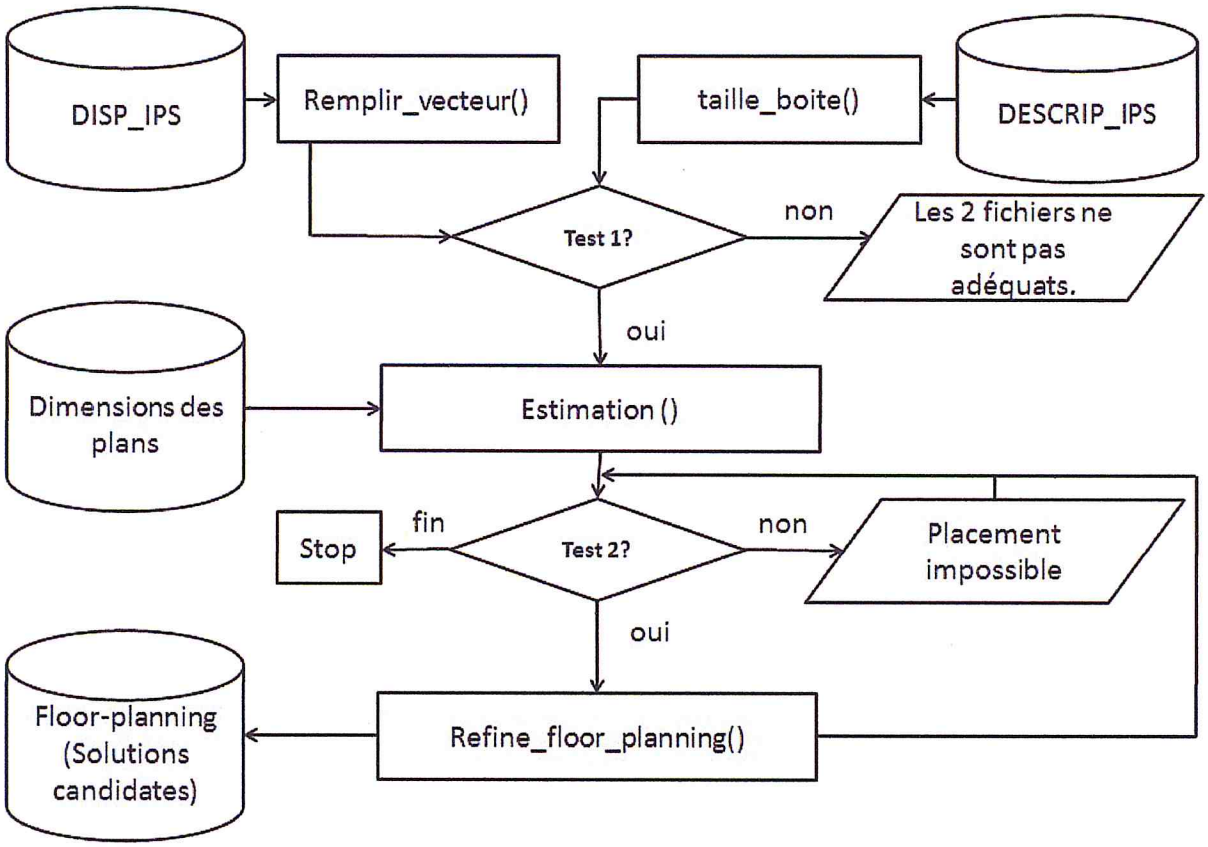



FIGURE 36 – *Diagramme de placement 3D en utilisant la fonction refine-floorplanning.*

- **Remplir_vecteur() :** calcule le nombre d'IPs dans le fichier DISP_IPS.
- **taille_boite() :** détermine la taille de la boite qui va contenir tous les nœuds, et retourne le nombre d'IPs du fichier DESCRIP_ IPS.
- **Test1() :** vérifie si le nombre d'IPs dans les deux fichiers est le même !
- **Estimation() :** Cette fonction nous permet d'estimer le nombre maximum d'IPs que nous pouvons placer dans un plan, le nombre maximum d'éléments par ligne, ainsi que le nombre maximum d'éléments par colonne.
- **Test2() :** vérifie si nous pouvons placer tous les IPs sur la surface disponible.
- **Refine_floor_planning() :** La fonction refine_floorplanning() utilise toutes les informations obtenues pour positionner les IPs sur les plans.

## 6.5 Mise en œuvre

Nous montrons dans cette section comment mettre en œuvre les principes que nous avons acquis lors de la phase de conception.

## Étape 1 : les fichiers "headers"

Ces fichiers "headers" appelés encore en-tête contiennent les prototypes des fonctions. Il faut inclure les fichier .h grâce à une directive de préprocesseur dans les fichiers sources adéquats. Ainsi, les fonctions peuvent être accessibles dans les différents modules. Pour cela, nous avons créé le fichier "placement.h" contenant :

```
/*
* placement.h
*/
# define LAMBDA 2e-06
# define HIGH_ PLAN 0.5e-02
# define WIDTH_ PLAN 0.6e-02
# define SURFACE_ PLAN (HIGH_ PLAN * WIDTH_ PLAN)
# define NBR_ MAX_ IPS 100
# define NBR_ PLAN 3
# define NBRE_ INTERCONNEXIONS 3
# define TAILLE_ INTERCONNEXIONS 8
# define N_ TOTAL_ INTERCONNECTS (NBRE_ INTERCONNEXIONS * TAILLE_
INTERCONNEXIONS)
# define W_ INTERCONNECT 5e-05
\\ prototypes des fonctions
int remplir_ vecteur();
int taille_ Boite();
void estimation(int nb_ elts);
void refine_ floorplanning(FILE* ptr, int nbr_ ips, int nb_ elts_ par_ ligne);
void lire_ graphe_ coloree(char chaine_ fichier[25]);
void ordonner(int niveau,char chaine_ fichier[25]);
```

Ce fichier sera alors inclus dans chaque module nécessitant l'utilisation de l'une de ces constantes de préprocesseur ou l'une de ces fonctions. Pour cela, on indique en début de module : # include "placement.h"

## Étape 2 : les fichiers sources

Le principe de la programmation modulaire est tout simple : plutôt que de placer tout le code de notre programme dans un seul fichier (main.c), nous le « séparons » en plusieurs petits fichiers "principal.c", "placement.c". Ces fichiers contiennent les fonctions elles-mêmes. Dans cette étape nous avons codé les fonctions nécessaire pour le bon fonctionnement du programme.

Le premier champ correspond au numéro d'IP, le deuxième à la hauteur de l'IP et le troisième à la largeur de l'IP.

### Les fichiers intermédiaires

Un fichier intermédiaire ou temporaire est un fichier qui est créé par le programme pour y stocker des informations qui ne sont utiles que pendant sa durée d'exécution. Nous avons utilisé quatre fichiers de ce genre :

- **COMMUNICATION_ ind_ind :** ces fichiers décrivent les graphes de communication partiels entre les IPs qui ont la même couleur seulement. Ils seront générés par le programme à partir du fichier "DONNEES" et il ont la même forme que ce dernier.
- **Nom_ de_ fichier.gra :** ce fichier décrit le graphe qui sera utilisé par la fonction de coloration. Nous l'obtiendrons par l'application de gen_ graphe. Un fichier de type ".gra" valide doit contenir, dans chaque ligne, soit un seul champ suivi par FIN ou 2 champs suivis par FIN. Une ligne Spéciale de fin de fichier se définie par un END. La forme générale d'un tell fichier est :

  1 FIN
  2 FIN
  4 3 FIN
  END

- **Nom_ de_ fichier.col :** ce fichier associe à l'ensemble d'IPs le minimum possible de couleurs (heuristique). Il est généré par la fonction de coloration. Un fichier de type ".col" doit contenir, dans chaque ligne, la couleur du groupe d'IPs suivi par 2 points : les IPs qui prennent cette couleur séparés par des blancs Pour indiquer la fin de ligne nous utilisons le mot clé FIN. La forme générale d'un tel fichier est :

  C1 : 4 2 FIN
  END

  Tant que trois ou plusieurs IPs, ont la même couleur, une procédure est récursivement appelée. Un exemple en est donné à la figure 37.

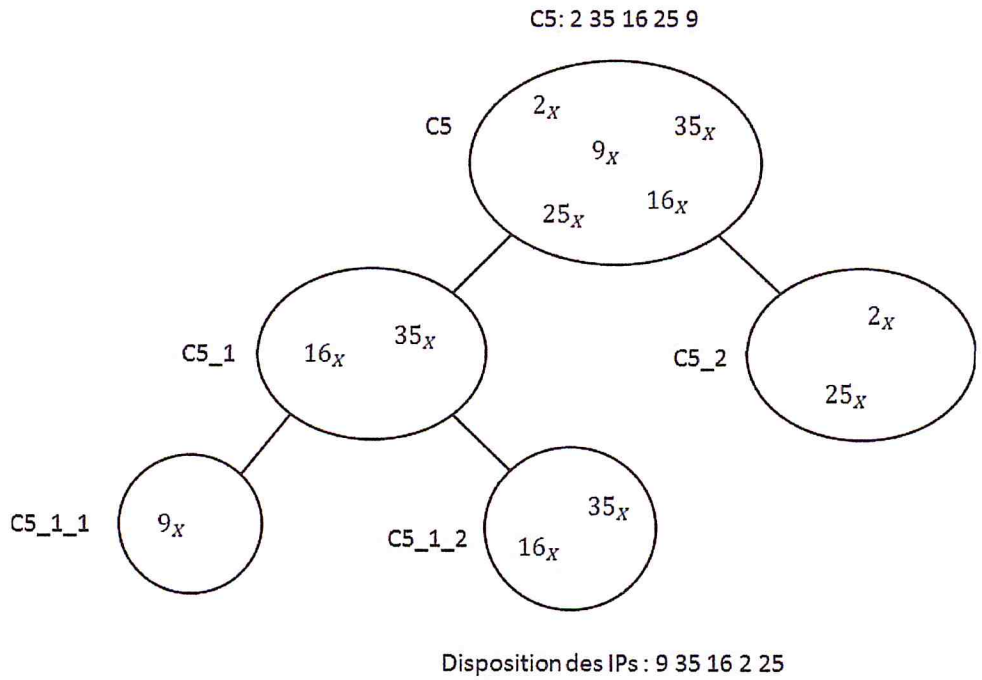



FIGURE 37 – *Colorations de graphes récursives pour ordonner les IPs selon le critère"degré de communication".*

- **DISP_ IPS :** ce fichier contient l'ordre optimal ou pré-optimal du placement des IPs. Il comporte tous les IPs ordonnés de gauche à droite séparés par des blancs. Un exemple d'un fichier quelconque de ce type en est le suivant :

  1 8 4 7 3 5 2 6 9

### Les fichiers de sortie

C'est dans ces fichiers que nous obtiendrons les résultats finaux de notre programme. Nous avons défini deux fichiers de ce genre.

- **CPU_ placement :** Ce fichier contient le temps nécéssaire pour l'exécution de notre programme. C'est un fichier secondaire indiquant les performances de notre programme en temps d'exécution.
  Une ligne type de ce fichier ressemble à ce qui suit :

  0 S souna 421 313 0 80 0 - 451 wait 21 :24 pts/2 00 :00 :00 placement

- **FLOOR_ PLANNING :** ce fichier contient le résultat voulu par la réalisation de ce programme. Il englobe les différentes configurations possibles suivant la disposition obtenue dans le fichier "DISP_ IPS". Il indique l'emplacement de l'IP à l'intérieur d'un plan et dans quel plan.

## Étape 5 : la compilation séparée

Pour compiler, il faut générer un fichier objet (extension .o) pour chaque module. Ensuite, il faut éditer les liens entre ces objets pour obtenir un programme exécutable. Par exemple, nous aurons les commandes suivantes :

cc - g -c placement.c

cc -g -c main.c

cc -g -o placement main.o placement.o -lm

Les deux premières invocations de cc génèrent le fichier objet de chacun des modules sources. La troisième entraîne l'édition de liens et la création du fichier exécutable placement.

D'une manière plus générale, la solution la plus simple pour réaliser la compilation d'un ensemble de modules est d'écrire un fichier Makefile. Utilisé par la commande gmake, un fichier Makefile spécifie les dépendances entre les modules d'une application.

De plus, gmake prend garde de ne déclencher que les traitements vraiment nécessaires : si un module source n'a pas été modifié depuis que son objet à été généré, il n'est pas de nouveau compilé. Nous pouvons ainsi gagner en temps de compilation et surtout d'éviter des erreurs (par exemple oublier de recompiler un fichier qui a été modifié).

Sous sa forme la plus simple, un fichier Makefile dans notre cas aura la forme suivante :

```
/*
* makefile
*/
placement : main.o placement.o
            cc -g -o placement main.o placement.o -lm
main.o : main.c placement.h
            cc -g -c main.c
placement.o : placement.c placement.h
            cc -g -c placement.c
```

Chaque règle est ici composée de deux lignes. Prenons la première règle :

```
placement : main.o placement.o
            cc -g -o placement main.o placement.o -lm
```

Elle indique que le fichier placement dépend des deux fichiers main.o et placement.o. Si l'un de ces deux fichiers est plus récent que placement, alors l'action pour le régénérer est déclenchée. Cette action, la ligne suivante, appelle cc pour éditer les liens entre main.o et placement.o et créer l'exécutable placement. Les deux fichiers main.o et placement.o dépendent eux-aussi d'autres fichiers ; les deux règles qui suivent l'indiquent et nous retrouvons une action qui permet de les générer si nécessaire.

gmake cherche à engendrer le premier fichier associé à la première règle trouvée dans le fichier Makefile. Ici, il cherche donc à générer un fichier placement.

gmake est un logiciel sophistiqué dont l'utilisation complète dépasse de loin l'objet de cette partie.

## Étape 6 : poursuite du projet

Une fois les primitives mises au point, nous pouvons compléter l'application par la création d'une interface graphique qui rend l'utilisation du l'outil beaucoup plus agréable et conviviale. Pour cela, nous avons utilisé l'IDE codeblocks et la bibliothèque wxWidgets.

Il est également classique de définir un module erreur.c qui affiche les messages d'erreurs et provoque éventuellement l'arrêt du programme. Les autres modules font appel à une fonction de erreur.c lorsqu'une condition d'erreur est rencontrée. Ainsi, la gestion des erreurs est centralisée.

Il est beaucoup plus facile d'avoir ainsi des messages d'erreur qui ont un aspect uniforme pour toute l'application. Il est également ainsi beaucoup plus facile de modifier tous les messages d'erreur que de devoir parcourir tous les modules source pour trouver et modifier les messages d'erreur.

## Étape 7 : utilisation de répertoires

Les fichiers constituant une application doivent être rangés dans des répertoires distincts pour ne pas mélanger sources, en-têtes, objets et exécutables. Ainsi, on utilise généralement :

- un répertoire nommé src qui contient tous les fichiers source.
- un répertoire nommé o qui contient tous les fichiers objet.
- un répertoire nommé h qui contient tous les fichiers en-tête.

Cela impose de spécifier les répertoires dans le Makefile et d'ajouter une option -I... à cc pour qu'il trouve les fichiers en-têtes. Ainsi, pour la notre cas, nous pourrons avoir la répartition suivante des fichiers :

- un répertoire racine contenant Makefile et main ;
- un répertoire racine/src contenant tous les modules source.
- un répertoire racine/o contenant tous les fichiers objet.
- un répertoire racine/h contenant tous les fichiers en-tête.

Le fichier Makefile pourrait être défini comme suit :

```
/*
* makefile
*/
SRC=/src
O=/o
H=/h
placement : $ (O) main.o $ (O) placement.o
            cc -g -o placement $(O) main.o $(O) placement.o -lm
$(O) main.o : $(SRC) main.c -I$(H) placement.h
            cc -g -c $(SRC) main.c
$(O) placement.o : $(SRC) placement.c -I$(H) placement.h
            cc -g -c $(SRC) placement.c
```

## 6.6 Conclusion

La phase de conception est très importante dans n'importe quel projet informatique, dans la mesure où elle permet la maintenance de ce dernier, garantit son intégrité et facilite son expansion.

Dans ce chapitre nous avons essayé de concevoir notre système qui répond aux exigences exprimées dans le chapitre précédent.

Ainsi il est plus que nécessaire de respecter la conception lors du développement de l'application.

L'algorithme qui a été conçu pour résoudre notre problème d'optimisation combinatoire avec contraintes sera analysé en utilisant différents jeux de tests dans le chapitre qui suit.

# Chapitre 7

# Tests et résultats

## 7.1 Introduction

Après avoir expliqué notre technique et la procédure suivie pour la mettre en œuvre dans les deux derniers chapitres, ce chapitre a pour principale vocation d'exposer et de décrire les tests que nous avons effectués ainsi que les résultats obtenus lors de nos expérimentations.

Nous commencerons par présenter les différents fichiers sur lesquels nous avons effectué nos tests, ainsi que le comportement de l'application en utilisant ces données. Puis, nous étudierons les résultats obtenus après la réalisation des tests.

## 7.2 Exemple de déroulement

Cette section permet de mieux comprendre le comportement de l'application à l'aide d'un exemple illustratif. Le nombre d'IPs dans cet exemple est relativement petit (6 IPs) pour faciliter la compréhension des étapes.

Nous présentons en premier lieu les 2 fichiers d'entrées cités précédemment et qu'il faudrait utiliser pour permettre le bon fonctionnement de l'application.

Le fichier de données :

```
/******
* DONNEES
******/

\\ extr1 extr2 taux de communication
   1     2     11
   1     3     6
```

| | | |
|---|---|---|
| 1 | 4 | 0 |
| 1 | 5 | 0 |
| 1 | 6 | 0 |
| 2 | 3 | 37 |
| 2 | 4 | 42 |
| 2 | 5 | 0 |
| 2 | 6 | 38 |
| 3 | 4 | 0 |
| 3 | 5 | 29 |
| 3 | 6 | 0 |
| 4 | 5 | 0 |
| 4 | 6 | 45 |
| 5 | 6 | 0 |

Ce fichier correspond au graphe de communication suivant :

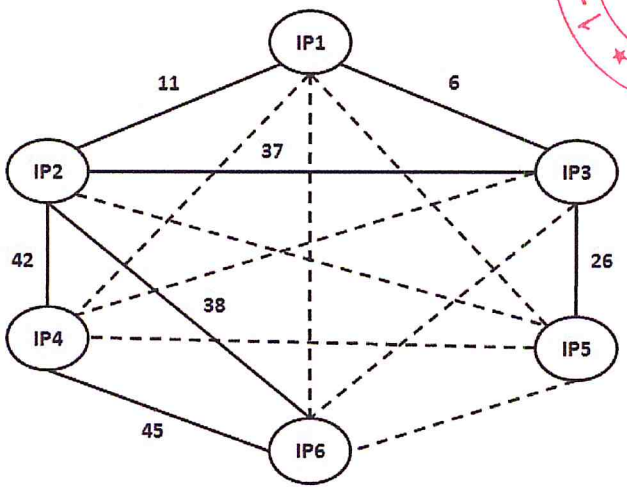



FIGURE 38 – *Graphe de communication qui correspond au fichier de DONNEES de notre exemple.*

Le fichier de description des IPs :

/ * * * * * *
* DESCRIP_ IPS
* * * * * * /

\\ Num_ IP Longueur_ IP Largeur_ IP

| | | |
|---|---|---|
| 5 | 1.2e-03 | 1e-03 |
| 2 | 1e-03 | 0.9e-03 |
| 4 | 0.8e-03 | 1.15e-03 |
| 3 | 1.13e-03 | 1.15e-03 |

| | | |
|---|---|---|
| 1 | 0.96e-03 | 0.85e-03 |
| 6 | 0.9e-03 | 0.75e-03 |

## Étape 1 : utilisation de fichier de données

Tout d'abord, le programme principal va exécuter "gen_ graphe DONNEES" pour générer le Fichier intermédiaire "DONNEES.gra".

L'idée de base de cette fonction consiste à calculer le taux de communication moyen (ou Poids_ Moy). Dans notre exemple, Poids_ Moy vaut 13, puis de comparer le taux de communication entre 2 IPs au taux moyen de communication. (cette comparaison revient à déterminer si deux IPs sont fortement communicant ou pas).

Si le taux de communication entre deux IPs est supérieur ou égal au taux de communication moyen, ces deux derniers seront écris dans deux lignes différentes. Sinon, ils seront écris dans la même ligne.

Le fichier de DONNEES.gra des IPs :

```
/******
* DONNEES.gra
******/
1 2 FIN
1 3 FIN
1 4 FIN
1 5 FIN
1 6 FIN
2 FIN
3 FIN
2 FIN
4 FIN
2 5 FIN
2 FIN
6 FIN
3 4 FIN
3 FIN
5 FIN
3 6 FIN
4 5 FIN
4 FIN
6 FIN
```

5 6 FIN
END

Ce fichier correspond au graphe suivant :

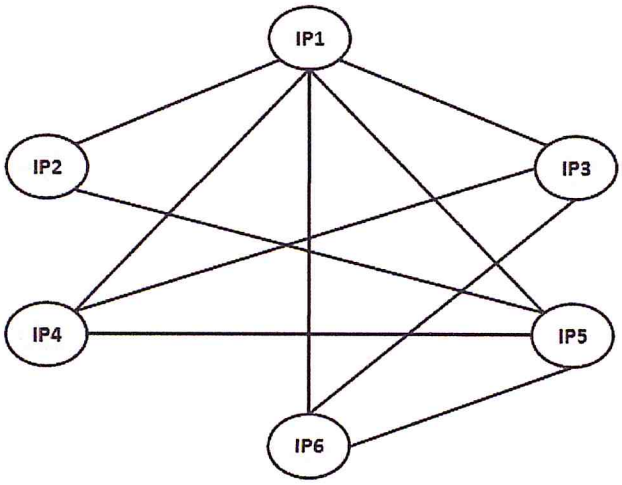



FIGURE 39 – *Graphe qui correspond au fichier "DONNEES.gra" de notre exemple.*

## Étape 2 : utilisation de fichier DONNEES.gra

Après avoir généré le fichier "DONNEES.gra", il sera utilisé par l'exécutable "gracol" (cet exécutable représente l'heuristique [21] mise en œuvre par une équipe de la division micro et nano-technologie du CDTA) pour produire le fichier "DONNEES.col" qui se présente comme suit :

/ * * * * * *
* DONNEES.col
* * * * * * /
C3 : 1 FIN
C2 : 5 3 FIN
C1 : 4 6 2 FIN
END

L'heuristique "gracol" cherche à minimiser le nombre de couleurs à utiliser dans le graphe "DONNEES.gra". Dans notre cas, cela revient à ce que les nœuds qui se communiquent fortement auront probablement la même couleur (c'est à dire ils ont plus de chance d'être côte à côte) alors que les nœuds qui ne communiquent pas souvent auront des couleurs différentes (c'est à dire qu'ils ne sont pas nécessairement mis côte à côte), ce qui est le but recherché.

## Étape 3 : utilisation de fichier DONNEES.col

Nous exploitons le fichier "DONNEES.col" de la manière suivante :

— Si une couleur ne contient pas plus de deux nœuds ce(s) nœud(s), ils seront écrits dans le fichier "DISP_ IPS".
— Sinon, nous devrons utilisé la fonction ordonner() (pour regénérer le fichier de données qui correspond à ces nœuds. L'appel à cette fonction se fait de manière récursive).

Nous obtiendrons dans le fichier "DISP_ IPS" le résultat intermédiaire suivant :

```
/******
* DISP_ IPS
******/
1 5 3
```

## Étape 4 : utilisation de la fonction ordonner()

Nous avons obtenu dans la couleur C1 3 nœuds (4, 6 et 2) donc le programme va exécuter la fonction récursive ordonner() qui va créer les fichiers suivants :

- Le fichier "communication_ 1_ 1" représentant du graphe de communication entre les nœuds (4, 6 et 2) :

```
/******
* communication_ 1_ 1
******/

\\ extr1 extr2 taux de communication
   4     6     45
   4     2     42
   6     2     38
```

Le taux de communication moyen dans ce cas vaut : 41.

- Le fichier "communication_ 1_ 1.gra" :

```
/******
* communication_ 1_ 1.gra
******/
4 FIN
6 FIN
4 FIN
2 FIN
6 2 FIN
END
```

- Le fichier "communication_ 1_ 1.col" qui se présente comme suit :

/******
* communication_ 1_ 1.col
******/
C2 : 2 FIN
C1 : 6 4 FIN
END

Nous n'avons plus besoin d'utiliser l'appel récursif, donc nous obtiendrons le fichier "DISP_ IPS" suivant :
/******
* DISP_ IPS
******/
1 5 3 2 6 4

Nous avons jusqu'à maintenant illustré les résultats obtenus avant l'utilisation de la disposition 3D. Le but de cette illustration était de comprendre la démarche à suivre pour avoir le contenu de fichier de disposition des IPs.

## Étape 5 : utilisation de fichier DESCRIP_ IPS

Nous avons utilisé ce fichier via la fonction taille_ Boite. Cette fonction lit la largeur et la longueur de tous les IPs à partir du fichier "DESCRIP_ IPS", puis elle modifie respectivement la largeur et la longueur maximale de la boite qui va contenir tous les IPs (architecture matricielle) et retourne le nombre d'IPs dans ce fichier.

Dans notre exemple, les résultats obtenus, après l'exécution de cette fonction sont les suivants :
Nombre d'IPs dans le fichiers description est : 6
La hauteur de la boite = 0.001200 m
La largeur de la boite = 0.001150 m

## Étape 6 : utilisation de la fonction estimation()

Cette fonction nous permet d'estimer le nombre maximum d'IPs que nous pouvons placer dans un plan, le nombre maximum d'éléments par ligne, ainsi que le nombre maximum d'éléments par colonne. Ceci, en utilisant la taille de la boite et la taille du plan (tous les plans ont la même taille) définies dans le fichier "placement.h".

Dans notre exemple, les résultats obtenus, après l'exécution de cette fonction sont les suivants :

Nombre d'IPs dans le fichiers description est : 6
nbr_ max_ elts_ par_ ligne = 5
nbr_ max_ lignes = 3
nbr_ max_ elts_ plan = 15

## Étape 7 : obtention du résultat final

La fonction refine_ floorplanning() utilise toutes les informations obtenues (le fichier DISP_ IPS, nbr_ max_ elts_ par_ ligne, nbr_ max_ lignes, nbr_ max_ elts_ plan) pour positionner les IPs sur les plans.

En s'inspirant de la marche du serpent utilisée pour ne pas fausser le résultat obtenus dans le fichier "DISP_ IPS" par exemple dans la (figure 40) l'$IP_3$ doit être placer en dessous de l'$IP_1$ plutôt qu'à la première colonne, la fonction refine_ floorplanning() place si c'est possible 1,2, ...,nbr_ max_ elts_ par_ ligne élément(s) par ligne. Si nous dépassons la capacité d'un plan nous aurons recours au plan suivant.

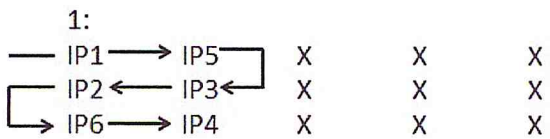



FIGURE 40 – *Méthode de placement.*

La figure 41 montre le résultat d'exécution de notre exemple en mode console.

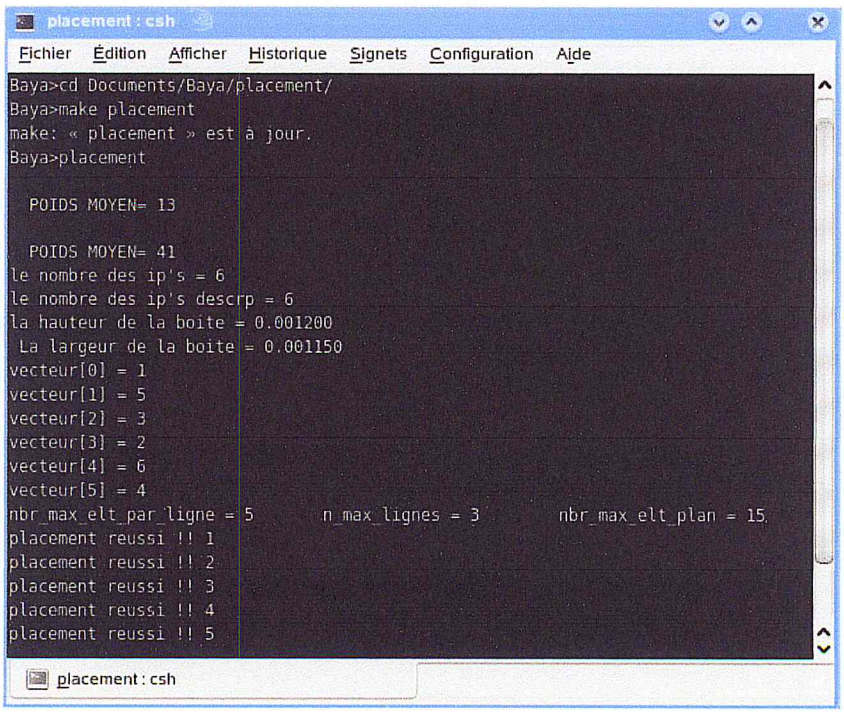



FIGURE 41 – *Résultat de placement des IPs de notre exemple.*

Nous obtiendrons finalement plusieurs configurations candidates qui serons enregistrées dans le fichier "FLOOR-PLANNING" (Figure 42). L'outil de génération du réseau sur puce sélectionnera, parmi ces solutions candidates, celle qui satisfait le mieux aux contraintes de largeur de bande, de surface et de consommation de puissance.

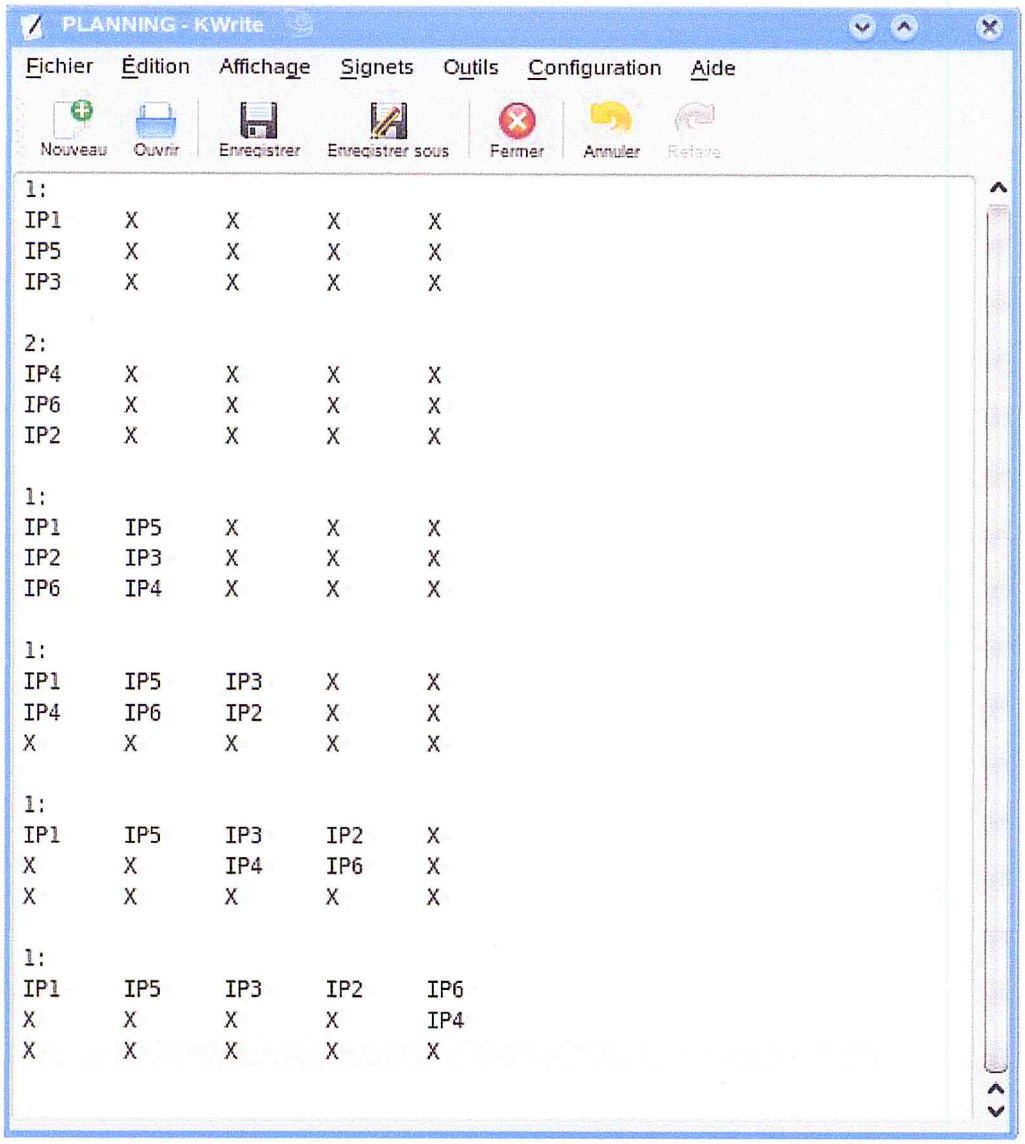



FIGURE 42 – *Fichier FLOOR-PLANNIG qui correspond à notre exemple.*

Le temps d'exécution de l'application est enregistré dans un fichier nommé T_ CPU illustré dans la (figure 43).

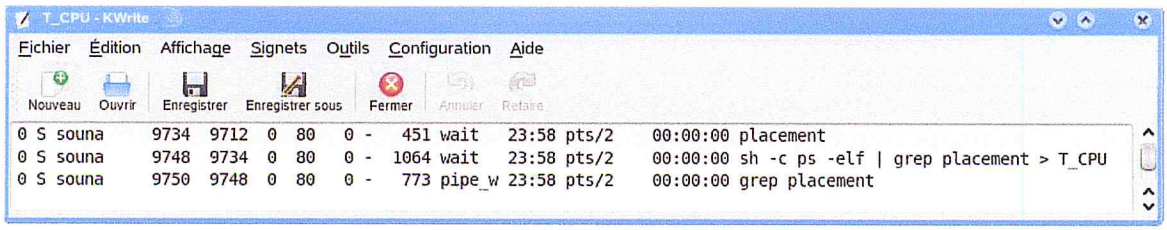



FIGURE 43 – *Temps d'exécution qui correspond à notre exemple.*

## 7.3 Résultats finaux

Le tableau suivant montre les caractéristiques d'un réseau sur puce 3D obtenus pour dix tests différents.

| Nbr IPs | Nbr Traces | Contraintes | | | Critères | | T-CPU Mon outil (ms) | T-CPU Outil Principal (s) |
|---|---|---|---|---|---|---|---|---|
| | | Larg-Bande (Gb/s) | Surface ($Cm^2$) | Puiss ($\mu w$) | Surface $IPs + Intc.$ ($Cm^2$) | $Puiss^*$ Intc. Dyn ($\mu w$) | | |
| 10 | 200 | 40 | 3.0 | 0.40 | 0.20 | 0.02 | 20 | 0 |
| 20 | 400 | 40 | 3.5 | 0.50 | 0.51 | 0.11 | 20 | 0 |
| 30 | 900 | 40 | 3.0 | 0.50 | 1.05 | 0.17 | 30 | 01 |
| 40 | 400 | 40 | 3.0 | 0.55 | 1.49 | 0.10 | 20 | 0 |
| 50 | 800 | 40 | 3.0 | 0.60 | 2.47 | 0.19 | 30 | 01 |
| 60 | 720 | 40 | 3.5 | 0.70 | 2.08 | 0.32 | 30 | 01 |
| 70 | 700 | 40 | 3.5 | 0.70 | 2.57 | 0.42 | 50 | 02 |
| 80 | 800 | 40 | 3.5 | 0.70 | 2.66 | 0.61 | 60 | 03 |
| 90 | 900 | 40 | 3.5 | 0.70 | 1.52 | 0.68 | 60 | 05 |
| 100 | 800 | 40 | 3.5 | 0.70 | 1.73 | 0.68 | 70 | 04 |

$*$ : Consommation dynamique des transferts de données SEULEMENT
(consommation des IPs et celle due aux capacités parallèles non comptabilisées).

TABLE 1 – *Tableau indiquant les résultats obtenus en terme de largeur de bande, de surface et de consommation de la puissance.*

Au niveau du SOFTWARE en cours de développement au CDTA, la largeur de la bande est celle fixée par l'utilisateur. Ainsi, le problème d'optimisation est plus relaxé, ce qui conduit à de meilleures valeurs de surface et de consommation de puissance que si on optimisait trois paramètres au lieu de deux seulement.

Comme on peut le constater à partir du tableau précédent, les trois contraintes sont toutes satisfaites.

Ces résultats ont été obtenus grâce à au software de génération de réseau sur puce en cours de réalisation au CDTA.

## 7.4 Conclusion

A travers les tests effectués, notre but était d'étudier les performances de la technique de placement proposée.

Après une introduction, la deuxième section a permis de présenter les résultats que nous avons obtenus lors de l'utilisation de l'outil qui a était le fruit de ce projet de fin d'étude en utilisant un exemple illustratif.

La troisième section, quant à elle, était consacrée pour démontrer l'efficacité de notre solution, en présentant un tableau illustratif de dix tests variés.

Nous remarquons que le temps CPU ne dépasse pas 05 s pour trouver une solution satisfaisant les trois contraintes, ce qui est intéressant pour un problème d'une telle difficulté. Notons aussi que ce temps CPU inclut le temps d'exécution de plusieurs tâches composant le software, dont notre algorithme de disposition et de placement d'IPs sur une architecture 3D.

# Conclusion générale et perspectives

Comme nous l'avions dit auparavant, le développement considérable de la technologie des semi-conducteurs permet d'intégrer de plus en plus de transistors sur la même puce. Néanmoins, cette forte intégration n'est pas sans problèmes et nécessite des méthodes de conception adéquates.

Parmi ces problèmes, se pose celui des interconnexions qui consiste à interconnecter de nombreux IPs présentant actuellement de fortes caractéristiques de communications. Ainsi, certaines architectures se trouvant limitées face à cette problématique, les architectures à base de réseaux sur puce se présentent comme une solution.

Dans le cadre d'un projet de conception de réseau sur puce mené au CDTA, nous avons contribué à sa réalisation par le développement d'une méthode de disposition d'IPs et de leur placement sur une architecture 3D.

Ceci nous a permis de mettre en pratique certaines de nos connaissances acquises durant notre formation universitaire.

Ce projet nous a été doublement bénéfique du fait que nous avons :

- amélioré nos connaissances en matière de conception de circuits et de systèmes intégrés.
- abordé une problématique ayant trait à un problème non polynomial et sa résolution moyennant une heuristique appropriée.

Pour finir, et avant de citer les perspectives du projet, nous pouvons dire que ce stage au niveau de CDTA nous a permis d'acquérir une très bonne expérience de recherche et nous a motivé pour compléter notre cursus universitaire.

Bien que nous avons obtenu des résultats satisfaisant, un projet de recherche n'est jamais complètement terminé, nous pouvons alors citer les perspectives et les développements suivants :

- Tester d'autres heuristiques afin d'obtenir de meilleurs résultats.
- Adapter notre méthode pour pouvoir résoudre d'autres problèmes non polynomiaux.
- Optimiser le nombre de contactes MIV aussi que de déterminer leurs emplacement en vue d'optimiser la largeur de bande, la surface et la consommation de puissance.

[14] Pankaj Gupta and Nick McKeown. Designing and implementing a fast crossbar scheduler. *Micro, IEEE*, 19(1) :20–28, 1999.

[15] A. Heatz. Les Métaheuristiques. *Ecole Polytechnique. Canada.*

[16] Michael Keating, David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. *Low power methodology manual : for system-on-chip design.* Springer Publishing Company, Incorporated, 2007.

[17] Cédric Koch-Hofer. *Modélisation, validation et présynthèse de circuits asynchrones en SystemC.* PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2009.

[18] Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johny Öberg, Kari Tiensyrjä, and Ahmed Hemani. A network on chip architecture and design methodology. In *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*, pages 105–112. IEEE, 2002.

[19] John H Lau. Critical issues of tsv and 3d ic integration. *Journal of microelectronics and electronic packaging*, 7(1) :35–43, 2010.

[20] Romain Lemaire. *Conception et modélisation d'un système de contrôle d'applications de télécommunication avec une architecture de réseau sur puce (NoC).* PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2006.

[21] A. Mahdoum. "A New Design Methodology of Networks on Chip" IEEE /ASQED'12 (Asian Symposium on Quality Electronic Design), July 10-11 2012, Penang, Malaysia.

[22] A. Mahdoum. Combined Heuristics for Synthesis of SOCs with Time and Energy Constraints. In *Computers and Electrical Engineering, ELSEVIER, Vol. 38, 2012, pp.1687-1702.*

[23] A. Mahdoum. Critique de Livre : "Book review : Representations for Genetic and Evolutionary Algorithms" written by Franz Rothlauf and edited by Springer Verlag Editions Review published in The Computer Journal, Oxford Journals, Vol. 49, No. 5, September 2006, ISSN : 0010-4620.

[24] A. Mahdoum. Démonstrations sur l'outil SAFT Forum de la Conférence Design, Automation and Test in Europe (DATE) 4-8 March 2002 Palais des Congrès Paris, France (Résumé de l'outil publié dans DESIGNER'S FORUM PROCEEDINGS DATE'02 4-8 March 2002 Paris, France).

[25] A. Mahdoum. SAFT : An Efficient State Assignment for Finite State Machine Tool. In *INFORMATION, Journal de l'Université de Hosei, Tokyo, Vol.3 N3 pp 379-395, July 2000, ISSN :1343-4500.*

[26] A. Mahdoum. SPOT : "An Estimation of the Switching Power Dissipation for CMOS Circuits and Data Paths Tool" SASIMI'97 1-2 Dec. 1997, Osaka, Japan.

[27] A. Mahdoum. SPOT : Un outil à base d'un algorithme génétique pour estimer la consommation de puissance dynamique des circuits CMOS. In *Conférence sur le Soft Computing et ses Applications CSCA99*, Alger, Algérie.

[28] A. Mahdoum. Démonstrations sur l'outil SPOT : Estimation de la consommation de puissance dynamique maximale et moyenne. In *Forum de la Conférence Design, Automation and Test in Europe (DATE)*. Palais des Congrès Paris, France (Résumé de l'outil publié dans DESIGNER'S FORUM PROCEEDINGS DATE'02 4-8 March 2002 Paris, France), 4-8 March 2002.

[29] A. Mahdoum, N. Badache, and H. Bessalah. An Efficient Assignment of Voltages and Optional Cycles for Maximizing Rewards in Real-Time Systems with Energy Constraints. In *Journal Of Low-Power Electronics (JOLPE)*, American Scientific Publisher, Vol. 2, No. 2, pp. 189-200, August 2006, ISSN : 1546-1998.

[30] A. Mahdoum, R. Benmadache, A. Chenouf, and M. L. Berrandjia. "an Efficient Low-Power Buffer Insertion with Time and Area Constraints" 14th ICC (International Conference on Circuits), 22-24 July 2010, Corfu, Greece.

[31] A. Mahdoum, R. Benmadache, and M. L. Chenouf, A. an dBerrandjia. A Low-Power Synthesis of Submicron Interconnects with Time and Area Constraints. International Journal of Circuits, Systems and Signal Processing, Issue 3, Vol.4, 2010, pp. 112-119, ISSN : 1998-4464.

[32] A Mahdoum and ML Berrandjia. FREEZER2 : Un outil à base d'un algorithme génétique pour une aide à la conception de circuits digitaux à faible consommation de puissance. In *IEEE/FTFC'07 (Faible Tension, Faible Consommation)*, pp. 143-148, Paris, France, May 21-23, 2007.

[33] A. Mahdoum, A. Boutamine, D. Touahri, and N. Toubaline. FREEZER1 : Un Outil d'Aide à la Conception de Circuits Digitaux à Faible Consommation de Puissance. pp. 65-70, Paris, France, May18-20 2005. IEEE/FTFC'05 (Faible Tension, Faible Consommation).

[34] A. Mahdoum, L. Hamimed, M. Louzri, and M. Saadaoui. Data Coding Methods for Low-Power Aided Design of Submicron Interconnects. Marrakech, Morocco, May30 - June 1 2011. IEEE /FTFC11 (Faible Tension Faible Consommation).

[35] A. Mahdoum, F. Louiz, and H. Belkouche. Fewercolors : A new technique for solving the graph coloring problem. In *7th IEEE/ICECS (International Conference on Electronics, Circuits and Systems)*, Beirut, Lebanon, 17-20 DEC 2000.

[36] A. Mahdoum, F. Louiz, and H. Belkouche. Démonstrations sur l'outil FEWERCOLORS. In *Forum de la Conférence Design, Automation and Test in Europe (DATE)*, Palais des Congrès Paris, France, 4-8 March 2002. (Résumé de l'outil publié dans DESIGNER'S FORUM PROCEEDINGS DATE'02 4-8 March 2002 Paris, France).

[37] Gordon Moore et al. Cramming more components onto integrated circuits, 1965.

[38] Nicolas Ngan. *Etude et conception d'un réseau sur puce dynamiquement adaptable pour la vision embarquée.* PhD thesis, Paris Est, 2011.

[39] Guy Pujolle. *Les réseaux : Edition 2014.* Editions Eyrolles, 2014.

[40] J. Rabaey. Busses and networking. *Computer Science*, 252, 2000.

[41] Nabil REKKAB Mohammed. Conception d'un simulateur de mapping dynamique sur une architecture hétérogene mp-soc basée noc. 2013.

[42] Séverine Riso, Gilles Sassatelli, Lionel Torres, Michel Robert, FG Moraes, et al. Réseau d'interconnexion pour les systèmes sur puce : le réseau hermes. *SCS'04 : Signaux, Circuits et Systèmes*, 2004.

[43] Maxime Rousseau. *Impact des technologies d'intégration 3D sur les performances des composants CMOS.* PhD thesis, Université Paul Sabatier-Toulouse III, 2009.

[44] James A Rowson and Alberto Sangiovanni-Vincentelli. Interface based design. In *Proceedings of the 34th annual Design Automation Conference*, pages 178–183. ACM, 1997.

[45] C. Gelatt S. Kirkpatrick and M. Vecchi. Optimization by Simulated Annealing. *Science, Vol. 220, No. 4598, pp. 671-680*, May 1983.

[46] Nabil Sadou. *Aide à la conception des systèmes embarqués sûrs de fonctionnement.* PhD thesis, INSA de Toulouse, 2007.

[47] Chawki Sahnine. *Architecture de circuit intégré reconfigurable, très haut débit et basse consommation pour le traitement numérique de l'OFDM avancé.* PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2009.

[48] Y. Soler. Planification et suivi d'un projet. *Centre National de la Kamusal Kalkınmada Proje Yönetimine Yönelik Modelsel Bir Yaklaşım*, 2001.

[49] C Tanougast, S Jovanovic, F Monteiro, C Diou, and A Dandache. Initiation à la modélisation et co-simulation comportementale c-vhdl d'un réseau de communication sur puce (network on chip).

[50] SK. Tewksbury, M. Uppuluri, and LA. Hornak. Interconnections/micronetworks for integrated microelectronics. In *Global Telecommunications Conference, 1992. Conference Record., GLOBECOM'92. Communication for Global Users., IEEE*, pages 180–186. IEEE, 1992.

[51] Marie-Emilie Voge. Graphes colorés-arbre couvrant coloré. *Huitiemes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel'06)*, pages 41–44, 2006.

[52] Mounir Zid, Abdelkrim Zitouni, Adel Baganne, and Rached Tourki. Nouvelles architectures generiques de noc. *TSI. Technique et science informatiques*, 28(1) :101–133, 2009.