Table des matières

Chapitre 2 : Les applications de calcul intensif

Chapitre 3 : Optimisation Multi- objectifs

Table des Figures

Liste des Tableaux

Introduction Générale

Introduction générale

1. CONTEXTE

Les semi-conducteurs constituent les briques élémentaires des industries modernes. Les technologies qui leur sont associées jouent un rôle essentiel dans tous les secteurs. Leur perpétuelle évolution a donné naissance à des systèmes composés de millions de transistors sur une seule et même puce. C'est ce que l'on appelle les Systèmes sur Puce ou SoC (pour « System On Chip »). Ces systèmes intègrent différents éléments comme des processeurs, des mémoires, des blocs d'entrée/sortie ou encore des médias de communications. Les progrès de la technologie et l'évolution des besoins font que la durée de vie de ces architectures diminue. Afin d'y remédier, de plus en plus de fonctionnalités sont introduites dans un même système, ce qui conduit à la mise en communication d'un grand nombre de blocs fonctionnels de natures hétérogènes. Cependant, les liens de communication n'évoluent pas à la même vitesse et deviennent un goulot d'étranglement.

Les solutions de communication actuelles telles que le bus partagé trouvent leurs limites en termes de bandes passantes et d'extension à mesure que le nombre d'éléments communicants augmente. Cette structure est la plus utilisée de nos jours mais ne semble pas s'adapter aux applications futures. C'est dans ce contexte que le concept des Réseaux sur Puce ou NoC (Networks On Chip) a vu le jour.

Ce paradigme d'interconnexion, inspiré des réseaux informatiques classiques, offre une structure de communication évolutive, flexible et propose des solutions efficaces aux problèmes d'intégrations complexes des systèmes sur puce. Cependant, il n'existe pas d'outils d'automatisation de l'ensemble des phases de conception. C'est pourquoi ils constituent l'un des axes les plus actifs de la recherche.

2. PROBLEMATIQUE ET MOTIVATION

La phase de mapping (placement physique) est une phase centrale dans la 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 du réseau sur puce de sorte à minimiser le coût de communications, la consommation d'énergie ou la latence du système et respecter les contraintes telles que la bande passante ou le temps réel.

Ce mapping a une particularité, dans ce contexte, car il ne s'agit pas uniquement d'un placement classique qu'on a l'habitude de rencontrer dans les systèmes parallèles mais il englobe trois phases: Assignation, Affectation et Scheduling (AAS) et que L'on peut caractériser par :

- Le placement des calculs (taches) qui permet de trouver sur quel processeur doit s'exécuter quelle tâche et à quel moment.

- Le placement de données qui doit déterminer un accès efficace et rapide aux données avec si c'est possible des chemins d'accès le plus court possible en prenant en considération les caractéristiques de la topologie cible.

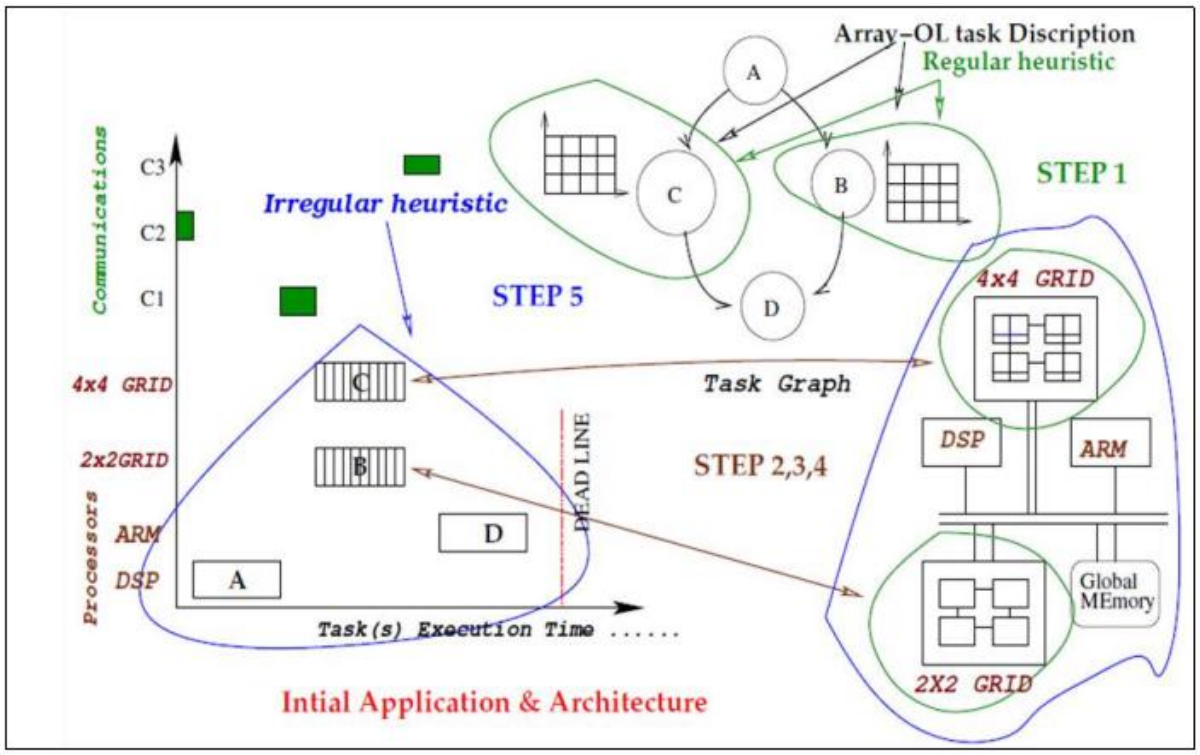


Figure 1 - Concept GILR : Globalement Irrégulier Localement Régulier [5]

Ce type de placement ou mapping dans sa globalité est connu comme étant un placement GILR : Globally Irregular Locally Regular (cf. Figure 1), il consiste à exploiter le parallélisme de tâches et de données dans une architecture multiprocesseurs [57,58,59].
Donc notre travail consiste à étudier tous les types des tâches, on ne peut pas faire un bon placement si on ne prend pas en considération tous les aspects où on a des tâches qui sont répétitives et d'autres qui ne le sont pas. Ce qui doit nous amener à extraire les parties régulières et irrégulières de l'architecture pour exploiter au maximum la correspondance dans l'application. Le graphe ci-dessus illustre un GILR où les couleurs représentent les étapes suivies.
En généralisant cet exemple par une modélisation graphique on obtiendra des graphes Hiérarchiques. Et là on parlera d'un placement hiérarchique multi-niveaux, où les nœuds peuvent être de trois types : élémentaire, composé ou répétitif. Donc un placement GILR nous amène à trouver le meilleur mapping d'un graphe hiérarchique de l'application sur le graphe hiérarchique de l'architecture.

3. CONTRIBUTION

Afin de traiter le problème dans sa globalité on l'a décomposé en sous problèmes complémentaires :

1. Pour le mapping de tâches irrégulières sur l'architecture correspondante on a proposé un algorithme évolutionniste hybridé. Ce dernier cherche l'optimisation multi-objectif du placement d'un graphe hiérarchique d'application sur un graphe hiérarchique d'architecture. En effet, pour prendre en considération les spécificités et les exigences des SoCs(NoC), on ne cherche pas la meilleure solution selon un seul objectif (temps, consommation d'énergie, taille mémoire,..) mais plusieurs. Dans notre cas on a fait du bi-objectif (temps et énergie), car ils sont les plus importants. Cependant, le coût global du placement ne dépend pas uniquement des coûts des unités de traitements, il dépend aussi des coûts des communications. D'où une autre méthode permettant de chercher le meilleur mapping des communications sur la topologie cible, ce qui fait de l'algorithme, proposé pour ce sous problème, un algorithme hybride d'optimisation multi-objectif.

Notre travail consiste à étudier certaines des principales techniques de mapping existantes et proposer une nouvelle solution qui permet de mapper une application sur architecture MPSoC tout en minimisant le coût de communications.

2. Placer toutes les tâches répétitives sur les ressources d'architecture en même temps va accroitre absolument la complexité du traitement du processus AAS. Pour cette raison et vu de la spécificité des tâches à placer, il sera bénéfique de le faire par partie en déterminant un paquet de taches (Motif) qu'on place en minimisant le temps d'exécution et la consommation d'énergie et en respectant les contraintes exigées. Ensuite c'est le même nombre de tâche qui est répété sur d'autres données pour générer le placement total (AAS) de toutes les tâches. Donc pour ce problème un algorithme hybride exact est proposé pour solutionner ce genre de problème et trouver le meilleur placement d'une application sur une architecture régulière.

4. PLAN DE DOCUMENT

Le document est organisé en 2 parties selon un ordre qui permettra, à notre avis, au lecteur de suivre et de comprendre la démarche :

La première partie est une synthèse bibliographique constituée de trois chapitres :

- Le premier chapitre traite du contexte de l'apparition des réseaux sur puce, Nous commençons par définir ces systèmes en citant leurs caractéristiques et leurs architectures et en insistant sur les méthodologies et les outils de leur conception.
- Le second chapitre introduit les notions des applications de traitement intensif. Nous donnons un aperçu général sur les principaux modèles de calcul existants pour leur spécification, et nous décrivons en particulier le modèle ARRAY-OL qui est principalement conçu pour la description des applications de traitement du signal qui nous intéressent.
- Le troisième chapitre est consacré à l'optimisation combinatoire multi objectifs. Les concepts qui y sont introduits sont nécessaires pour comprendre notre approche de résolution.

La deuxième partie est divisée en deux chapitres et présente l'aspect pratique du projet.

- Le quatrième chapitre présente une proposition pour la résolution du problème de mapping , il s'intéresse à l'approche de résolution adoptée et détaille les méthodes ainsi que les formalismes qui les entourent.
- le cinquième chapitre sera dédié à la conception et la mise en œuvre de notre approche. En plus il sera consacré aux résultats des expérimentations générées par notre application ainsi que la démonstration de l'efficacité de notre approche proposée.

Enfin, ce document sera clôturé par une conclusion qui résume l'apport essentiel de notre approche proposée. Elle ouvre également de nouveaux éléments de réflexion et quelques perspectives de recherche.

Chapitre I

« Introduction aux Systèmes Embarqués et flot de Conception »

1.1 INTRODUCTION

L'évolution des technologies informatiques au cours des dernières années a entraîné des modifications radicales dans la conception des applications. Les nouveaux réseaux de télécommunication rapides et à large bande passante (RNIS, ATM, High Performance Ethernet) ont permis le regroupement des ordinateurs en réseaux, d'abord de taille réduite (entreprise), puis moyenne (nationale) puis finalement mondiale avec Internet [1].

Tout ceci a favorisé la démocratisation des communications informatiques (à commencer par les messageries électroniques et la navigation sur le Web) puis, plus récemment, l'émergence des applications distribuées. Chaque jour, notre vie courante devient plus orientée vers les réseaux. Plus récemment on trouve dans la majorité des appareils mobiles des réseaux intégrés dans des puces, qui appartiennent à la famille des systèmes embarqués qui est elle-même issue des systèmes distribués.

C'est en 1971 que débute réellement l'histoire des circuits intégrés. Cette année-là, Intel met au point le premier microprocesseur, le 4004, comportant 2300 transistors et gravé dans une technologie PMOS, fonctionnant à une fréquence de 108 kHz. En 2004, On estime le nombre de circuits intégrés en utilisation à 25 milliards dans le monde. A titre d'exemple, le Pentium IV "Extrême Edition" contient 125 millions de transistors en technologie CMOS [2], pour des fréquences de fonctionnement au-delà de 3 GHz. Ces quelques chiffres illustrent à eux seuls la gigantesque dynamique de recherche et de production qui s'est mise en place en à peine plus de trente ans [1].

Qui aurait pu imaginer cela en 1971 ? Eh bien, un homme, Gordon Moore, cofondateur d'Intel en 1968, avait prédit que les capacités d'intégration sur les technologies silicium doubleraient tous les 2 ans (la loi de Moore).

Dans cette partie nous définissons : Qu'est-ce qu'un système embarqué ? Puis Ses caractéristiques, son architecture, ses domaines d'application et ses contraintes. Puis nous situons leur évolution et les questions auxquelles ils sont confrontés. Ensuite nous développons les spécificités des Socs et leur Interconnexion du fait que les NoC's constituent une nouvelle approche visant à intégrer un réseau sur une plateforme SoC.

1.2 DEFINITION DES SYSTEMES EMBARQUÉS

Un system embarqué [3] peut être définit comme un system électronique et informatique autonome ne possédant pas d'entrées/sorties standards comme un clavier ou un écran. On peut avoir un ou plusieurs systèmes embarqués sur puces (SOC, SYSTEM ON CHIP), de natures hétérogènes et donc complexes. Ils sont souvent constitués de plusieurs processeurs de types différents (FPGA, DSP, GP par exemple), avec des fonctions dédiées ou reconfigurables. Ils captent des informations de leur environnement, s'adaptent et agissent sur lui sans presque aucune intervention humaine.

La réalisation de ces systèmes réunit de nombreuses compétences : informatique, électronique, architecture de systèmes et mathématiques. Elle nécessite à prendre fondamentalement en compte l'hétérogénéité des composants et des logiciels [4].

1.3 CARACTERISTIQUE DES SYSTEMES EMBARQUÉS

On énumère les caractéristiques principales suivantes :

- C'est un système principalement numérique.
- Il met en œuvre généralement un ou plusieurs processeurs.
- Il exécute une application logicielle dédiée pour réaliser une fonctionnalité précise et n'exécute donc pas une application scientifique ou grand public traditionnelle.
- Il n'a pas réellement de clavier standard (bouton poussoir, clavier matriciel...).
- Ce n'est pas un PC en général mais des architectures similaires à basse consommation d'énergie qui sont de plus en plus utilisées pour certaines applications embarquées.

De ce fait on constate que :

- Les ressources (énergie, surface) sont limitées.
- L'interface IHM peut être aussi simple qu'une LED qui clignote ou aussi complexe qu'un cockpit d'avion en ligne.
- Des circuits numériques ou des circuits analogiques sont utilisés en plus des processeurs pour augmenter les performances du système embarqué ou sa fiabilité [5].

Les systèmes embarqués se caractérisent aussi, plus particulièrement par l'impératif de leur sûreté de fonctionnement. Les dommages au système peuvent être provoqués par le non-respect des contraintes temps. Par exemple, la pression du bouton d'alarme d'un appareil devra nécessairement s'accompagner d'une action immédiate et prioritaire comme par exemple :

activation de la sonnerie, coupure de l'alimentation électrique et déclenchement des opérations de sauvetage. Autre exemple : la mesure d'une différence de valeur d'état par rapport à une valeur de consigne (pression, température) devra entraîner une opération propre à ramener le système vers son état normal ou état de référence en vue d'éviter des risques qui peuvent entraîner des pertes humaines et matérielles. C'est notamment le cas dans les applications de contrôle (comme dans les processus industriels, dans la commande de réacteurs chimiques ou nucléaires, dans des applications aux systèmes de transport commandes de vols, système de freinage du genre ABS, conduite de locomotives métros ou dans la navigation aérienne, maritime etc...). Ce sont des systèmes temps réel dits aussi critiques. Afin d'assurer la sûreté de fonctionnement d'applications informatiques embarquées, des normes ont été établies. Ils doivent être validés et certifiés pour les applications envisagées. Il en existe pour la plupart des domaines d'applications critiques, tels que : avionique, espace, nucléaire, défense, ferroviaire, industrie, médical, automobile etc...

1.4 DOMAINES D'APPLICATIONS DES SYSTEMES EMBARQUÉS

Les domaines dans lesquels on trouve des systèmes embarqués sont de plus en plus nombreux :

-Transport : Automobile, Aéronautique (avionique) etc.

- Astronautique : fusée, satellite artificiel, sonde spatial, etc.

- Militaire : missiles.

- Télécommunication : Set-top box, téléphonie, routeur, pare-feu, serveur de temps, téléphone -portable etc.

-Electroménager : télévision, four à micro-ondes.

- Impression : imprimante multifonctions, photocopieur etc.

- Informatique : disque dur, lecteur de disquette etc.

- Multimédia : console de jeux vidéo, assistant personnel.

- Guichet automatique bancaire (GAB).

- Equipement médical.

-Automate programmable industriel, contrôle-commande.

- jeux : consoles.

- Métrologie.

1.5 UN SYSTEM EMBARQUE TYPIQUE

La figure ci-dessous nous présente un modèle standard et typique d'un système embarqué : On trouve en entrée des capteurs généralement analogiques couplés à des convertisseurs A/N. On trouve en sortie des actionneurs généralement analogiques couplés à des convertisseurs N/A.

Au milieu : le calculateur mettant en œuvre un processeur embarqué et ses périphériques d'E/S. Il est à noter qu'il est complété généralement par un circuit FPGA jouant le rôle de coprocesseur afin de proposer des accélérations matérielles à celui-ci.

On trouve en fait un système d'asservissement entre les entrées et les sorties. Notons que l'expression la plus simple de cette figure est de considérer comme capteurs, des interrupteurs et comme actionneurs, des leds. Sur ce schéma théorique (cf. figure 1.1) se greffe un paramètre important : le rôle de l'environnement extérieur. Contrairement au PC dans un bureau, un système embarqué doit faire face à des environnements plus hostiles. Il doit faire face à un ensemble de paramètres agressifs :

- Variations de la température.
- Vibrations, chocs.
- Variations des alimentations.
- Interférences RF (Radio Fréquence).
- Corrosion.
- Eau, feu, radiations.

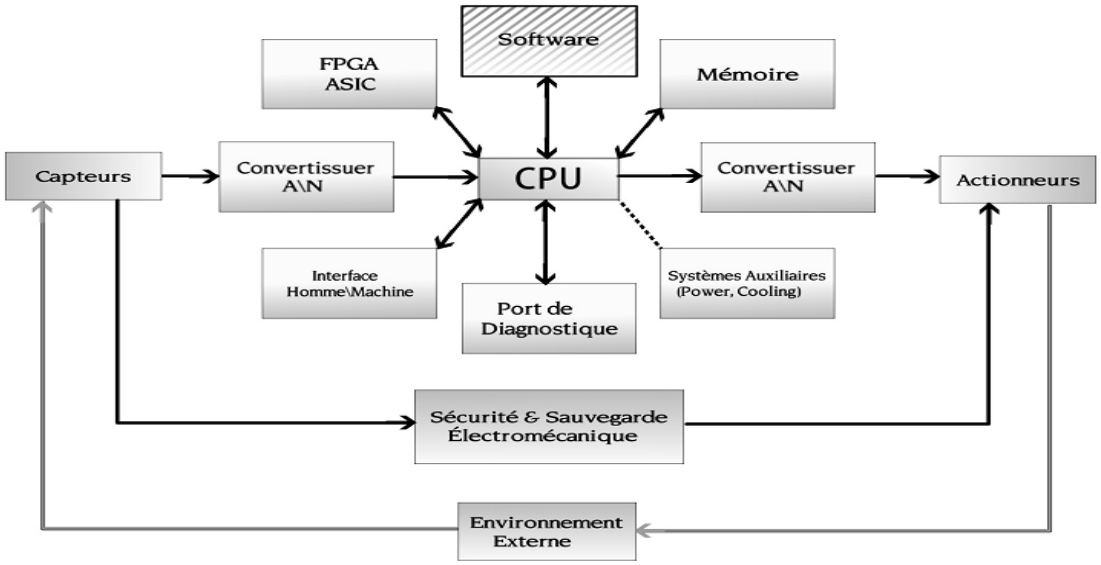


Figure 1.1 – Schéma typique d'un système embarqué [13]

1.6 ARCHITECURE DES SYSTEMES EMBARQUÉS

Les systèmes à base de puces électroniques font de plus en plus partie de notre quotidien. Du simple transistor au circuit intégré, ces structures intègrent aujourd'hui des milliers de composants complexes et de fonctionnalités hétérogènes.

L'augmentation constante du nombre de fonctions intégrées dans un seul système a fortement augmenté les contraintes et les exigences d'architecture en terme de composants et les réseaux de communication. La figure 1.2 simplifie la vue des tendances de conception adoptée :

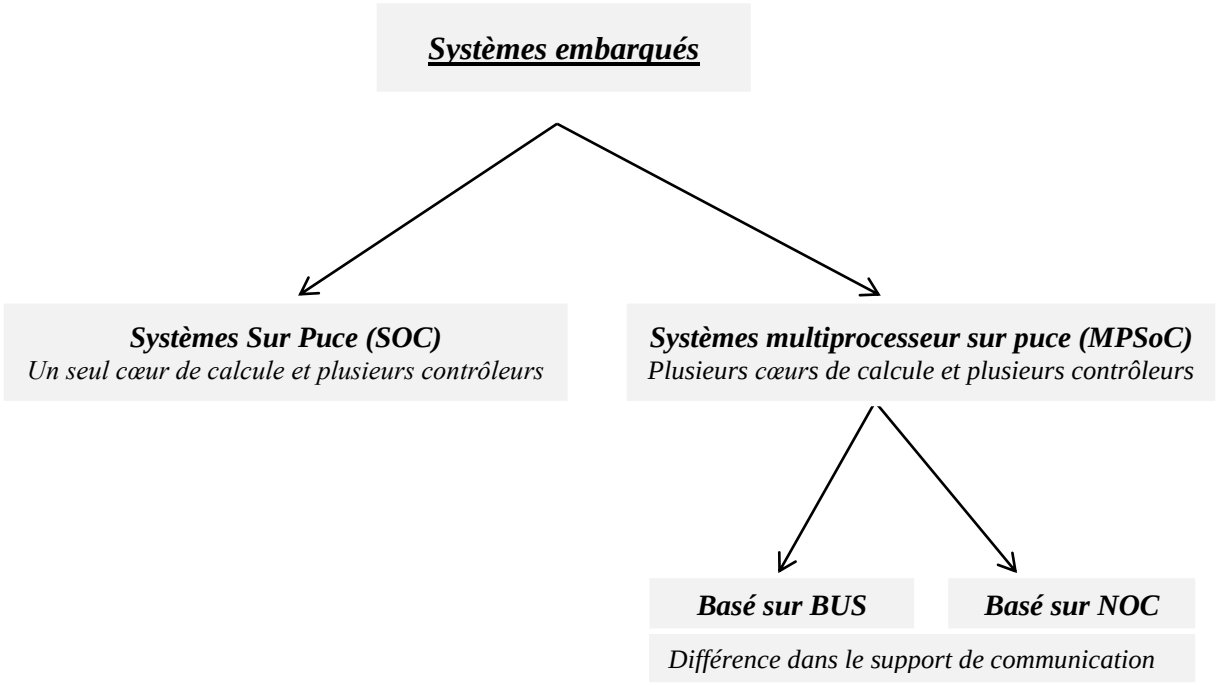


Figure 1.2 - Les tendances de conception actuelle et future d'un système embarqué [11]

1.6.1 Système Mono Puce (SOC)

L'expression SoC (System On Chip) est utilisée pour désigner un système complet embarqué sur une puce. Il peut comprendre de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interface, ou tout autre composant nécessaire à la réalisation de la ou les fonctions attendues. On peut intégrer de la logique, des dispositifs (capteurs) mécaniques, optoélectroniques, chimiques ou biologiques ou des circuits radio [12].

De nos jours, nous assistons à un immense progrès de ces systèmes. Ils sont de plus en plus

utilisés dans les ordinateurs, téléphones portables ou encore consoles de jeux. Cette évolution a donné naissance à des architectures de plus en plus complexes exécutant plusieurs fonctionnalités à la fois.

Dans la figure 1.3 est présenté un exemple de système mono-puce typique. Il se compose d'un Cœur de processeur (CPU), d'un processeur de signal numérique (DSP), de la mémoire embarquée, et de quelques périphériques tels qu'un DMA (Direct Memory Access) et un contrôleur d'E/S.

Figure 1.3 – Schéma typique d'un SOC [13]

1.6.2 Système Multiprocesseurs sur puce (MPSoC)

Les systèmes sur puce multiprocesseurs ou *MPSoC (Multi-processeur System-on-Chip),* sont apparus grâce à l'évolution de la technologie de fabrication des circuits intégrés qui a permis l'intégration de plusieurs composants sur une seule puce. Ces systèmes ont rapidement pris une place importante dans le domaine de la micro-électronique car ils intègrent dans un même circuit plusieurs processeurs, des nombreux composants numériques spécialisés et hétérogènes (mémoires, périphériques, unités de calcul spécifiques), du logiciel et souvent des circuits mixtes pour fournir un système intégré complet.

Les systèmes mono-puce multiprocesseurs sont généralement dédiés aux applications spécifiques dans des domaines différents comme par exemple dans le domaine de l'automobile, des télécommunications et du multimédia et leur diversité ne cesse d'accroître. La figure 1.4 montre l'architecture d'un tel système mono-puce multi-processeur.

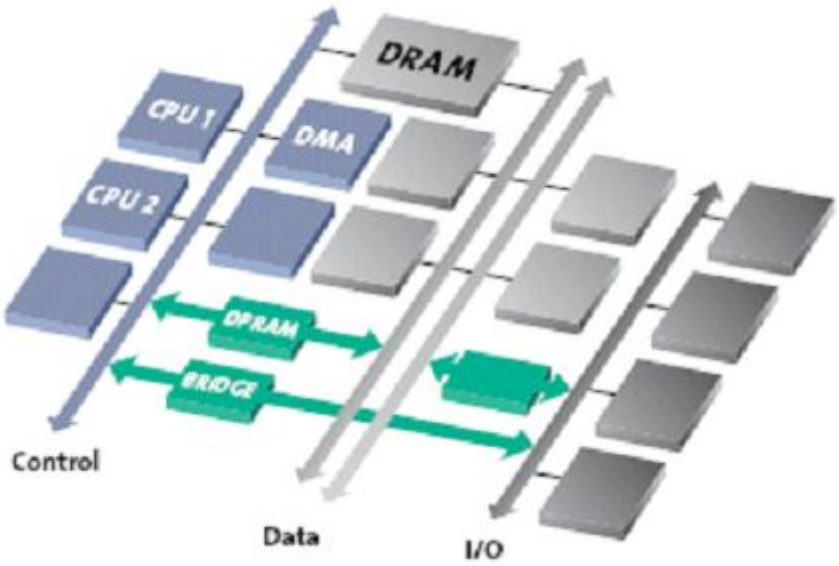


Figure 1.4- L'architecture des MPSoC's [15]

1.6.3 Network on Chip (NOC)

Du succès du *SoC (System-on-Chip)* est né le *NoC (Network-on-Chip)*. Le premier de ces concepts consiste à placer au sein d'une même puce plusieurs fonctions jusque-là gérées par des puces dédiées et regroupées sur une carte. Créée en début 2003, la start-up française Arteris propose la notion de réseau à l'intérieur de la puce, qu'elle appelle NoC.

Dans le *NoC*, les informations transitent entre les nœuds à chaque tic horloge, on peut donc prévoir exactement où sera le paquet à transmettre à chaque instant. L'envoi des données est géré par un *TDMA (Time Division Multiple Access)*.

Une plateforme de NOC est une structure de maille composée de passages avec chaque commutateur connecté à une ressource, comme il est montré sur la figure 1.4. Les ressources sont placées sur les fentes constituées par les commutateurs. Le commutateur de réseau offre une communication pour des ressources. Les ressources exécutent leurs propres fonctionnalités informatiques et fournissent le *Ressource-Réseau-Interface (RNI)* [7].

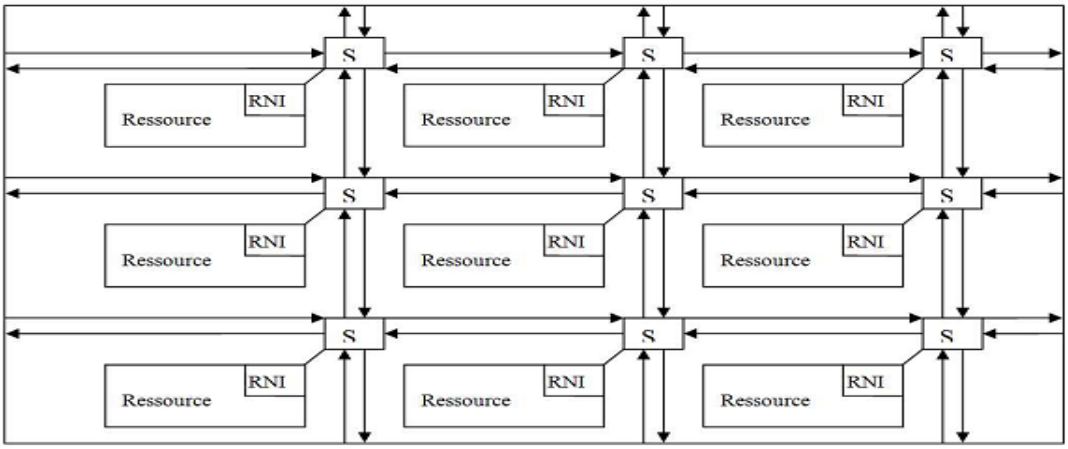


Figure 1.5- L'architecture des NoC's [6]

1.7 INGENERIE DIRIGEE PAR LES MODELES (IDM)

La compréhension d'un système complexe passe par une représentation abstraite et simplifiée de ce système : une modélisation. Un modèle met en évidence certaines caractéristiques du système en Faisant abstraction notamment de ses détails d'implémentation. Plusieurs méthodes et langages ont permis par le passé de manipuler les modèles : essentiellement : *UML (Unified Modeling Language)* [8], etc.

Actuellement, différentes approches permettent la modélisation d'un système, et cela en fonction du domaine d'utilisation. On retrouve parmi elles l'Ingénierie Dirigée par les Modèles *(IDM ou MDE pour Model Drivent Engineering)* [9] qui est plus particulièrement utilisée pour la conception de modèles productifs, c'est-à-dire compréhensibles et interprétables par les machines [10]. De cette façon, l'IDM se démarque d'autres méthodologies de modélisation qui ont vocation à concevoir des modèles contemplatifs, c'est-à-dire non interprétables par les machines.

Nous présentons l'aspect méthodologique lié à l'IDM et faisons référence à certains aspects technologiques et aux outils. Les trois concepts de base qui permettent de caractériser la méthodologie liée à l'IDM sont les modèles, le méta modèle et les différents niveaux d'abstraction.

1.7.1 Modèles :

Un modèle correspond à une abstraction de la réalité et en permet une représentation à l'aide de concepts et de relations entre ces concepts. L'IDM utilise les modèles tout au long du

développement logiciel. Cette notion de modèle était déjà présente en informatique dans les années 1970.

Les modèles ne sont donc pas nouveaux mais reflètent une manière de penser la résolution de certains problèmes. C'est un mouvement de la pensée pour conceptualiser les problèmes et concevoir des solutions. l'IDM reprend cette manière de penser et l'applique à l'informatique.

1.7.2 Méta-modèles :

L'IDM recommande l'utilisation des modèles à différents niveaux d'abstraction. Un modèle représente une vue abstraite de la réalité et est conforme à un méta modèle qui définit précisément les concepts présents à ce niveau d'abstraction ainsi que les relations entre ces concepts. Un méta modèle permet donc de représenter des mécanismes complexes faisant intervenir plusieurs concepts.

1.7.3 Niveaux M0, M1, M2 et M3 :

Le niveau M0 correspond à la réalité. Le niveau M1 correspond à une première abstraction de la réalité, c'est un modèle. Le niveau M2 correspond à une abstraction du modèle, c'est un méta modèle.

Un méta modèle est aussi un modèle, il est par conséquent conforme à un autre méta-méta modèle (niveau M3) qui est lui-même conforme à un autre méta modèle, etc. En pratique, afin d'éviter une infinité de niveaux, le méta modèle du niveau M3 peut s'auto définir. La figure 1.6 illustre les relations entre réalité (M0), modèles (M1), méta modèles (M2) et méta-méta modèles (M3) :

-M0 représente la réalité (un programme informatique). Sur la figure 1.6, les variables Numéro et Solde sont affectées par des valeurs.

- M1 correspond au premier niveau d'abstraction, c'est un modèle et c'est lui que manipule un développeur. Pour cet exemple, le modèle contient la déclaration des variables utilisées dans M0 et la notion de Compte. Un modèle de niveau M1 est conforme à un méta modèle de niveau M2.

- M2 est le niveau dans lequel sont dénis les concepts manipulés dans un modèle de niveau M1 (c'est-à-dire les concepts manipulés par un développeur *« UML »*). Dans cet exemple, Compte est une classe tandis que les déclarations de variables sont des Attributs de cette Classe. Classe et Attributs sont tous deux des concepts du méta modèle de niveau M2, qui est conforme à un méta modèle de niveau M3.

-M3 correspond au plus bas niveau d'abstraction, un méta-méta modèle de niveau M3 est conforme à lui-même. Dans cet exemple, les concepts Classe et Attributs sont des Méta-Classes, tandis que la notion de contenance est une Méta-relation. Ce méta-méta modèle se décrit lui-même, la Méta-Classe et la méta-Relation sont des Méta-Classes et les relations source et destination qui les lient sont des méta-Relations.

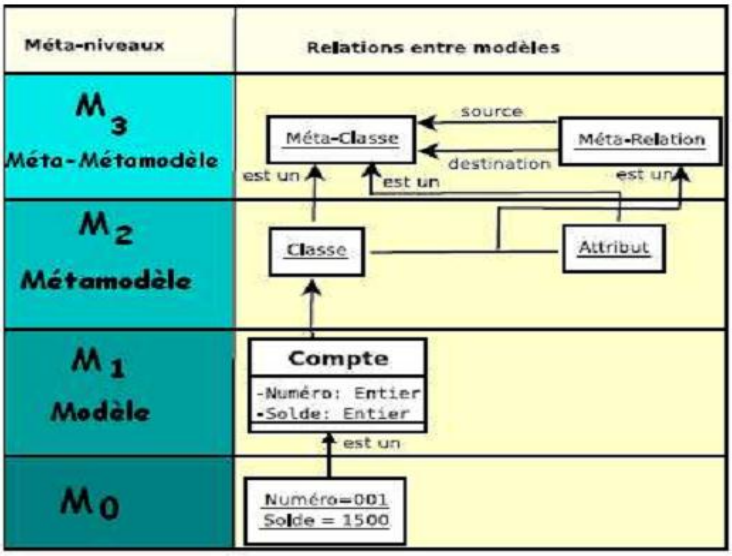


Figure 1.6 - Les différents niveaux de modélisation [9]

1.8 LE PROFILE MARTE « EXTENSIBILTE D'UML »

MARTE (Modeling and Analysis of Real - time and Embedded systems) [16,17] est un profil standard d'UML proposé par *l'OMG (Object Management Group)*, visant principalement à ajouter à *UML* des capacités et des concepts pour le développement dirigé par modèles des systèmes embarqués temps réel.

Comme on peut le voir dans la figure 1.7, ce profil s'organise en différents paquetages fournissant un ensemble d'éléments de notations facilitant l'utilisation d'*UML* pour la modélisation et l'analyse des systèmes temps réels embarqués.

L'architecture *MARTE* est fondée sur quatre paquetages : le paquetage de base, le modèle de conception, le modèle d'analyse et les annexes :

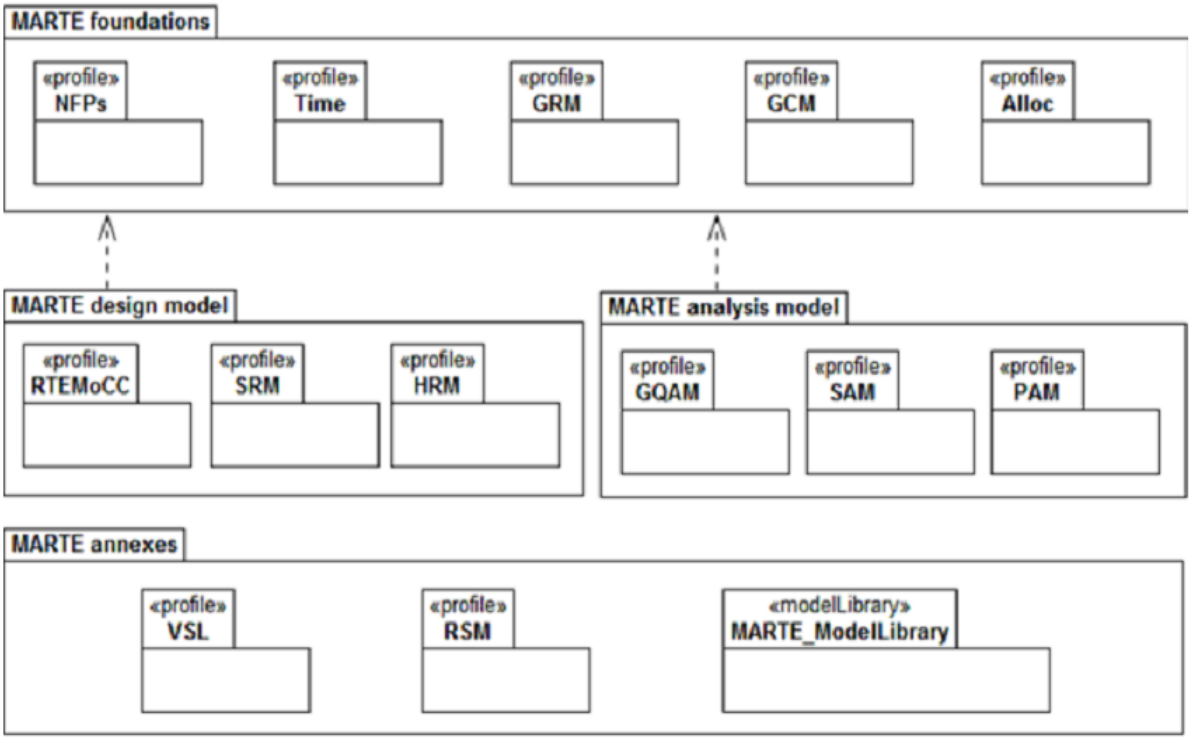


Figure 1 .7- Architecture globale du profil MARTE [18]

Le paquetage de base : comprend le profil des *NFPs* pour la modélisation des propriétés non fonctionnelles; le profil *TIME* pour la modélisation du temps logique et physique; le profil *GRM* pour la modélisation générique de ressources.

Le *GRM* pour la modélisation détaillée de ressources; le profil *GCM* pour la modélisation de composants génériques et le profil *ALLOC* pour modéliser l'association matériel/logiciel.

Le paquetage de modèle de conception : représente le profil de base, il englobe le profil *RTEMOCC* pour les modèles de calcul et de communication temps réels; le profil SRM pour la modélisation de ressources logiciel et le profil HRM pour la modélisation des ressources matériel.

Le paquetage d'analyse de MARTE: pour prendre en compte les trois types d'analyse: La SAM pour l'analyse d'ordonnançabilité , la *PAM* pour l'analyse des performances et *GQAM* pour l'analyse générique.

Le paquetage d'annexes : inclut en particulier le sous - profil *VSL (Value Specification Language)* qui est un langage d'expression, utilisé pour spécifier les valeurs non-fonctionnelles; le sous profil RSM pour modéliser les structures répétitives ainsi bien que des librairies prédéfinies.

1.9 ENVIRONNEMENT DE CO-CONCEPTION POUR SOC « GASPARD »

GASPARD (Graphical Array Specification for Parallel and Distributed Computing) est un environnement qui propose une méthodologie de conception conjointe pour les systèmes intégrés sur puce ou *SoC*, basée sur l'approche *MDE (Model Driven Engineering)* comme illustré par la figure1.8 Il propose un profil *UML* intégré au profil MARTE de *l'OMG (object management group),*en cours d'adoption et dédié aux systèmes embarqués et temps réel, qui permet à des utilisateurs de modéliser à la fois des applications de traitement de données intensives et leurs architectures. Un mécanisme d'association est fourni pour les deux aspects, ainsi qu'un ensemble de transformations pour la simulation et la synthèse. Notre approche de modélisation vise à tenir compte de toutes ces caractéristiques de *GASPARD* et à pouvoir garantir la correction des modèles manipulés au sein de sa méthodologie de conception.

Les modèles *GASPARD* sont spécifiés à l'aide du langage *ARRAY-OL (Array Oriented Language).*Ce dernier permet de modéliser des applications manipulant des quantités importantes de données. Il adopte des représentations multidimensionnelles permettant d'exprimer tout le parallélisme potentiel présent dans des applications cibles. Les données sont structurées dans des tableaux aux dimensions pouvant être infinies. Une tâche *ARRAY-OL* consomme et produit des tableaux par morceaux de taille constante, appelés les motifs. Des tâches différentes sont reliées entre elles par des dépendances de données.

Lorsqu'une dépendance est spécifiée entre deux tâches, cela signifie que l'une d'entre elles requiert des données de la part de l'autre avant de s'exécuter. Ces dépendances exhibent donc initialement un ordre partiel minimal d'exécution. Les applications peuvent être composées hiérarchiquement à différents niveaux de spécifications. En pratique, leurs spécifications consistent en un modèle global et en un modèle local.

1.9.1 Le modèle globale :

Le modèle globale d'une application est représenté par un graphe acyclique dirigé où les nœuds symbolisent les tâches et les arcs transportent des données à travers des tableaux multidimensionnels. Il n'y a aucune restriction quant au nombre de tableaux d'entrée et de sortie. Les tableaux sont supposés toriques, c'est-à-dire leurs éléments peuvent être consommés ou produits modulo la taille des tableaux. Si le modèle global fournit des informations permettant d'ordonnancer l'exécution des tâches, il n'exprime pas le parallélisme de données présent au sein de ces tâches. Ce dernier aspect est décrit par le modèle local.

1.9.2 Le modèle local :

Le modèle local décrit le parallélisme de données exprimé à travers des répétitions. L'ensemble de l'environnement Gaspard est construit dans une démarche IDM (Ingénierie Dirigée par les Modèles), de la modélisation en UML jusqu'à la génération de code.

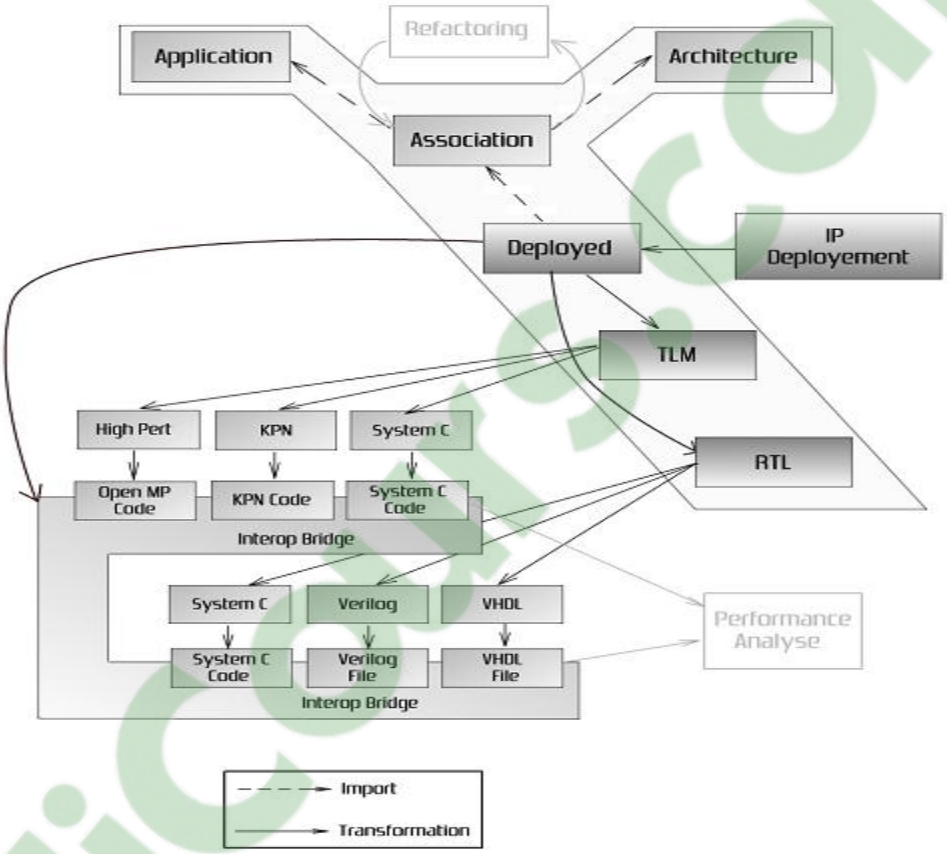


Figure 1.8 -Conception de systèmes intégrés sur puce selon Gaspard [13]

La figure 1.9 illustre un exemple d'unité de calcul *« Processing Unit »* définit à l'aide de trois composants élémentaires : un processeur *MIPS*, une mémoire ***Memory*** et un *crossbar* ***Crossbar4***. Ce modèle est généré par l'environnement de ***Gaspard***.

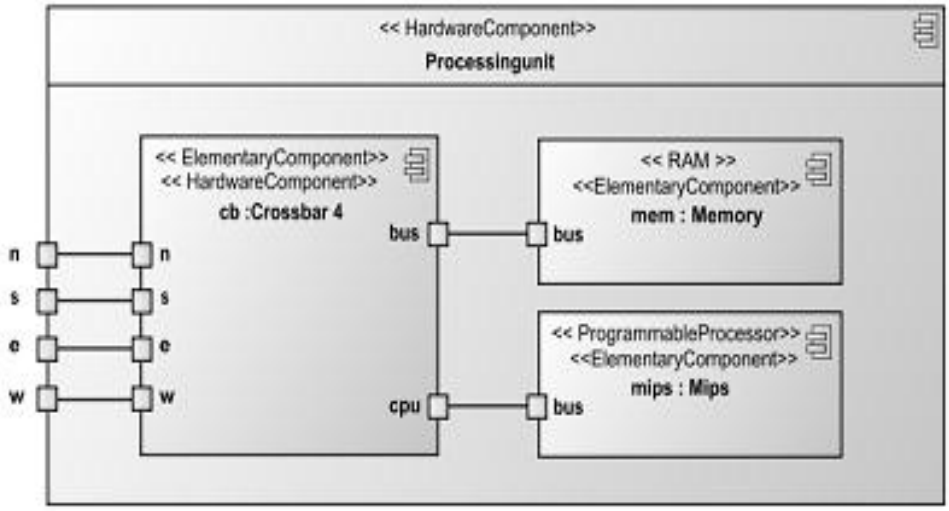


Figure 1.9 - Un exemple de modélisation par Gaspard [18]

1.10 CONCLUSION

Dans ce chapitre, la notion de systèmes embarqués a été définie, ainsi que leur évolution importante depuis l'ère des SOC jusqu'à l'ère des NOC, qui dominent maintenant sur le marché des systèmes embarqués. L'architecture de ses derniers exige que la durée et le cout de conception de ses différents composants soient réduits et limités afin de répondre à des besoins de fonctionnalité et de mise sur le marché. Aucun système embarqué ne peut être entièrement développé à la main en temps raisonnable tout en tirant l'efficacité de telles plateformes. Seule une méthode qui prend en compte en même temps toutes les subtilités et capacités de l'architecture et de l'application, et offre un environnement de développement avancé qui abstrait la complexité de tels systèmes et automatise chaque étape de développement peut atteindre des niveaux d'efficacité acceptables. Cet environnement peut être basé sur l'ingénierie dirigée par les modèles IDM.

Chapitre II

« Les Applications de Calcul Intensif »

2.1 INTRODUCTION

Le calcul intensif est considéré comme le fait d'introduire la puissance de calcul à l'exécution de l'application [1], telle que les émissions satellites, les téléphones sans fil et les réseaux internet,…

Avec l'augmentation de la quantité de données à transporter sur ces applications cela conduit à de grandes compétences en traitement systématique et intensif pour pouvoir répondre aux besoins d'accès rapide et direct aux informations d'un côté et d'adopter les meilleurs techniques de calcul pour pouvoir répondre aux exigences des consommateurs.

Les applications de traitement intensif sont de plus en plus présentes dans plusieurs domaines. Elles se trouvent aussi bien dans le domaine de calcul scientifique que dans le domaine de traitement de signal intensif (télécommunications, traitement multimédia, Traitement d'images et de la vidéo, etc.…). Le développement des systèmes logiciels se base énormément sur ces applications qui occupent une place importante dans le milieu de la recherche scientifique. Ces applications sont caractérisées par de grandes quantités de calcul régulier traité en temps réel. Elles sont aussi, généralement complexes et critiques.

Dans ce qui suit, nous essayons d'identifier les caractéristiques essentielles du domaine de la modélisation des applications de traitement intensif de signal pour l'embarqué.

2.2 TRAITEMENT INTENSIF DE SIGNAL

Le traitement du signal (TS) est la discipline qui développe et étudie les techniques de traitement, d'analyse et d'interprétation des signaux [20], il est l'une des applications des SOC les plus importantes, Il concerne l'interprétation, l'analyse, le stockage et la manipulation des signaux provenant des capteurs. Nous nous intéressons principalement aux applications de traitement de signal intensif et régulier dénotée TSI dans la figure 2.1

Les applications de traitement intensif opèrent sur une grande masse de données faisant souvent appel à des techniques de calcul parallèle et distribué. Leur principal objectif est d'assurer le traitement de grandes quantités d'informations tout en exploitant au mieux le parallélisme présent dans l'application mais il ne faut pas oublier la performance temps réel qui est un besoin clé de la plupart de ces applications.

Une modélisation capable d'exprimer les différentes caractéristiques de telles applications faciliterait l'étape de la spécification et permettrait d'effectuer de nombreuses optimisations[3].

Figure 2.1- Composition de traitement de Signal intensif [18]

2.3 EXEMPLE D'APPLICATIONS DE TRAITEMENT INTENSIF(REGULIER)

Des applications de calcul intensif sont prédominantes dans plusieurs domaines d'applications, tels que le traitement d'images et vidéo ou les systèmes de détection (radar,sonar). Quelques exemples représentatifs des applications de TSI composées d'une phase de TSS et une phase de TDI sont illustrés ci-dessous :

2.3.1 Traitement sonar : un système de détection sous-marine se base sur une première étape de traitement systématique (constituée d'une FFT qui ajoute une dimension fréquentielle aux données traitées) sur les données produites par les hydrophones (microphones répartis autour du sous-marin). Cette première phase produit des données représentant les échos captés. Ces échos sont ensuite analysés par un traitement de données : la poursuite. Ce traitement est dédié à la détection d'objets et à leur poursuite au cours du temps [21].

2.3.2 Convertisseur 16/9 - 4/3 : Consiste au passage d'un format vidéo à un autre. Ce processus de conversion se décompose en deux phases: la première consiste à créer un nouveau signal au format intermédiaire par l'interpolation du signal vidéo d'entrée 16/9. Alors la seconde phase supprime certaines pixels de ce signal de manière à ne conserver qu'une partie de l'information, cela permet le passage au format 4/3.

2.3.3 Récepteur de radio numérique : cette application en émergence fait appel à une partie frontale de TSS consistant à la numérisation de la bande de réception, la sélection du

canal et l'application de filtres permettant d'éviter les parasites. Les données fournies par ces traitements systématiques sont ensuite envoyées dans le décodeur dont le traitement est plus irrégulier (synchronisation, démodulation, etc.).

2.3.4 Radar anti-collisions : la prévention des collisions dans un système se fait par le biais d'une antenne qui renvoie un signal sous la forme d'un flux de données contenant des informations relatives à la présence d'obstacles, un écho. Ces informations seront filtrées dans une phase de traitement systématique de manière à en faire ressortir les caractéristiques intéressantes (la présence d'obstacles). Cette phase est suivie d'une analyse irrégulière de ce nouveau flux de données de manière à valider la présence d'un obstacle, à déclencher un freinage d'urgence ou à le poursuivre si nécessaire [22].Cette application est illustrée à la figure 2.2

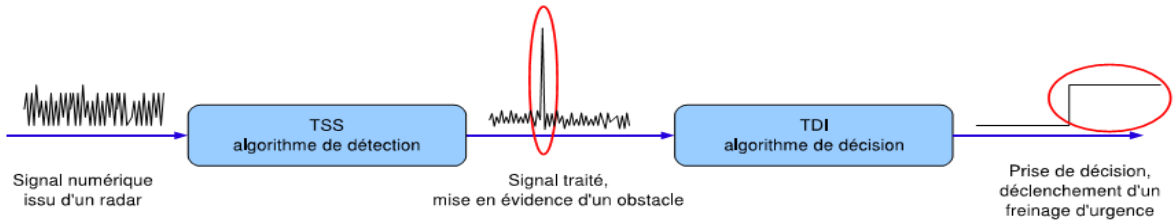


Figure 2.2 – Représentation de l'application radar anti-collision[22].

2.4 EXEMPLE D'APPLICATION DSP A TACHES IRRÉGULIERS

Si on considère des domaines d'application courants tels l'intelligence artificielle, surveillance, maintenance, médecine à distance, gestion des catastrophes, etc. avec des plateformes de calcul à files ou sans où il y a différents types de nœud (laptops, palmtop, mobile, PDA, etc.) on a ici un scénario commun typique de la transmission vidéo et audio (one to one, one to many, many to many)[5]. Le problème ici, en plus des vitesses de calcul différentes des différentes unités de traitement, se situe aussi au niveau des communications. Cette particularité est due aux bandes passantes différentes, aux périphériques hétérogènes, à la latence etc. Dans ce type d'application le transcoder, qu'on présentera juste après, joue tout son rôle.

4.1 Transcoder : Il peut être mobile ou relié à un réseau fixe. Le transcoder convertit un format vidéo en d'autres en considérant la compatibilité avec des

passantes du réseau, puissance de calcul et l'utilisation des périphériques utilisateurs [23]. Transcoder un flot vidéo se fait en deux phases :

a- Décoder X-Format

b- Encoder dans Y-Format (cf. figure 2.3)

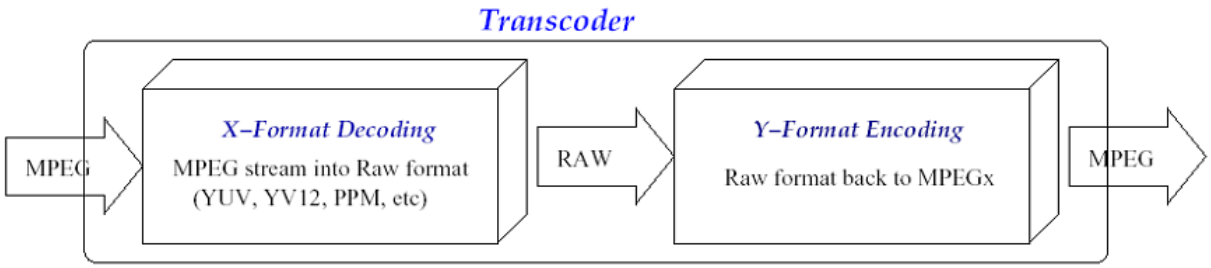


Figure. 2.3- Phases d'un Transcoder [23]

La phase encoder nécessite plus de puissance de calcul que l'autre phase (en général 3 à 4 fois plus de temps pour le même encodage) et elle est très dépendante des paramètres (tels la taille des frames). La fonction de base pour les deux étapes est :

1- Décoder, le flot MPEG dans un format ligne (YUV, YU12, PPM, etc.).

2- Encoder, transforme la ligne dans la vidéo MPEG avec les nouveaux paramètres.

2.4.2 Eliminateur de parasites (CSLC Coherent Side Love Cancellation) :

Comme indiqué dans le titre cette application est un autre exemple de traitement technique. Elle a pour but de réduire les parasites avec cohérence sur les antennes. Dans la figure 2.4, nous donnons le graphe global de l'application CSLC où on voit que la FFT est appelée deux fois et l'IFFT une seule. Ceux-ci constituent les blocs majeurs d'une application CSLC. Les autres blocs constituent des nœuds réguliers mais sont moins gourmand en termes de calcul par rapport au FFT et l'IFFT. Un tel domaine d'application est hautement à données intensives et donne plus de chance de faire du parallélisme de données que celui de tâches.

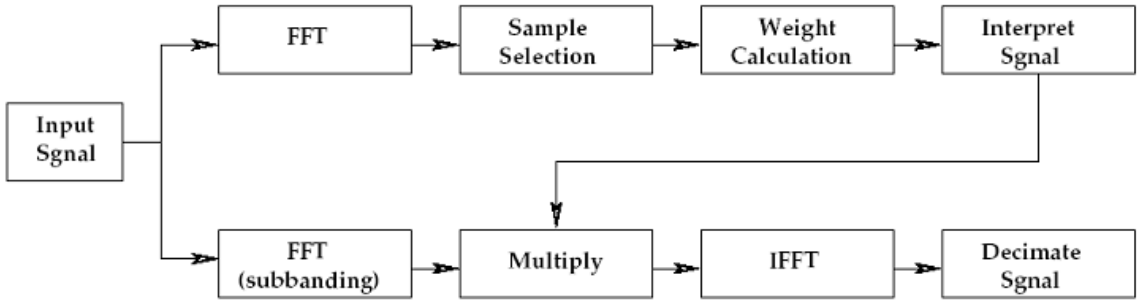


Figure 2.4 - Application CSLC (CoherentSide Love Cancellation)[5]

2.5 LANGAGE SPÉCIFICATION MULTIDIMENSIONNELLE ET MODEL DE CALCUL

On peut retrouver plusieurs modèles de calcul et description destinés essentiellement à la modélisation et mise en œuvre des applications de traitement de signal intensif. Parmi les modèles les plus répandus,on cite : MATLAB/SIMULINK, ALPHA[24], MDSDF (Multi Dimensional Synchrous Data Flow) [25], GMDSDF (Generalized Multi Dimensional Synchronous Data flow)[26] et le langage Array-OL (Array Oriented Langage).

Les différentes applications qu'on a citées sont d'habitude massivement parallèles, complexes et qu'elles manipulent des structures de données multidimensionnelles et les modeles de calcul qu'on va citer, ce sont les modèles qui peuvent répondre à leurs exigences.

2.5.1 Alpha

Alpha [27], est développé au laboratoire Irisa de Rennes, c'est un langage à parallélisme de données permettant la description de haut niveau d'algorithmes de calcul réguliers, il est aussi un langage fonctionnel, à assignation unique et fortement typé qui permet de représenter les algorithmes sous forme d'équations récurrentes [28]. La notion de temps, ou même d'exécution, n'existe pas en Alpha : Les dépendances entre les calculs imposent des conditions sur l'ordre dans lequel ils peuvent être exécutés.

En Alpha, les variables sont définies à l'aide de fonctions sur le domaine Zn. En effet, les données manipulées par Alpha sont multidimensionnelles : Leurs formes ne sont pas restreintes à de simples tableaux rectangulaires.

L'exemple suivant montre la déclaration d'une variable dont le domaine est l'ensemble des points dans le triangle :

$0 \leq i \leq j; j \leq 10: \mid a: \{i, j \mid 0 \leq i \leq j; j \leq 10\}$

Afin d'accéder aux différentes valeurs de ces données, il est possible de faire des restrictions sur les domaines. On se sert pour cela de l'instruction case.

```
a =
Case
{i, j | j = 0}   0. (i, j ->);

{i, j | j > 0}   a. (i , j -> i, j+1)+1. (i, j ->);
```

Dans cet exemple, nous avons :

a [i; j] = 0 lorsque j = 0
a [i; j] = a [i; j + 1] + 1 lorsque j > 0

Alpha se révèle donc être en mesure d'exprimer de façon simple des formes de données très complexes, mais il s'avère incapable de gérer les accès cycliques.

2.5.2 StreamIt

StreamIt [29], développé au Massachusetts Institute of Technology, pour faciliter la programmation de grande applications de type streaming. C'est un langage impératif utilisant la programmation orienté objet, il reprend la syntaxe de langage JAVA, il est d'ailleurs possible d'utiliser un compilateur JAVA sur un code StreamIt après une phase de traduction. En StreamIt les filtres constituent la classe de base.

Elles sont constitués principalement de deux méthodes : la première contient la liste des traitements et des communications réalisés par un filtre en utilisant des FIFO's et les trois méthodes Push, Pop et Peek. Alors la deuxième est pour l'initialisation du filtre et ses FIFO's et surtout pour positionner le nombre de données qui seront consommées ou produites par Push, Pop et Peek dans la première méthode.

En StreamIt, Le passage simple et direct de la spécification vers l'implémentation le rend très attractif, mais il est assez limité au niveau de l'expressivité ; il sait manipuler seulement des flots de données mono dimensionnels.

2.5.3 MATLAB/SIMULINK

MATLAB [30] est un puissant logiciel de calcul et peut être vu comme un système interactif et convivial de calcul numérique et de visualisation graphique où beaucoup de ses fonctions sont basées sur le calcul matriciel simplifié. Il permet de distinguer plusieurs types de matrices : Monodimensionnelles (vecteurs), bidimensionnelles (matrices), tridimensionnelles...etc.

SIMULINK est l'extension graphique de MATLAB qui est un langage permettant la représentation des fonctions mathématiques sous forme de diagrammes structurels facilitant la modélisation, l'analyse et la simulation d'une large variété de systèmes physiques et mathématiques.

Pour beaucoup, MATLAB/SIMULINK est devenu un standard pour la conception de systèmes. Cependant, ils est une pure création de numériciens et d'automaticiens, et n'a aucune des qualités informatiques requises pour la génération du code pour des systèmes embarqués critiques. Ce sont des logiciels principalement conçus pour la simulation des applications de traitement numérique et ne prennent pas en considération l'expression et l'exploitation du parallélisme de données présent dans ces applications.

2.5.4 SDF (SynchronousDataFlow)

SDF [31,32] a été créé et développé par Lee et Messerschmitt à partir de 1986, c'est un modèle de calcule à flot de données permettant la description d'application du traitement de signal. L'ajout du terme « synchrone » implique que le nombre de données consommées et produites soit connu dès la conception de l'application, ce qui permet de réaliser des ordonnancements statiques.

En SDF les calculs d'une application sont représentés par des nœuds (actor) et les données (tokens) par des arcs, Chaque nœud consomme des données en entrée pour produire des résultats en sortie, La figure 2.5 illustre une application en SDF.

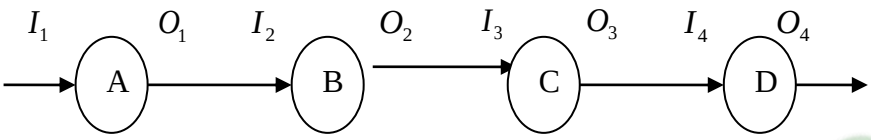


Figure 2.5 - Exemple d'une application en SDF

En SDF, l'application est définie statiquement. Il est donc possible de détecter les inters blocages dès la conception, d'ordonnancer l'application dès la modélisation et d'avoir une exécution déterministe (résultats identiques pour quelconque ordonnancement).

Calcul de l'ordonnancement :

Le calcul de l'ordonnancement se base sur la constatation suivante : lorsque l'application est exécutée, le nombre total de jetons produits par une tâche est égal au nombre de jetons consommés par la tâche suivante.

Soit I_x et O_x le nombre de jetons respectivement consommé et produit par la tâche X^e et soit r_x le nombre d'exécution de cette tâche (cf. figure 2.5), on peut alors écrire l'équation suivante:

$r_x O_x = r_{x+1} I_{x+1}$ …… ***(2.1)***

Dans le cas où ($\vec{r}$) est indéfinie, cela veut dire que le système d'équation (2.1) n'a aucune solution et le graphe est inconsistant et il ne peut être ordonnancé.

Dans le cas contraire, l'ensemble des solutions est décrit par $k\vec{r}$ avec $k \in N^+$ et le graphe de dépendance et l'ordonnancement de l'application sont obtenus facilement grâce aux résultats présentés dans [32].

Exemple : Soit l'exemple présenté sur la Figure2.6 : la première tâche produit trois jetons et

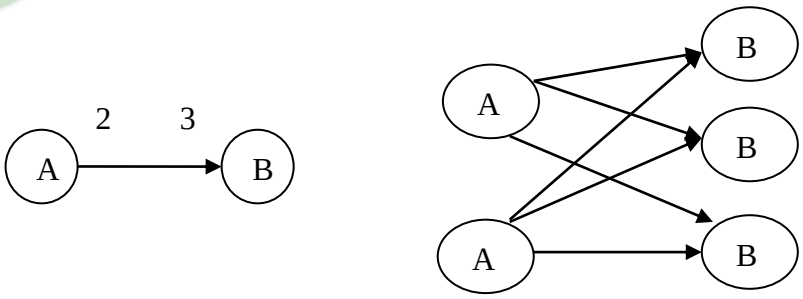


Figure 2.6 - Graphe de dépendance d'une application SDF

la deuxième en consomme deux. La résolution du système d'équation donne $\vec{r} = \binom{2}{3}$ et le graphe de dépendance obtenu est présenté sur la droite de la figure.

2.5.5 MDSDF (Multi Dimensional Synchronous Dataflow) :

Le modèle MDSDF (Multi Dimensional Synchronous Dataflow), proposé par Edward Lee en 2002 [25], étend le concept du modèle SDF [33, 34] pour l'appliquer dans un contexte multidimensionnel.

Le fonctionnement de MDSDF est similaire à celui de SDF. Ainsi, il suffit juste de préciser pour une tâche le nombre de données consommées et produites sur chacune des dimensions du flot. Ce principe est illustré par la Figure 2.7

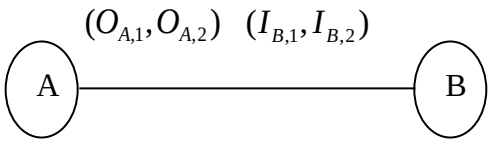
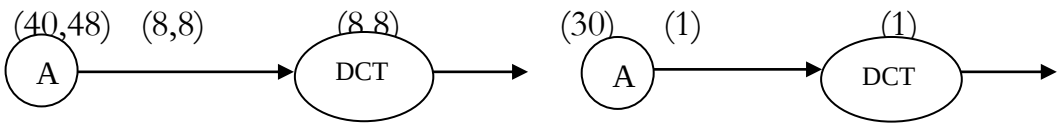


Figure 2.7-Une application MDSDF [34]

Afin de comparer les deux modèles MDSDF et SDF en termes d'expression d'une application donnée, Les figures Subfig. 2.8. a et Subfig. 2.8. b montrent la même application décrite respectivement avec MDSDF et avec SDF.

Il apparaît que la première tâche, en SDF, produit trente jetons et que la deuxième les consomme un par un. SDF ne permet donc pas d'exprimer les dépendances de façon optimale lorsque la structure du flot de données est multi dimensionnelle.



Subfig 2.8.a : en MDSDF Subfig 2.8.b : en SDF

C'est le contraire avec le MDSDF qui rend possible la modélisation d'application qui ne pouvait être décrite en SDF.

Calcul de l'ordonnancement :

Le calcul de l'ordonnancement se fait de la même façon que celui de SDF, sauf qu'ici on a pour chaque dimension un système d'équation à résoudre.

$$r_{A,1}O_{A,1} = r_{B,1}I_{B,1}$$
$$r_{A,2}O_{A,2} = r_{B,2}I_{B,2} \quad \text{……..} \ (2.2)$$

Le nombre total d'itérations d'une tâche est égal au produit du nombre d'itérations pour chacune des dimensions.

L'utilisation du MDSDF pour les applications du traitement intensif est limité à un nombre de dimension supérieur à deux, pour ces raisons nous introduisons dans la section suivante un autre modèle de calcul multidimensionnel appelé ARRAY-OL qui ne présente pas ce genre d'inconvénients.

2.6 ARRAY-OL

ARRAY-OL (Array Oriented langage) : est un langage spécialisé dans la description d'application de traitement de signal systématique. Ce type d'application est caractérisé par la manipulation de grandes quantités de données qui sont traitées par un ensemble de tâches de façon régulière.

Un bon langage de modélisation pour ce genre d'application devrait être un langage déterministe, arrive à exploiter le maximum de parallélisme trouvé dans l'application, que ce soit le parallélisme de données ou le parallélisme de taches et assure et respecte en même temps les contraintes de temps et de ressources d'exécution.

Tout ces exigences et caractéristiques sont couvertes par les principes de base du langage ARRAY-OL (ARRAY Oriented Langage) [33,21].

Le but initial d'ARRAY-OL est de fournir un langage mixte graphique et textuel pour modéliser des applications multidimensionnelles de traitement intensif de signal.

Etant donnée une application décrite en ARRAY-OL, il est facile de calculer l'ordonnancement de cette dernière en utilisant la sémantique formelle et la définition du noyau d'ARRAY-OL qui sont détaillées dans [35].

Formellement l'application en ARRAY-OL, est une série de taches connectées par des ports [34]. Les données sont contenues dans des tableaux, les flux de données sont eux représentés

par des tableaux ayant une dimension infinie. La lecture de ces données est faite sur les ports d'entrée et l'écriture est faite sur les ports de sortie.
Les accès aux données sont faits par des sous tableaux, appelés motifs, caractérisés par leur forme et le nombre d'éléments sur chacune de leur dimension.
Les tâches sont de trois types : élémentaire, composée et répétition [34].
ARRAY-OL est un langage à assignation unique où seules les dépendances de données sont exprimées et les dimensions spatiales et temporelles sont traitées de la même façon dans les tableaux. En particulier, le temps est expansé comme une dimension des tableaux.
Le modèle ARRAY-OL est multidimensionnel, sa seule limitation sur les dimensions des tableaux est qu'il doit y avoir une seule dimension infinie par tableau qui représente généralement le temps. De plus, les tableaux utilisés dans ARRAY-OL sont considérés comme toriques dans le sens où la consommation ou la production de leurs éléments peut se faire modulo à leur taille.
ARRAY-OL permet d'exprimer un parallélisme potentiel complet sur les applications, que ce soit un parallélisme de données ou un parallélisme de tâches.

2.6.1 Parallélisme de tâches

Graphiquement, une tache composée en ARRAY-OL est décrite par un graphe orienté acyclique ou chaque nœud représente une tâche et chaque lien une dépendance entre deux ports. Il est possible pour une tâche de consommer deux tableaux bidimensionnels et de produire un tableau tridimensionnel donc il n'y a pas de relation entre les formes des ports d'entrée et de sortie d'une tâche.
De cette description, il est facile de calculer l'ordonnancement d'une application mais vu le manque de détails sur les calculs réalisés par une tache, il est impossible d'exprimer le parallélisme de données de l'application en question.

2.6.2 Parallélisme de données

Une application intensive est décrite comme un ensemble de tâches indépendantes qui ont la même structure et les mêmes caractéristiques, elles s'appliquent sur des données de même structure et dimension avec des valeurs différentes qu'on appelle motifs.
Afin de pouvoir décrire l'accès aux motifs et la création de ces données, un Tiler est associée à chaque couple «tâche_tableau». Ce Tiler est utilisé pour l'extraction des motifs à partir d'un

tableau d'entrée et le rangement des motifs dans un tableau de sortie. Chaque Tiler contient les informations suivantes :

$\vec{O}$: L'origine vecteur du motif de référence

P : La matrice de pavage (paving matrix) qui permet de décrire comment les motifs couvrent le tableau;

F : La matrice d'ajustage (fitting matrix) qui décrit comment remplir le motif par les éléments du tableau;

Un exemple de la description visuelle d'une répétition est donné dans la Figure 1.9 où on retrouve la répétition du Produit Scalaire de l'application produit de matrices.

Les Tilers sont associés aux dépendances qui font le lien entre les tableaux et les motifs.

Pour énumérer les motifs, chaque lien dispose d'une matrice de vecteurs de pavage (paving) et d'un point de départ appelé origine (origin). À partir de ces deux éléments, il est possible de calculer le premier point de chaque motif.

Les coordonnées de ces points sont calculées comme la somme des coordonnées de l'origine et d'une combinaison linéaire des vecteurs de pavage, le tout modulo la taille du tableau (équation 2.3).

$$\forall r, 0 \le r < S_{répétition}, réf_r = O + P.r \bmod S_{tableau} \quad \text{…….. (2.3)}$$

Où $S_{tableau}$ est la forme du tableau.

Les éléments d'un tableau constituant un motif sont calculés comme la somme des coordonnées du premier élément de ce motif et d'une combinaison linéaire de la matrice d'ajustage, le tout modulo la taille du tableau (équation 3.4).

$$\forall i, 0 \le i < S_{motif}, e_i = réf = F.i \bmod S_{tableau} \quad \text{…….. (2.4)}$$

Où S_{motif} est la forme du motif et F la matrice d'ajustage.

Les formules précédentes décrivent quels éléments d'un tableau d'entrée ou de sortie sont consommés ou produits par une répétition.

Le lien entre les entrées et les sorties est fait par l'indice de répétition, r. Pour une répétition donnée, les motifs de sorties (de l'indice r) sont produits par la tâche répétée depuis les motifs d'entrées (de l'indice r).

Pour bien comprendre l'utilisation du langage ARRAY-OL, nous prenons l'exemple d'un produit de matrices (Figure 3.10). Cet exemple permet de montrer la puissance du modèle pour l'expression du parallélisme de données dans un algorithme.

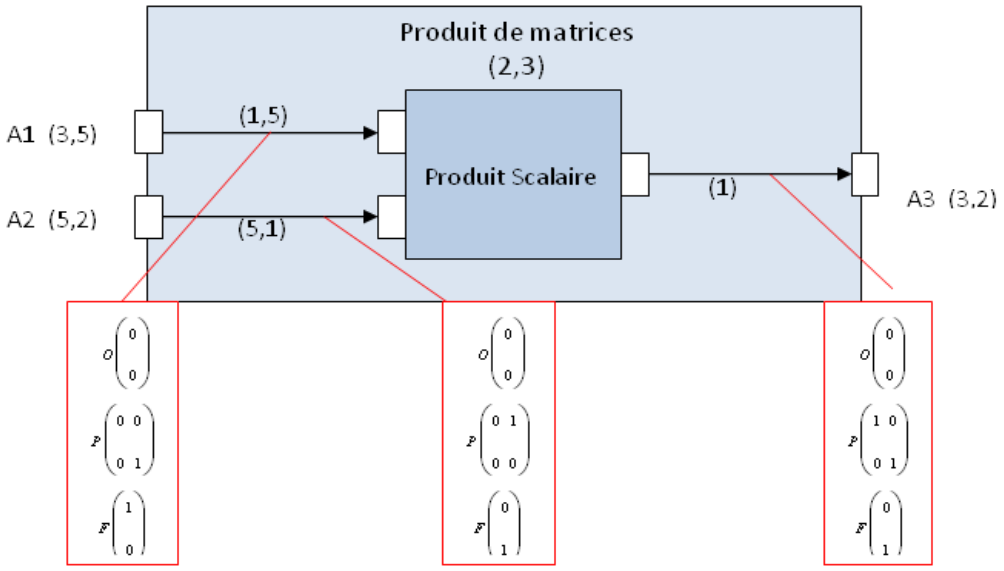


Figure 3.10 - Exemple d'un produit de matrices [36]

Nous calculons le produit A1×A2 = A3 avec A3 de taille 3 × 2, A1 une matrice 3 ×5, et A2 une matrice 5×2. Le calcul d'un produit de matrice revient au calcul du produit scalaire de chaque ligne d'A1 par chaque colonne d'A2. Les différents produits scalaires sont indépendants et peuvent être alors effectués en parallèle.

2.7 COMPARAISON ENTRE ARRAY-OL ET MDSDF

ARRAY-OL et *MDSDF* sont les seuls modèles existants permettant une description de haut niveau des applications multidimensionnelles pour le traitement du signal systématique [37]. Du point de vue théorique, qui est mieux que l'autre ? Cette section a pour but de faire une petite comparaison entre ces deux langages afin de répondre à cette question.

ARRAY-OL et *MDSDF* ont chacun leurs forces et leurs faiblesses, mais le premier a plus d'avantages sur le deuxième. En effet, malgré que le nombre d'informations à fournir pour *ARRAY-OL* est plus grand et qui est estimé à trois ou à quatre fois que celui à fournir pour *MDSDF*, l'utilisation de tableaux toriques donne l'avantage à *ARRAY-OL*. Cette option est très intéressante réellement vu son utilisation pour modéliser plusieurs applications dont celle présentées précédemment, où une des dimensions toriques représente les sonars se trouvant autour d'un sous-marin et où une autre représente une dimension fréquentielle.

De plus, la manipulation de la structure des données et les calculs sur ces données en *ARRAY-OL* ne s'effectuent pas au même niveau de modélisation et les phases de modélisation et d'ordonnancement sont clairement séparées [37].
Mais la différence la plus notable entre ces deux langages est sans nul doute qu'*ARRAY-OL* est le seul à ne souffrir d'aucune limite sur le nombre de dimensions ce qui fait de lui le seul modèle réellement multidimensionnel.

2.8 CONCLUSION

Nous avons défini dans ce chapitre les applications et les domaines où on fait du calcul intensif, ainsi que la problématique de la modélisation de ce type d'applications. Pour ces derniers, nous avons cité les langages de spécification les plus connus. L'objectif essentiel de ces langages est de permettre avec une relative simplicité de mettre en évidence le parallélisme de tâches ou de données inhérent à ces applications.

Chapitre III

« Optimisation Multi-Objectifs »

3.1 INTRODUCTION

L'optimisation multi-objectifs est une branche fondamentale de l'aide à la décision multi- critère, au quelle la plupart des problèmes réels nécessitent de faire face.

Dans le cas de l'optimisation mono-objectif, la solution optimale est facilement définie. Ce qui n'est pas le cas quand on a plusieurs objectifs. Au lieu d'une solution unique, le résultat d'une proposition multi-objectif est généralement un assortiment de solutions, qui se distinguent par différents compromis réalisés entre les objectifs. Les solutions qui le composent sont optimales, dans le sens où il n'existe dans l'univers de recherche aucune solution meilleure si tous les objectifs sont considérés simultanément.

Dans le monde industriel, généralement, les problèmes d'optimisation sont de nature multi-objectifs puisque plusieurs critères permettent de caractériser une solution. Pour les *SOC*, objet de notre étude, les critères les plus connus sont le temps de calcul, la consommation d'énergie, les dimensions de la puce, taille mémoire, etc.

Satisfaire ces critères est notre objectif, ce qui revient à optimiser les performances du système par rapport à ces objectifs. En améliorant chacun sans pour autant détériorer les autres ; ce n'est pas évident à faire, là on est en train de faire de l'optimisation multicritère ou bien multi-objectifs. Donc on est face à un problème multi objectif (PMO), pour la résolution de ce problème il existe plusieurs méthodes, purement mathématiques, inspirées du réel ou bien le mélange des deux. Ainsi, le but de l'Optimisation Multi-objectifs, consiste à obtenir les solutions des compromis possibles entre les objectifs.

3.2 DEFINITIONS

Cette section couvre les principaux concepts, définitions et notations liés à l'optimisation multi objectifs, tels que la relation de dominance Pareto, l'optimalité Pareto, l'ensemble Pareto optimal, et le front Pareto. De nombreuses définitions sont tirées du livre de l'Ehrgott (2005).

3.2.1 Modèle Mathématique d'un problème Multi Objectif

Un problème d'optimisation multi objectif (multi-objective optimisation, MOP) : peut être défini comme suit :

$$MOP = \begin{cases} \min\ f(x) = (f_1(x), f_2(x), ..., fk(x)) \\ telque\ \ x \in C \end{cases}$$

Où k > 2 est le nombre de fonctions objectifs, x= (x_i,..., x_j) est le vecteur représentant les variables de décision, C représente l'ensemble des solutions réalisables associées à des contraintes d'égalité, d'inégalité et des bornes explicites et F (x) = (f_1(x), f_2(x),..., f_k(x)) est le vecteur des objectifs à minimiser. Dans le cas combinatoire, c'est un ensemble discret. A chaque solution $x \in C$ est associer un vecteur objectif $z \in Z$ sur la base d'un vecteur de fonctions $f : C \rightarrow Z$ tel que z=(z_1,z_2,z_3.....z_n)= f(x)=(f_1(x),f_2(x),.....f_n(x)),ou Z=f(C) représente l'ensemble des points réalisables de l'espace objectif (figure3.1) sans perte de généralité, nous supposerons par la suite que z c R^n et que toutes les fonctions objectif sont à minimiser.

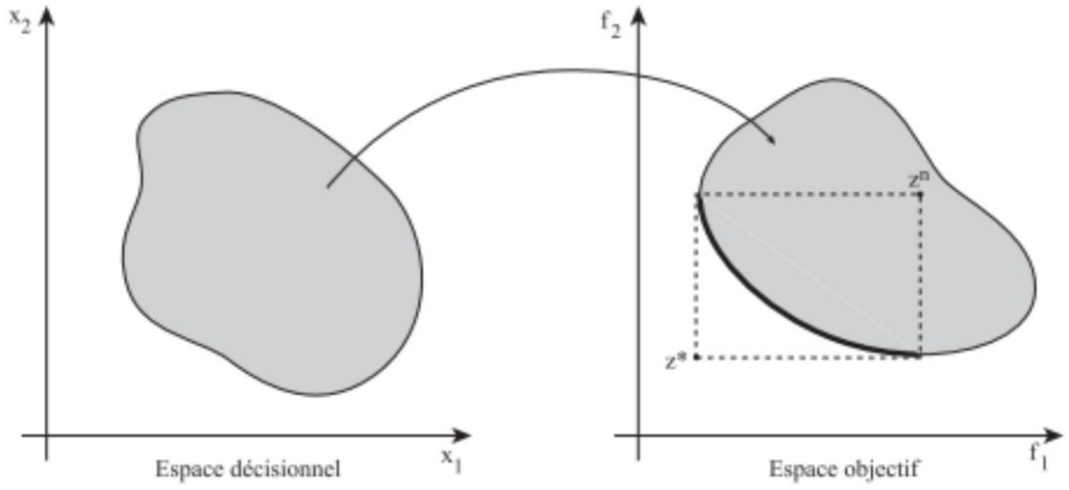


Figure 3.1- Espace décisionnel et espace objectif d'un problème d'optimisation multi objectif (exemple avec deux variables de décisions et deux fonctions objectif) [38]

3.2.2 Notion de dominance Pareto et d'optimalité

Dans le domaine de l'optimisation multi objectif, le décideur évalue généralement une solution par rapport à chacun des critères, et se positionne donc naturellement dans l'espace objectif. Néanmoins, contrairement au cas mono objectif ou il existe un ordre total parmi les solutions réalisables, il n'existe généralement pas de solution qui serait à la fois optimale pour

chaque objectif, étant donnée la nature conflictuelle de ces derniers. Ainsi, une relation de dominance, d'ordre partiel, est généralement définie.

3.2.2.1 Dominance Pareto

Un vecteur objectif $z \in Z$ domine un vecteur objectif $z` \in Z$ ssi $\forall_i \in \{1, \ldots, n\}$, $z_i \leq z_i`$et $\exists\ j \in \{1, \ldots, n\}$ tel que $z_j \leq z_j`$.cette relation sera notée $z > z`$. Par extension, une solution $x \in X$ domine une solution $x` \in X$ , noté $x > x`$, ssi $f(x) > f(x`)$.

3.2.2.2 Solution Pareto optimale

Une solution $x \in X$ est Pareto Optimale (ou non-dominée) ssi $\forall\ x` \in X$ , $x` > x$ .Alor, Tout solution Pareto optimale peut être considérée comme optimale puisqu'aucune amélioration n'est possible sur la valeur d'un objectif sans en dégrader celle d'un autre ,la notion de Pareto optimalité, initialement utilisée en économie et dans les sciences de management, prend ses racines dans les travaux de Edgeworth (1881) et Pareto (1896),Ces solutions Pareto optimales forment l'ensemble Pareto Optimale, noté X_E .l'image de cet ensemble dans l'espace objectif est le front Pareto, et sera noté Z_N.

3.2.2.3 Ensemble Pareto optimal

Étant donné un MOP (fx), l'ensemble Pareto optimal est défini comme suit :

$X_E = \{x \in X | \nexists\ x` \in X, x` > x\}$

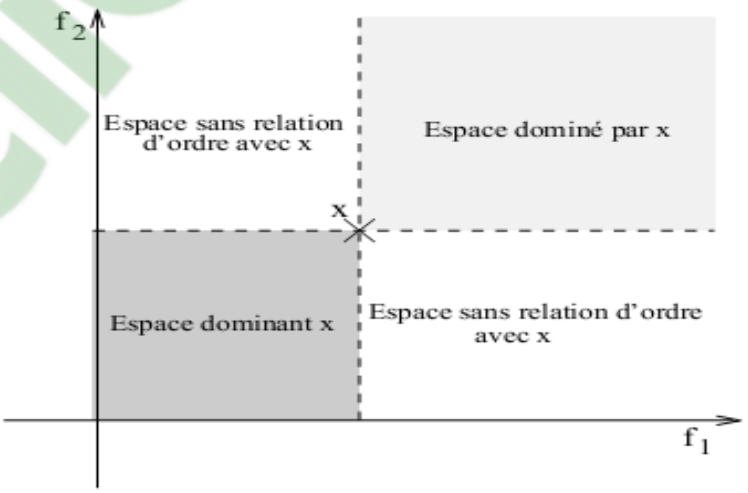


Figure 3.2- Notion de dominance [39]

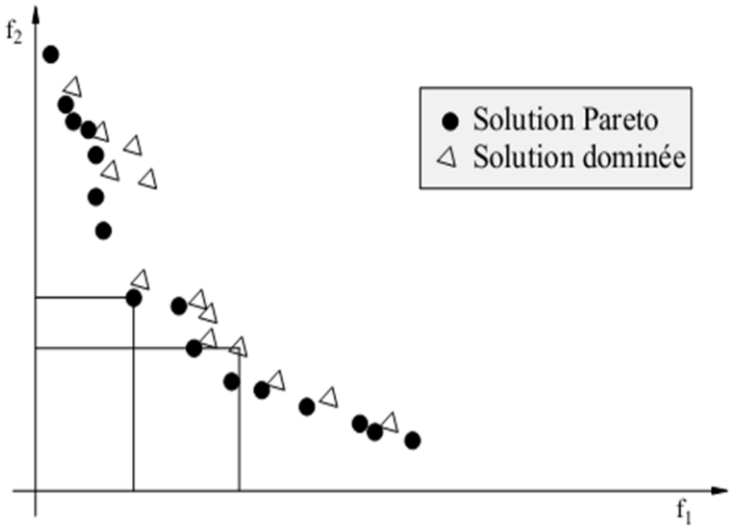


Figure 3.3- Exemple de Front Pareto Optimal [39]

3.3 CLASSIFICATION DES METHODES DE RESOLUTION

Cette section présente les différentes approches de résolution de l'optimisation multi objectif. Dans la littérature, une attention particulière a porté sur les problèmes à demi critères en utilisant les méthodes exactes tels que le *"branch and bound"*, l'algorithme *A** et la programmation Dynamique. Ces problèmes sont efficaces pour des problèmes de petites tailles. Pour des problèmes à plus de deux critères ou de grandes tailles, il n'existe pas de procédures exactes efficaces, étant donné les difficultés simultanées de la complexité NP-difficile, et le cadre multicritère des problèmes.

Ainsi, des méthodes heuristiques sont nécessaires pour résoudre les problèmes de grandes tailles et/ou les problèmes avec un nombre de critères supérieurs à deux. Elle ne garantit pas de trouver de manière exacte l'ensemble *PO (Pareto optimale)*, mais une approximation de cet ensemble noté *PO**. Les méthodes heuristiques peuvent être divisées en deux classes : d'une part les algorithmes spécifiques à un problème donné qui utilisent des connaissances du domaine, et d'autre part les algorithmes généraux applicables à une grande variété de PMO, c'est-à-dire les métas heuristiques qui feront l'objet de notre intérêt dans la suite de ce chapitre. Plusieurs adaptations de méta heuristiques ont été proposées dans la littérature pour

la résolution de PMO et la détermination des solutions Pareto : le recuit simulé, la recherche tabou et les algorithmes évolutionnaires (algorithmes génétiques, stratégies évolutionnaire).

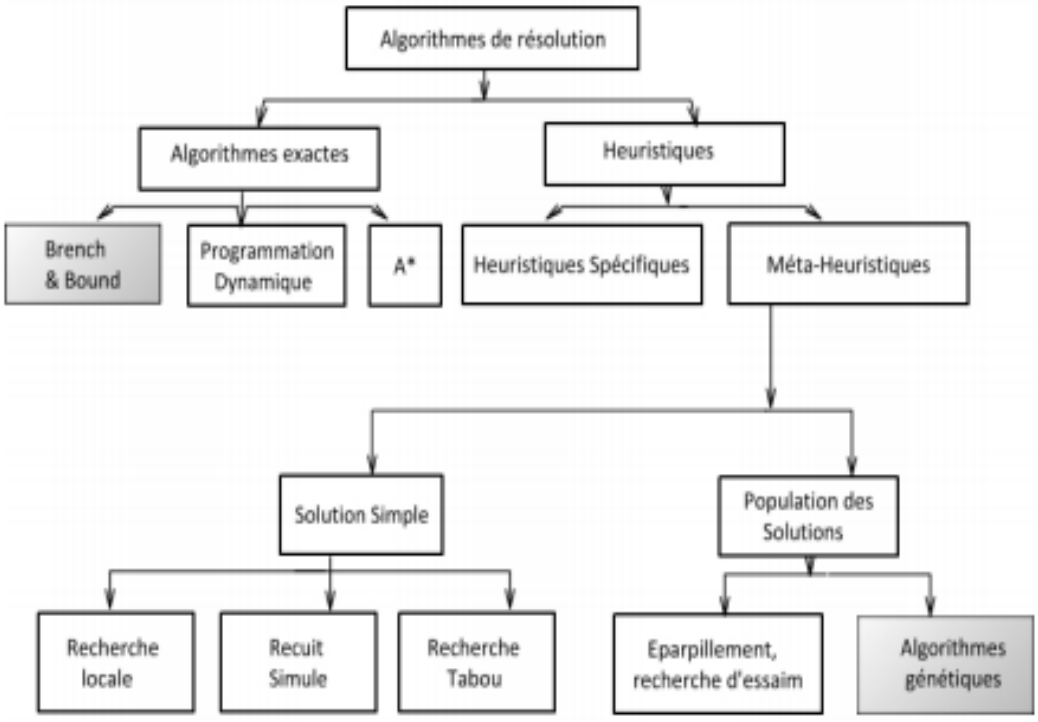


Figure 3.4 - Classification des méthodes d'optimisation multi-objective[13]

La figure 3.4 illustre la classification des différentes méthodes de résolution des PMO's, les rectangles en gris sont les méthodes qu'on va détailler plus tard dans ce chapitre.

Ci-dessous les différentes approches de résolution par rapport à la notion de multi-objectif et la notion Pareto.

3.3.1 Approches basées sur la transformation du problème en un problème uni-objectif

Cette classe d'approches comprend par exemple les méthodes basées sur l'agrégation qui combinent les différentes fonctions : coût f(x) du problème en une seule fonction objectif F. Ces approches nécessitent pour le décideur d'avoir une bonne connaissance de son problème.

3. 3.2 Approche non-Pareto

Les approches non-Pareto ne transforment pas le PMO en un problème uni-objectif. Elles utilisent des opérateurs de recherche qui traitent séparément les différents objectifs.

3.3.3 Approche Pareto

Les approches Pareto utilisent directement la notion d'optimalité Pareto dans leur processus de recherche. Le processus de sélection des solutions générées est basé sur la notion de non-dominance. Parmi ces trois approches c'est cette dernière qui sera retenue lors de la résolution de notre problème.

3.4 LES ALGORITHMES D'OPTIMISATION

3.4.1 Les algorithmes génétiques "AGs"

Dans la nature, les créatures évoluent suivant des critères de sélection naturelle qui s'appuient sur leurs adaptation (fitness) avec l'environnement (la sélection naturelle: Théorie de Darwin [40]. Une telle sélection ne laisse qu'une très petite chance aux individus les moins performants pour survivre. Donc, ce sont les plus adaptés qui vont reproduire un nombre relativement large de descendants. De surcroît, on peut dire que lors d'une reproduction, la combinaison des bonnes caractéristiques de chaque ancêtre donne lieu à un descendant amélioré, ce qui fait, que le descendant aura une plus grande fitness que ses parents pour survivre.

John Holland a développé cette idée dans «Adaptation in Natural and artificiel system» dans les débuts des années 1970. Par la description du comment appliquer les principes de l'évolution de la nature dans des problèmes d'optimisation par exploitation de l'univers des solutions possibles afin de converger vers la (les) solution(s) optimales(s) [41]. Ce qui a donné lieu aux algorithmes génétiques, qui vont être le sujet de ce document.

3.4.1.1 Définitions et Terminologies

Les algorithmes génétiques (AGs) appartiennent à la famille des algorithmes appelée méta heuristiques évolutives. Le but de cette catégorie d'algorithmes est d'obtenir une solution convenable à un problème d'optimisation en un temps correcte (admissible). De même, le but des algorithmes génétiques est de se rapprocher au maximum d'une solution optimale, sans malheureusement garantir si elle est la meilleure [42].

Les algorithmes génétiques sont directement dérivés de l'évolution de la nature. Pour mieux comprendre leurs théories, il est important d'introduire quelques terminologies biologiques [41].

Gène: est l'unité responsable de l'hérédité qui contrôle les caractères et/ou les aptitudes propres à un organisme. Les chromosomes se décomposent en gènes qui peuvent prendre des valeurs différentes que l'on appelle allèles.

Individu: les chaînes des systèmes génétiques artificiels sont analogues aux chromosomes des systèmes biologiques. Ils portent les informations génétiques d'un individu.

Population: les AGs n'opèrent pas sur un seul individu à la fois, mais sur une population d'individus afin d'effectuer des opérations de recherches sur un domaine de possibilités plus importantes. A chaque instant t, la population est appelée «génération».

3.4.1.2 Principe des « AGs »

Les algorithmes génétiques opèrent sur une population de solutions possibles, où chaque solution est représentée par un chromosome (la représentation abstrait de la solution). Donc, coder les solutions possibles sous forme de chromosome est la première étape. Par la suite, un ensemble d'opérations de reproduction vont être appliquées sur les chromosomes, pour aboutir à la création de nouvelles solutions du problème. Dans cette section, nous allons introduire les différentes étapes du processus d'exécution d'un algorithme génétique:

3.4.1.2.1 La représentation génétique :

Comme première étape avant l'application des algorithmes génétiques sur un problème, il est nécessaire de définir une représentation génétique des solutions. Il faut trouver une structure pour le chromosome afin de coder les solutions, cette structure est appelée la représentation génétique [43]. Le choix du codage est lié à la formalisation mathématique du problème. Cependant, il n'existe aucune règle qui impose le codage à choisir. Parmi les codages que nous pouvons utiliser: le codage binaire, codage à caractère multiples….etc

3.4.1.2.2 Fonction d'adaptation (Fitness)

Une fois la représentation des solutions est déterminée, la deuxième étape consiste à associer pour chaque solution (chromosome) une valeur d'évaluation qui correspond à sa fitness, et qui indiquera la qualité de l'individu. La fonction de fitness est déterminée selon la nature du problème, elle dépend de ce qu'on veut optimiser. Cependant, le choix de cette fonction affecte les résultats de la sélection des individus qui contribueront à la création des nouveaux individus.

Généralement, les individus ayant une fitness supérieure vont avoir une grande probabilité de se reproduire. En d'autres termes, ils auront plus de chances pour contribuer à la création d'une nouvelle solution.

3.4.1.2.3 Population initiale

Comme indiqué dans la définition des, les *AGs* opèrent sur des populations d'individus, et non pas sur un seul individu. A partir d'une population initiale de solutions, les algorithmes génétiques créent de nouvelles générations de solutions améliorées, ces dernières participeront aussi à la génération de nouvelles populations. Cependant, il est nécessaire de choisir une population initiale avant de lancer le processus des algorithmes génétiques. Donc, comment choisir cette population ? Et qu'elle est sa taille ?

Le choix de la population initiale est très crucial à la réussite du processus des AGs, la diversité de l'ensemble des individus initiaux influe sur la qualité des résultats obtenus. Il est recommandé de bien choisir les individus initiaux, ainsi que leurs nombre (taille de la population), afin de pouvoir exploiter le maximum de l'espace de recherche.

Généralement les individus initiaux sont choisis aléatoirement. Cependant, parfois des heuristiques peuvent être utilisées pour déterminer ces individus initiaux. Par exemple, choisir des individus ayant une grande fitness. Mais tout en conservant la diversité. Le manque de diversité peut mener à exploiter qu'une partie de l'espace de recherche, ce qui peut engendrer la convergence vers une solution optimale mais locale (l'optimum d'un sous ensemble de l'espace de recherche), qui ne sera pas forcément l'optimale de l'ensemble globale.

3.4.1.2.4 Sélection d'individus

Le rôle de la sélection est de choisir les individus pour participer à la prochaine étape qui est la génération de nouveaux individus (création d'une nouvelle population). Entre autre, éliminer les individus qui sont moins bons, et ne doivent pas participer à la reproduction. La sélection se base sur la valeur de la fonction fitness des individus [44]. Lors de la sélection, un individu peut être sélectionné une ou plusieurs fois, comme il peut ne pas être sélectionné du tout [45]. Il existe plusieurs stratégies de sélection. Parmi lesquels on cite: sélection par roulette (Wheel), Sélection par rang, Steady State , Sélection par tournoi ..etc

3.4.1.2.5 Croisement

Le croisement, où bien la recombinaison, est le processus de produire un ou plusieurs individus fils à partir de deux parents. Il existe plusieurs manières de faire la reproduction, parmi lesquelles on trouve la plus connue qui est le croisement multipoints [45]. Le principe de ce croisement consiste à prendre deux chromosomes parents, les découper en N+1 segment par N points (choisis aléatoirement où suivant une stratégie déterminée par le concepteur de l'AG). Donc, un fils sera créé en choisissant alternativement des segments de chaque parent. Cette opération est illustrée dans l'exemple suivant:

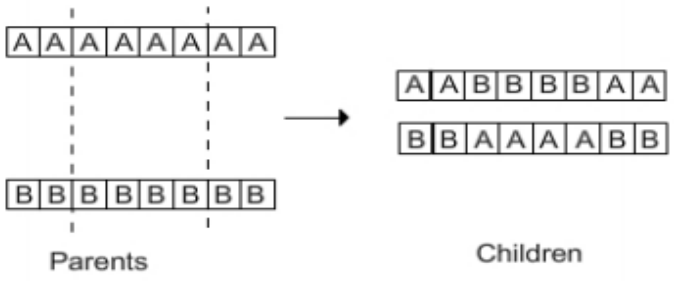


Figure 3.6 : Croisement multipoints

De plus que le croisement multipoints, il existe un autre type de croisement qui est le croisement uniforme [3]. Le principe de ce croisement consiste à faire copier un gène de chaque parent en utilisant un masque de croisement. Ce qui est illustré dans l'exemple suivant:

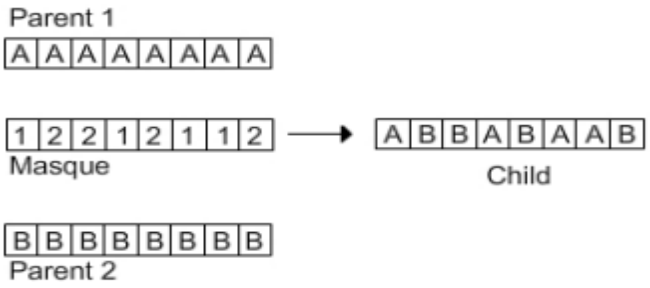


Figure 3.7 Croisement uniforme

3. 4.1.2.6 Mutation

La mutation est une simple opération qui consiste à modifier les valeurs des gènes d'un chromosome, elle est utilisée dans le cas où les nouveaux individus nécessitent des

modifications. D'un autre côté, la mutation peut garantir l'exploitation de tout l'espace de recherche par le maintien de la diversité de la population [46].

Il existe plusieurs formes de mutation pour les différents types de représentation des individus. Par exemple, pour la représentation binaire, la mutation consiste à inverser la valeur des gènes comme illustré dans l'exemple:

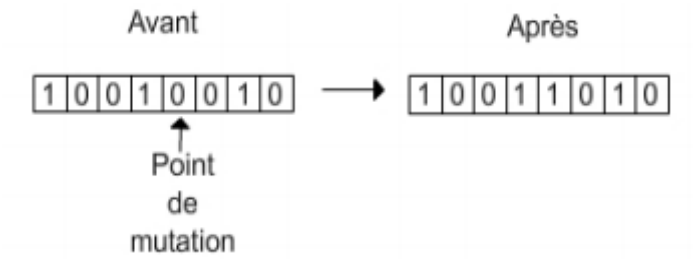


Figure. 3.8- Mutation de bits

3.4.1.2.7 La convergence

L'évolution se montre toujours comme étant un processus sans fin. Les algorithmes génétiques ne peuvent jamais s'arrêter sur une solution optimale globale du problème. Donc l'évolution doit s'arrêter à un certain point déterminé en fonction de certains critères. Pour cela, il existe plusieurs façons pour décider la contrainte d'arrêt de l'algorithme, et qui est déterminée par le concepteur selon les besoins du problème. Par exemple, la contrainte la plus simple consiste à arrêter le processus après un certain nombre d'itérations. Une autre méthode consiste à laisser les itérations jusqu'à l'observation de certains changements, où jusqu'à ce que la population se stabilise (les individus deviennent tous identiques). Donc, le choix de cette condition est souvent lié au problème.

3.4.1.3 Pseudo Code de l' AG

Début

Initialisation *de la population.*

Evaluation *des fonctions objectives.*

Pour *i = 0 à MaxIter faire*

Sélection

Croisement

Mutation

***Evaluation** des fonctions objectives*

Fin pour

Fin

3.4.1.4 Organigramme de l'AG

Les algorithmes génétiques représentent une des puissantes méthodes d'optimisation utilisées dans un grand nombre de domaines d'application, lorsqu'il ne s'agit pas de trouver la solution exacte. Pour conclure, nous illustrons les différentes étapes des algorithmes génétiques dans la figure suivante:

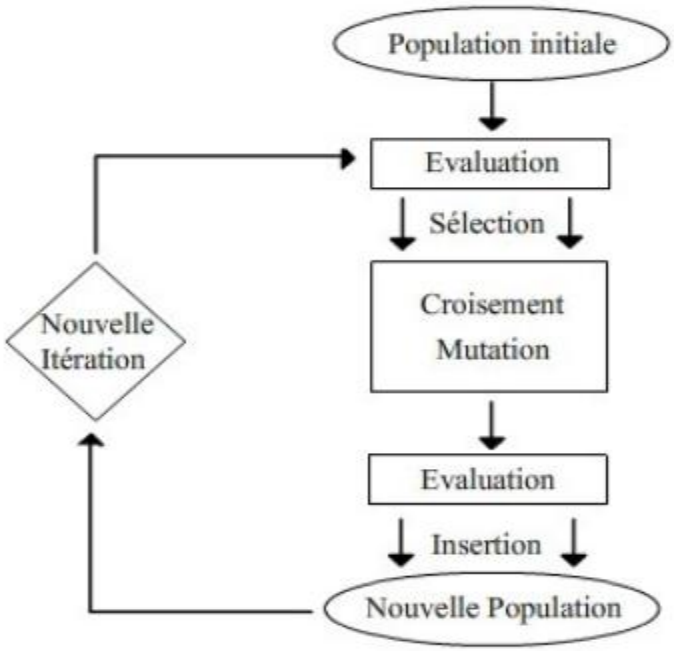


Figure. 3.9- Le processus d'exécution d'un algorithme génétique

3.4.2 L'algorithme de Branch & Bound

La plupart des problèmes, en particulier les problèmes d'optimisation, l'ensemble de leur solutions est fini (en tous les cas, il est dénombrable). Il est donc possible, en principe, d'énumérer toutes ces solutions, et ensuite de prendre celle qui nous arrange. L'inconvénient majeur de cette approche est le nombre prohibitif du nombre de solutions : il n'est guère évident d'effectuer cette énumération. La méthode de Branch & Bound (procédure par

évaluation et séparation progressive) consiste à énumérer ces solutions d'une manière intelligente en ce sens que, en utilisant certaines propriétés du problème en question, cette technique arrive à éliminer des solutions partielles qui ne mènent pas à celle que l'on recherche. De ce fait, on arrive souvent à obtenir la solution recherchée en des temps raisonnables. Bien entendu, dans le pire des cas, on retombe toujours sur l'élimination explicite de toutes les solutions du problème. Pour ce faire, cette méthode se dote d'une fonction qui permet de mettre une borne sur certaines solutions pour soit les exclure soit les maintenir comme des solutions potentielles. La performance d'une méthode de Branch & Bound dépend, entre autres, de la qualité de cette fonction (de sa capacité d'exclure des solutions partielles tôt).

3.4.2.1 Définition et Terminologies

Branch & Bound, est une méthode générique de résolution de problèmes d'optimisation, plus Particulièrement d'optimisation combinatoire ou discrète. C'est une méthode d'énumération implicite :

Toutes les solutions possibles du problème peuvent être énumérées mais, l'analyse des propriétés du Problème permet d'éviter l'énumération de larges classes de mauvaises solutions. Dans un bon algorithme par séparation et évaluation, seules les solutions potentiellement bonnes sont donc énumérées [13].

3.4.2.2 Principe de « B&B »

Soit S un ensemble fini mais de "grande" cardinalités qu'on appelle ensemble (ou espace) des solutions réalisables.

On dispose d'une fonction f qui, pour toute solution réalisable x de S, renvoie un coût f (x). Le but du problème est de trouver la solution réalisable x de coût minimal. D'un point de vue purement existentiel, le problème est trivial : une telle solution x existe bien car l'ensemble S est fini. En revanche, l'approche effective du problème se confronte à deux difficultés [47].

- La première est qu'il n'existe pas forcément un algorithme simple pour énumérer les éléments de S.

- La seconde est que le nombre de solutions réalisables est très grand, ce qui signifie que le temps d'énumération de toutes les solutions est prohibitif (la complexité algorithmique est généralement exponentielle).

Dans les méthodes par séparation et évaluation, la séparation permet d'obtenir une méthode

Générique pour énumérer toutes les solutions tandis que l'évaluation évite l'énumération systématique de toutes les solutions.

Pour lancer l'algorithme on doit avoir en possession :

1. Un moyen de calcul d'une borne inférieure d'une solution partielle.
2. Une stratégie de subdiviser l'espace de recherche pour créer des espaces de recherche de plus en plus petits.
3. Un moyen de calcul d'une borne supérieure pour au moins une solution.

3.4.2.3 Parcours de l'arbre

Par convenance, on représente l'exécution de la méthode de *B&B* à travers une arborescence. L'algorithme utilise le principe de la profondeur d'abord, en parcourant l'arbre de l'espace des solutions. La méthode commence par considérer le problème de départ avec son ensemble de solutions, appelé la racine. Des procédures de bornes inférieures et supérieures sont appliquées à la racine [38]. Si ces deux bornes sont égales, alors une solution optimale est trouvée, et on s'arrête là. Sinon, L'ensemble des solutions est divisé en deux ou plusieurs sous problèmes Séparation, devenait ainsi des enfants de la racine.

La méthode est ensuite appliquée récursivement à ces sous problèmes, engendrant ainsi une arborescence. Si une solution optimale est trouvée pour un sous problème, elle est réalisable, mais pas nécessairement optimale, pour le problème départ.

Comme elle est réalisable, elle peut être utilisée pour éliminer toute sa descendance : si la borne inférieure d'un nœud dépasse la valeur d'une solution déjà connue, alors on peut affirmer que la solution optimale globale ne peut être contenue dans le sous-ensemble de solution représenté par ce nœud. La recherche continue jusqu'à ce que tous les nœuds soient explorés ou éliminés.

3.4.2.4 Pseudo code de l'algorithme Branch & Bound

Initialisation : miel-sol=∝; Bi(p0) := f(p0) ; p := {(p0,Bi(p0))}
Tanque p = 0 faire
Sélection : sélectionner un nœud p de P ; p := p/{p}
Séparation : Décomposer p en p1, p2, p3, ...pk ;
Pour i := 1 à k faire
Évaluation Bi(pi) := f(pi) ;
Si Bi(pi) == f(x), x réalisable et (f(x) < meil-sol) alors

meil-sol :=f(x) ;
Solution=x ;
Sinon si Bi(pi) > meil-sol Alors
Élaguer : Élaguer pi ;
Sinon p := p ∪ {(pi,Bi(pi))}
Fin Si ;
Fin Si ;
Fin Faire ;
Fin Tan que ;
Résultat : Solution Optimale := solution ; Valeur Optimale := meil-sol ;

3.4.2.5 Organigramme de Branch & Bound

La figure 3.10 représente un diagramme de fonctionnement de l'algorithme de Branch & Bound :

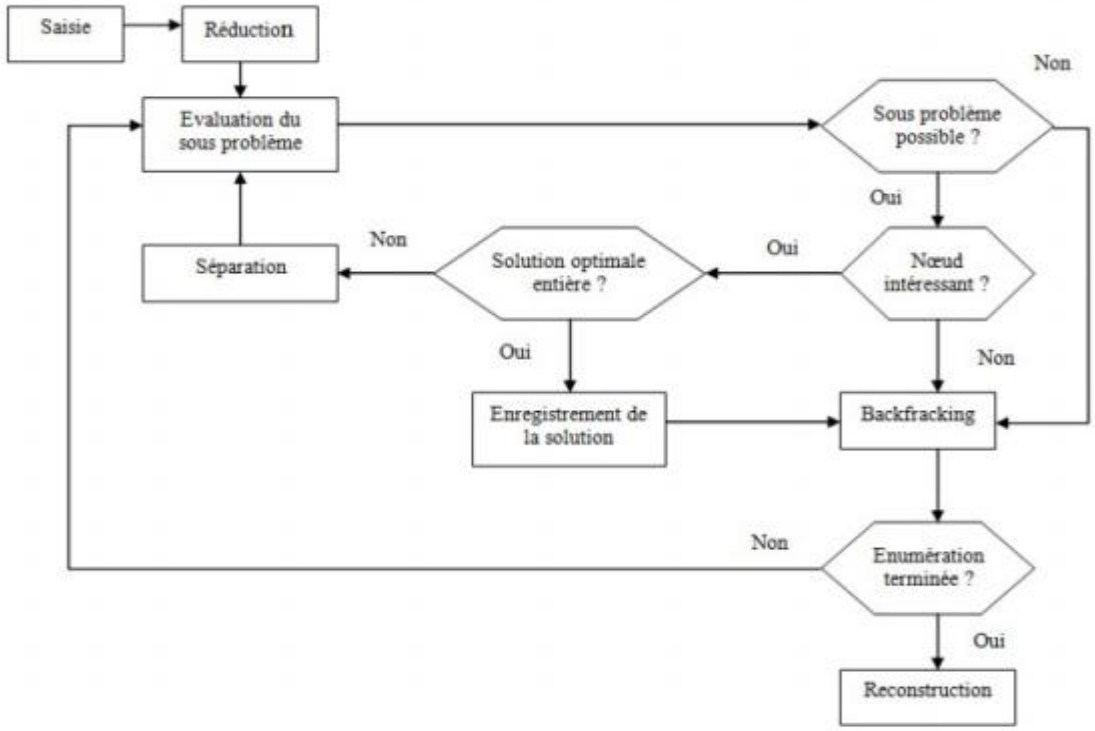


Figure 3.10- Organigramme de Branch & Bound

3.5. CONCLUSION

Les méthodes d'optimisation sont extrêmement nombreuses, elles sont basées sur des principes totalement différents, chacune explore et exploite l'espace de recherche selon des techniques qui lui sont propres. Pour notre étude, nous avons retenu les Algorithmes génétiques et branch &bound parce qu'elles sont extrêmement performantes dans de nombreux domaines. Ce sont des méthodes très efficaces lorsqu'il s'agit d'exploiter une zone de l'espace de recherche. D'autre part, elles s'adaptent assez bien au problème posé. Les concepts cités dans ce chapitre sont nécessaires à la compréhension de la démarche qu'on a apportée pour résoudre le problème de mapping dans les MPSoC et qui fera l'objet du chapitre suivant.

Chapitre IV

« Contribution »

4.1 INTRODUCTION

Le mapping ou l'association « application/matérielle », constitue l'étape la plus importante et la plus critique dans le flot de conception et de développement des SoC (System on chip). Elle participe d'une façon directe dans l'atteinte des objectifs voulus lors de la spécification des SOC, comme la minimisation de la consommation d'énergie et la maximisation des temps de performance ou équilibrage de charges des mémoires.

Le but de notre travail est de concevoir un algorithme hybridé basé sur le comportement des génétiques et la méthode B&B qui permet la résolution du problème d'affectation et d'ordonnancement d'une application sur une architecture cible tout en essayant d'atteindre des objectifs et vérifier des contraintes. L'architecture adoptée est MPSoC, c'est à dire des microprocesseurs hétérogènes utilisant des réseaux d'interconnexion sur la puce. L'application est Composée de plusieurs tâches communicantes par des messages.

Atteindre les objectifs et vérifier les contraintes revient à résoudre un problème d'assignation, d'affectation et d'ordonnancement ***AAS*** (Assignation Affectation and Scheduling). La majorité des travaux de résolution de ce problème ne prennent en considération que la topologie des communications en bus ce qui présente une incompatibilité avec notre architecture (Lahua [48], Prestana [49] et Benini [50]). Sinon d'autres travaux présentent des méthodes d'optimisation mono-objectif (Murale [51], Lei [52], Hu [53]), C'est-à-dire que ces méthodes ne permettent l'obtention que d'un seul objectif : temps d'exécution, consommation d'énergie, surface, etc. ce qui cause un problème dans notre cas où on est face à deux objectifs. Donc notre approche apporte du nouveau aux anciens travaux en résolvant le problème d'AAS sur architecture MPSoC avec une méthode d'optimisation multi-objectifs.

AAS appelé aussi "Mapping and Scheduling" est une étape qui suit la spécialisation du NoC, où on implémente l'application sur l'architecture sélectionnée. Ceci consiste à assigner et ordonnancer les tâches et communications de l'application sur les ressources d'architecture cible de telle façon que les objectifs spécifiés soient atteints.

Dans ce chapitre, on va proposer notre stratégie hybridée sous le mapping hiérarchique constitué d'une part de stratégie de mapping pour le global irrégulier et d'une stratégie pour le local régulier. D'où notre démarche proposant une solution globale hybride basée sur les algorithmes génétiques, et la méthode B&B. Ça consiste à proposer et mettre En œuvre une méthode évolutionniste et une autre exact pour solutionner ce genre de problème, et trouver

le meilleur placement des taches selon plusieurs objectifs. D‘autre part, pour optimiser le mapping des communications on a utilisé une méthode hybridée basée sur Dijkstra pour le calcul du plus court chemin.

4.2 PROBLEMATIQUE DE MAPPING

Le problème à résoudre dans sa globalité est un problème d’assignation, d’affectation et d’ordonnancement «AAS». Ceci consiste principalement à assigner et ordonnancer les tâches et les communications de l'application sur les ressources d'architecture cible de telle façon que les objectifs spécifiés soient atteints.

On sait bien que l'énergie ou plutôt l'autonomie en énergie est une notion très importante dans un système embarqué, chose qui demande des efforts de conception plus élevés puisqu'on sait bien que le fonctionnement de notre système dépend de la durée de vie de la batterie sans négliger l'effet indésirable de Joule où la hausse de température des circuits qui peut mettre en péril l'équilibre de notre système.

L’autre performance visant la diminution de la vitesse d'exécution qui est très importante dans le systèmes temps réel peut entrainer l’augmentation de la consommation d’énergie. D’où, trouver un mapping qui prend en considération ces deux objectifs revient à trouver un compromis entre eux.

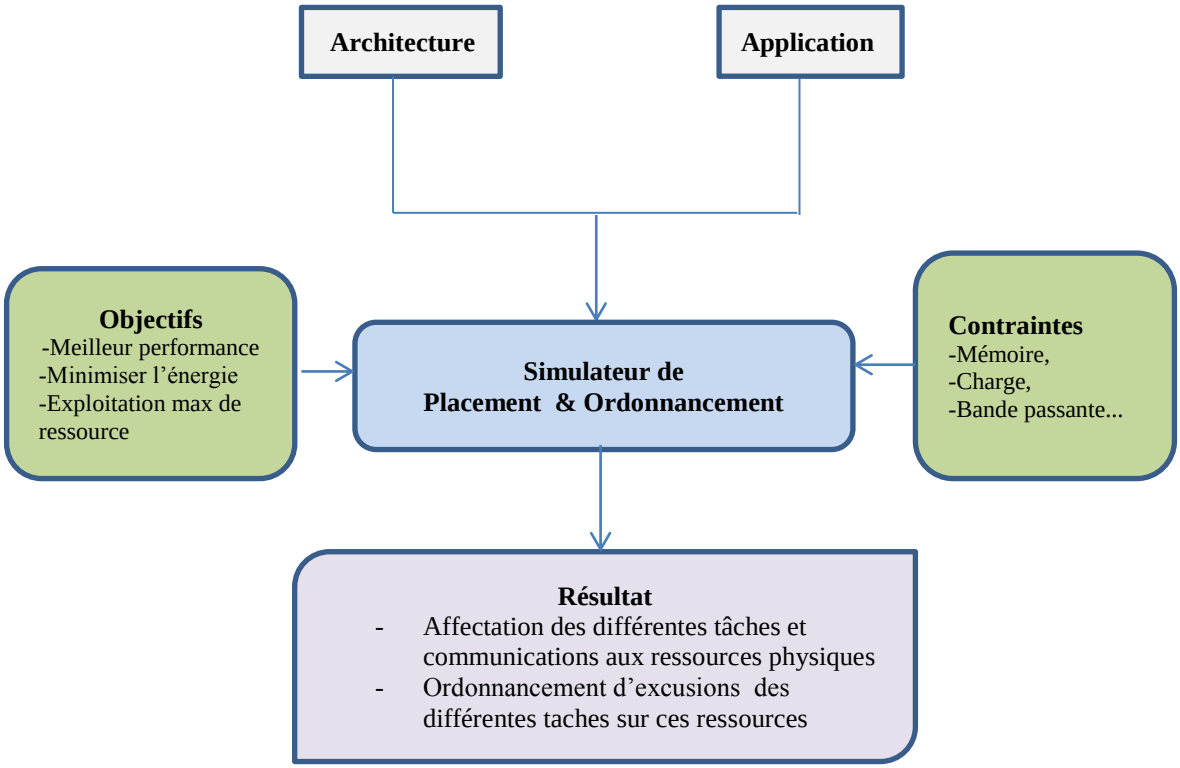


Figure 4.1 -Description du problème de mapping

4.3 DEFINITION ET CONTRAINTES

Les communications entre les composantes d'une application sont représentées par un graphe d'application orienté (cf. figure 4.2).

4.3.1 Définition 1 (Modèle d'application)

Le graphe d'application TG (Task Graph) est un graphe orienté G(T, E) où chaque sommet $t_i \in V$ représente un module (tâche) de l'application et l'arc orienté $(t_i ; t_j)$ noté $e_{ij} \in E$ représente les communications entre les modules t_i et t_j. Le poids de l'arc e_{ij} noté par Q_{ij} Représente le volume des communications de t_i à t_j.

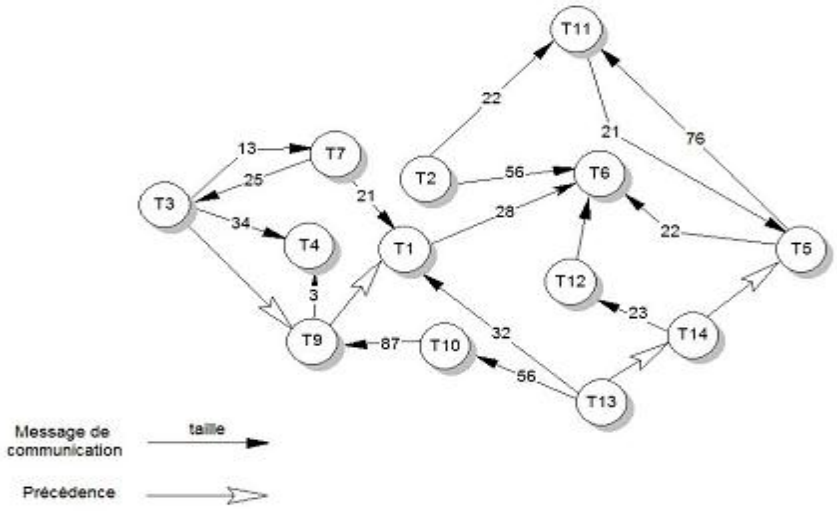


Figure 4.2- Modèle d'application

La connectivité et la bande passante des liens physiques, latence et l'énergie sont représentées par le graphe d`architecture.

4.3.2 Définition 2 (Modèle d'architecture)

Le graphe d'architecture NT (NoC Topology graph) est un graphe orienté P (S; F) où chaque Sommet $s_i \in S$ représente un nœud de la topologie et l'arc (s_i; s_j) noté par $f_i \in F$ représente un Lien physique direct entre les éléments de s_i et s_j de l'architecture. Le poids de l'arc f_{ij} noté par bw_{ij} représente la bande passante, latence et l'énergie disponibles à travers le lien physique f_{ij} (cf. Figure 4.3).

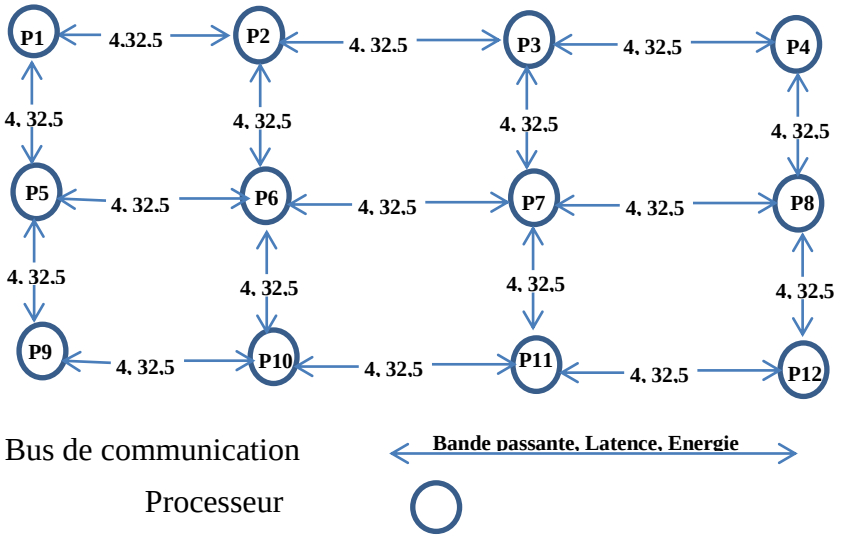


Figure 4.3- Modèle architecture

4.3.3 Définition 3 (Mapping)

Le mapping du graphe d'application G(T;E) sur le graphe d'architecture P (S; F) est illustré sur la figure 4.4 ci-dessous :

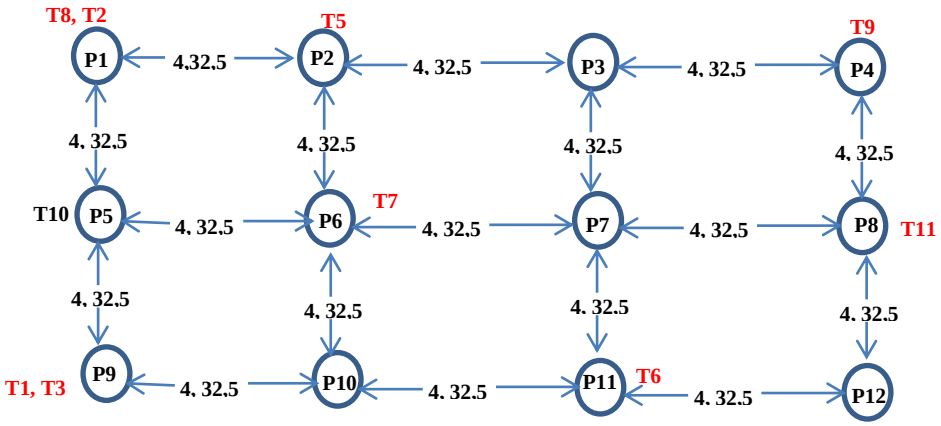


Figure 4.4- Mapping TG sur NT

4.3.4 Définition 4(Contraintes)

4.3.4.1 Mémoire : La mémoire d'exécution requise par une tâche doit être inférieure ou égale à celle disponible dans le SoC auquel elle a été affectée par le placement.

4.3.4.2 Bande passante : Dans le monde numérique la bande passante, en terme plus simple, c'est la capacité de charge d'une transmission de données d'un lien, mais la largeur de bande est mesurée en termes de taille de données max par unité de temps.

Par cette définition, nous pouvons conclure que la bande passante d'un lien est la capacité maximale de données dans le lien que peut transporter par unité de temps d'une source à la destination.

4.3.4.3 Latence : la latence désigne le délai entre le moment où une information est envoyée et celui où elle est reçue. De façon plus générale, la latence peut aussi désigner l'intervalle entre la fin d'un événement et le début de la réaction à celui-ci, par exemple :

- le délai entre une requête à un médium de stockage de données et le début du transfert de l'information demandée. Les latences les plus importantes sont causées par des déplacements

Mécaniques, par exemple sur un disque dur par le positionnement de la tête de lecture et la rotation du disque.

- le délai entre une requête d'un client et la réponse du serveur.

- le délai de l'entrée analogique d'une carte son, c'est-à-dire le temps nécessaire à la conversion du signal analogique en signal numérique.

- le temps de réponse d'un écran à cristaux liquides, c'est-à-dire le temps que met la lumière émise par un pixel à réagir correctement à un changement de valeur électrique.

Ensemble, la latence et la bande passante définissent la vitesse et la capacité d'un lien réseau.

4.3.4.4 Equilibrage de charge : Principe constituant à répartir aussi équitablement que possible la charge de travail entre plusieurs éléments d'exécution de l'architecture cible. Son but est de garantir qu'aucun processeur ne reste inactif quand il y a d'autres processeurs qui sont lourdement chargés dans le système hétérogène.

4.4 FORMULATION MATHEMATIQUE

Pour un TG chaque nœud ou sommet représente une tâche avec ses caractéristiques ou propriétés. Soit T = $\{t_1,t_2,\ldots\ldots\ldots t_n\}$ l'ensemble des tâches représentées par l'ensemble des sommets T_i dans TG. S = $\{S_1,S_2,\ldots\ldots S_p\}$ l'ensemble des processeurs représentés par l'ensemble des nœuds S_i dans NT. On considère que chaque processeur S peut s'exécuter à des modes différents (mode rapide, mode économique…etc.) : m= m1, m2…mn.

- Mettre les processeurs dans plusieurs modes d'exécution qui nous donnent plus de diversité dans L'optimisation de placement.
- Un modèle d'architecture cible.
- Des contraintes de performance et d'énergie.
- Des fonctions objectifs à optimiser.

On utilise des variables comme suit :

D : Le temps total d'exécution de notre application.

$D_{Exec}^{m}(ij)$: La durée d'exécution d'une tâche i placée sur un processeur j s'exécutant au mode d'exécution m (sans prendre en compte les communications).

D_i *:* le temps total d'exécution d'une tâche i en prenant en considération les communications .

Taille(i) : Le nombre de cycles nécessaires d'une tache pour s'exécuter sur un processeur.
Freq : la fréquence d'horloge d'un processeur au mode m.

$D_{Com}(i)$: Durée de communication prise le transfert de données (résultat) de la tache (i) ver ses successeurs.

$E^m_{Exec}(ij)$: L'énergie d'exécution issue à l'exécution de tache i sur le processeur j au mode d'exécution m.

e_j : Energie d'exécution unitaire.

E_{com} (i) : La consommation d'énergie due à la communication pour la tâche (i)

E : l'énergie totale.

4.4.1 Durée (Exécution et communication)

$$D^m_{Exec}(ij) = Taille\ (i) / Freq^m_j ; \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.1)$$

- Soit une tache (T_i) place sur un processeur (P_j), calculer la communication $D_{com}(i)$ revient à chercher le plus court chemin entre (P_j) et tout autre processeur (P_L), là où les tâches successeurs sont placées.

- Soit $|P_j , P_L|$: le plus court chemin entre P_j et P_L alors :

$$D_{Com}(i) = Max_{k \in Succ(i)} \left(\frac{Q_{ik}}{Min\ (Bd\ |P_j,P_L|)} \cdot \sum_{i \in |P_j,P_L|} Latence(P_j , P_{j+1}) \right) ; \ldots\ldots\ldots\ldots (4.2)$$

Tel que :

Q_{ik} : Quantité de données échangées entre les taches : T_i et T_k

$Min\ (Bd\ |P_j,P_L|)$: La bande passante minimale dans le chemin de P_j à P_L

$$Donc\ \ Di = D_{Exec\ (i)} + D_{Com\ (i)}; \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.3)$$

Après l'ordonnancement:

$$D = Max\ (D_i) ; \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.4)$$

Tel que i : le nombre de tâches.

De (4), on déduit la relation suivante :

$$D \leq Deadline ; \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.5)$$

4.4.2 L'énergie (Exécution et communication)

Energie d'exécution :

La consommation d'une tâche i affectée au processeur j au mode m :

$$E^m_{Exec}(ij) = \text{Taille (i)} * e^m_j ; \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.6)$$

Energie de communication :

La consommation d'énergie due à la communication :

$$E_{com}(i)= \sum Qi*e(Pk,Pl);\dots\dots(4.7)$$

La consommation d'énergie totale est donnée par la formule suivante en prenant en considération la consommation des processeurs et des liens :

$$E= \sum_{i=1}^{Nbre\ Tache}(E_{Exec(i)} + E_{Com(i)}) \ ;\dots\dots(4.8)$$

4.5 LES METHODES DE RESOLUTION CHOISIES

4.5.1 Graphe potentiel Taches

La méthode Potentiel-Tâche est une application de la théorie des graphes à l'optimisation de l'ordonnancement de tâches. Elle s'appuie sur l'utilisation d'un graphe où les tâches sont représentées par des sommets et les relations de précédence par des arcs values. On a utilisé cette méthode pour dégager les classes d'exécution, pour les nœuds irréguliers et composés de chaque niveau de la hiérarchie, qui seront affectés à des processeurs du même niveau.

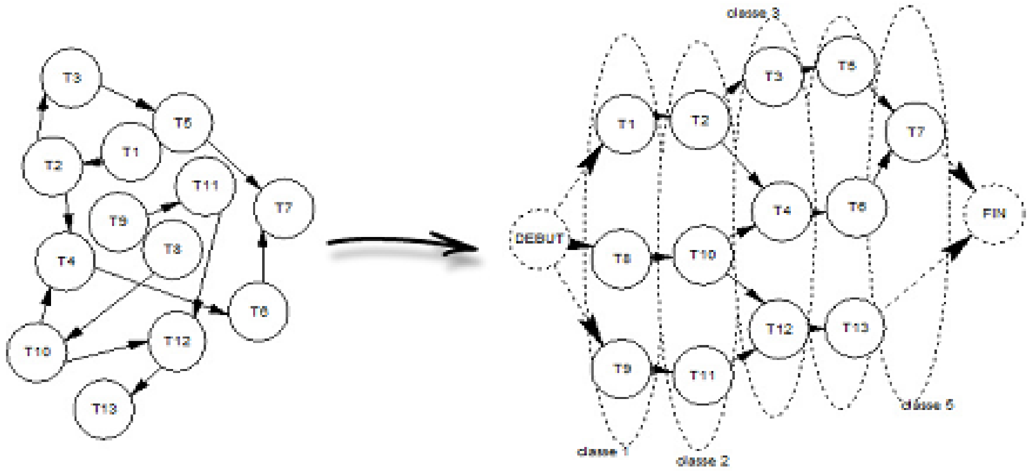


Figure 4.5- Exemple de représentation des niveaux du graphe

4.5.1 Dijkstra Multi-Objectifs

Sert à résoudre le problème du plus court chemin entre deux sommets d'un graphe connexe dont le poids lié aux arêtes est positif ou nul.

L'algorithme porte le nom de son inventeur, l'informaticien néerlandais Edsger Dijkstra et a été publié en 1959.

On a modifié l'algorithme original pour qu'il puisse faire la comparaison sur deux critères: la latence et l'énergie consommée avec priorité à la latence. Si l'algorithme trouve deux chemins vers la même destination, à la fin de l'algorithme il compare les diffèrents chemins en se basant sur la latence puis sur l'énergie.

On a donné priorité au temps car l'énergie consommée par un bus au moment du Transfer est Beaucoup moins importante que l'énergie consommée par le processeur pendant l'attente.

4.6 APPROCHE DE RESOLUTION

Pour la résolution de notre problème, on propose de faire une hybridation de trois méthodes, deux méthodes pour calculer le placement des taches irrégulieres et régulières et l'autre pour trouver le placement optimal des communications sur les processeurs de l'architecture.

Ce placement a une particularité, dans ce contexte, car il ne s'agit pas uniquement d'un placement classique qu'on a l'habitude de rencontrer dans les systèmes parallèles mais il englobe trois phases: Assignation, Affectation et Scheduling (AAS) et que l'on peut caractériser par :

- Le placement des calculs (taches) qui permet de trouver sur quel processeur doit s'exécuter quelle tâche et à quel moment.
- Le placement de données qui doit déterminer un accès efficace et rapide aux données avec si c'est possible des chemins d'accès le plus court possible en prenant en considération les caractéristiques de la topologie cible.

Ce type de placement ou mapping dans sa globalité est connu comme étant un placement GILR (Globally Irregular Locally Regular), Nous nous intéressons dans notre travail à sa partie irréguliere (GI) et à sa parie régulière (LR), donc notre objectif consiste à chercher le placement des tâches irrégulieres et les tâches répétitives d'une application intensive sur les ressources d'une architecture MPSoC.

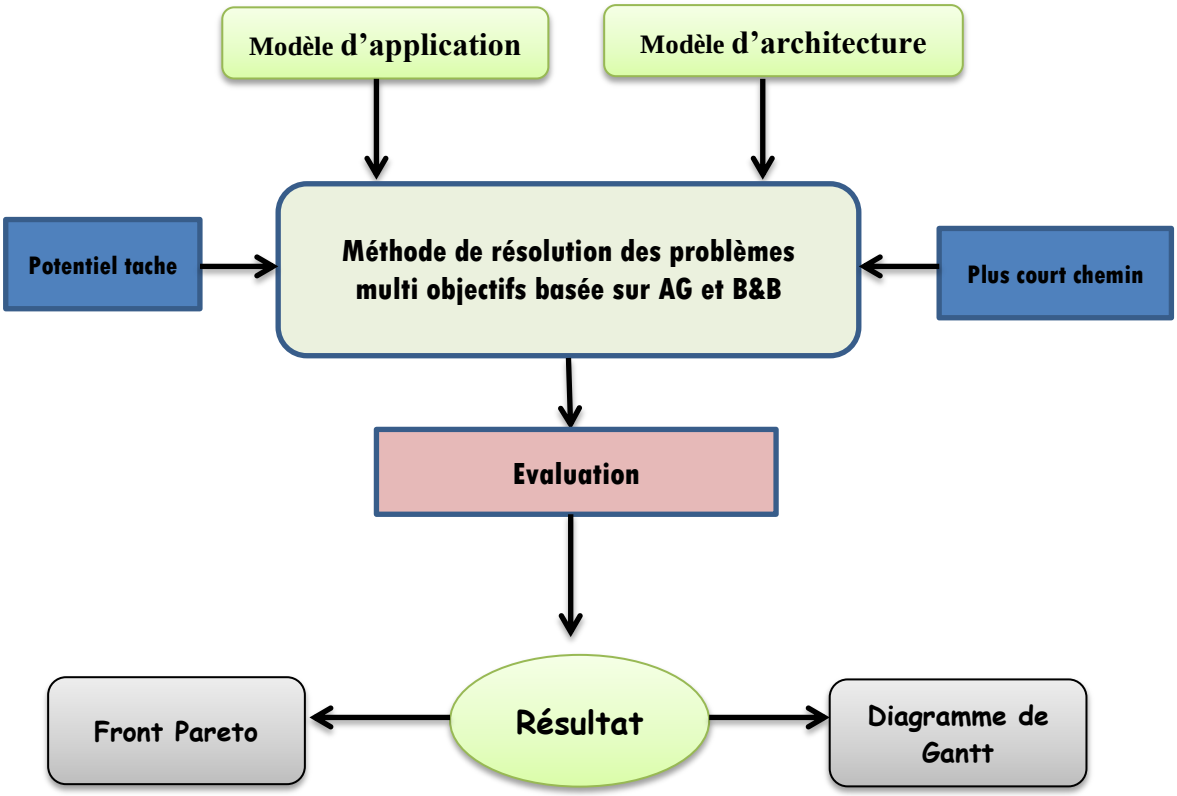


Figure 4.6- Diagramme illustrant l'approche de résolution

4.7 ALGORITHME DE PLACEMENT ET D'ORDONNACEMENT

La démarche suivie pour résoudre le problème de mapping consiste à faire l'affectation d'un HTG (Hierarchical Task Graph) à un HNT (Hierarchi-cal Network Graph) avec des niveaux identiques. Dans notre approche on s'impose certaines hypothèses :

- Les deux graphes ont les mêmes niveaux.
- Pour chaque type de nœud d'application, on a au moins un équivalant dans l'architecture dans le même niveau.
- On considère aussi qu'à tous les niveaux, nous avons le même nombre de nœuds composés.
- Le dernier niveau contient uniquement des nœuds élémentaires représentants des tâches dans le HTG ou processeurs dans le HNT.

L'affectation des tâches aux processeurs et placement de communication se fait comme suit :

A partir du niveau n − 1 :

- on construit les ensembles suivants :

– A partir de HTG :

S_{TC}^{N-1} = {ensemble des tâches composées.}

S_{TR}^{N-1} = {ensemble des tâches répétitifs.}

S_{TIr}^{N-1} = {ensemble des tâches Irrégulières.}

– A partir de HNT :

S_{PC}^{N-1} = {ensemble des Processeurs composées.}

S_{PR}^{N-1} = {ensemble des Processeurs répétitifs.}

S_{PIr}^{N-1} = {ensemble des Processeurs Irréguliers.}

- On trie par ordre décroissant :

S_{TIr}^{N-1} : selon la taille.

S_{TC}^{N-1} : selon la taille.

S_{TR}^{N-1} : selon le nombre des fils.

- S_{PC}^{N-1}, S_{PIr}^{N-1} : Pour grouper les processeurs, les plus similaires (vitesse, énergie),et on trie S_{PR}^{N-1} selon le nombre de fils dans un ordre décroissant.
- Mapping :

- Taches composées : Tan que $S_{TC}^{N-1} \neq 0$ *Faire*

$Liste_T \leftarrow$ *Top de la liste* S_{TC}^{N-1}

$Liste_P \leftarrow$ *Top de la liste* S_{PC}^{N-1}

Mapping($Liste_T$ *,* $Liste_P$ *)*

$S_{TC}^{N-1} = S_{TC}^{N-1} - Liste_T$

$S_{PC}^{N-1} = S_{PC}^{N-1} - Liste_P$

Fin

- Taches Irréguliers : Tan que $S_{TIr}^{N-1} \neq 0$ *Faire*

$Liste_T \leftarrow$ *Top de la liste* S_{TIr}^{N-1}

$Liste_P \leftarrow$ *Top de la liste* S_{PIr}^{N-1}

$Mapping_{Ir}(Liste_T , Liste_P)$

$S_{TIr}^{N-1} = S_{TIr}^{N-1} - Liste_T$

$S_{PIr}^{N-1} = S_{PIr}^{N-1} - Liste_P$

Fin

- Tâches Répétitives : <u>Tan que</u> $S_{TR}^{N-1} \neq 0$ *Faire*

$Liste_T \leftarrow$*Top de la liste* S_{TR}^{N-1}
$Liste_P \leftarrow$*Le plus similaire (*$Liste_T$*) dans* S_{PR}^{N-1}
$Mapping_R(Liste_T , Liste_P)$
$S_{TR}^{N-1} = S_{TR}^{N-1} - Liste_T$
<u>Fin</u>

- Pour chaque élément de la liste $Mapping_{Ir}$ appeler **AG**.
- Pour chaque élément de la Liste $Mapping_R$ appeler **B&B**.
- Pour chaque tâche de S_{TC}^{N-1} :

– Etablir la liste scheduling des ces fils.

– Calculer le plus court chemin pour tout message échangé entre ces fils.

– Combiner les fronts Pareto générés par l'application de B&B ou AG sur ces fils.

- Si Racine alors Arrêter sinon Remonter dans les niveaux.

4.8 ALGORITHME GENETIQUE POUR LE PLACEMENT IRRÉGULIER

Dans notre travail nous avons opté pour les algorithmes génétiques comme moyen de résolution du problème de placement des tâches irrégulières. Notre choix pour cette technique est motivé par les arguments suivants :

- Les Algorithmes génétiques sont très efficaces à condition que les paramètres soient bien choisis, ainsi que les opérateurs génétiques (sélection, mutation et croisement) sont appropriés à l'exploration de l'espace de recherche,
- Les Algorithmes génétiques permettent de prendre en considération le multi-objectif. Avec plus de précision, un algorithme évolutionnaire front Pareto (PAES Pareto Archived Evolution Strategy) qui maintient un ensemble externe pour préserver les solutions non dominées rencontrées auparavant, ce qui signifie que notre solution repose sur deux populations :

  1. Archive : Population conservant les meilleures solutions.

  2. Evolution : Population conservant la diversité.

Rappelons que l'organigramme général d'un algorithme génétique est celui représenté à la figure 4.7

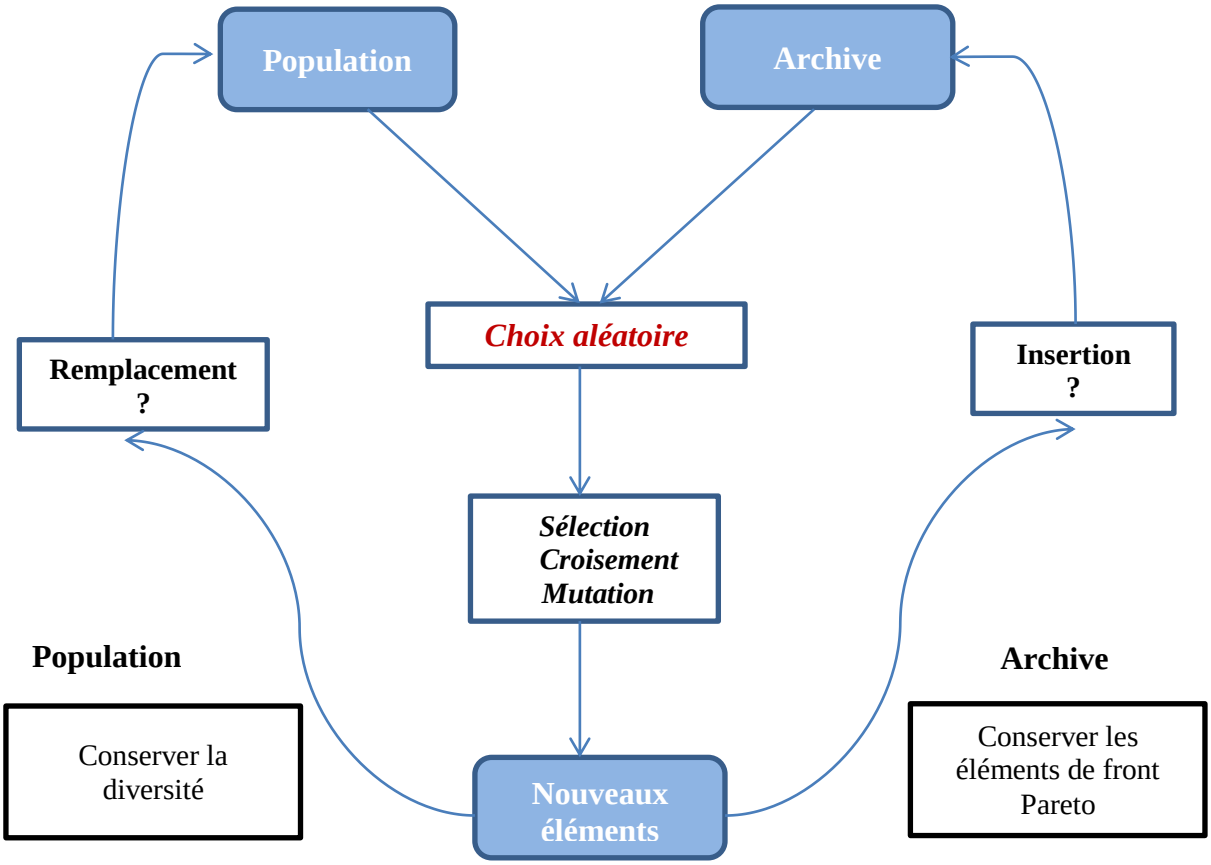


Figure 4.7 - Organigramme général d'un algorithme génétique

L'algorithme utilisé est simple et inspiré des stratégies d'évolution, les deux populations sont d'une part une population classique, composée d'individus évoluant et d'autre part une population archive regroupant les individus définissant l'estimation du front de Pareto à un instant donné. Ces deux populations sont combinées au cours des itérations de l'algorithme génétique. La première population a pour objectif l'exploration de l'espace des paramètres et la seconde l'exploitation des connaissances acquises ainsi que la convergence de l'algorithme.

4.8.1 Pseudo code de notre algorithme d'optimisation

Début

Initialisation de la Population I.

Initialisation de l'Archive A : éléments non-dominés de I.

Répéter

F: Fusion Population-Archive.

(I1, I2) Selection aléatoire (F).

(I3, I4) Croisement (I1, I2).

Mutation (I3).

Mutation (I4).

Evaluation (I3).

Évaluation (I4).

Si I3 domine I4 ou inversement Alors

I5: dominant (I3, I4).

Sinon

Conserver(I3, I4).

Fin Si

Mise à jour éventuelle de A / I5 ou (I3, I4).

Mise à jour éventuelle de I / I5 ou (I3, I4).

Jusqu'à nombre de générations.

Fin

4.8.2 Paramétrés de l'algorithme génétique

4.8.2.1 Nombre de gènes

On a une configuration de ***P*** processeur s'exécutant en ***M*** modes et de ***T*** tâches. On prend la taille d'un chromosome en gènes : ***Ch = P* M * T***

4.8.2.2 Le codage

L'optimisation de placement et d'ordonnancement sont exécutées en exploitant les algorithmes évolutionnaires. Par conséquent, le placement et l'ordonnancement sont exprimés comme un chromosome. Ce dernier est une représentation de la solution du problème qui dans notre cas décrit le mapping. Chaque élément d`exécution dans le mesh lui est associé un gène qui encode la tâche qui lui est affectée. Si on a un NoC en Mesh de S processeurs s'exécutants en M modes et une application avec N tâches alors le chromosome est formé de S gènes.

On a choisi le codage binaire pour les gènes, le 1 dans un gène signifie que la tâche Ti

est placée dans le processeur P_k au mode Mj, 0 autrement. Codage binaire du chromosome :

nbrprocesseur * nbrMode * nbrTâche

Pour chaque processeur, on a un nombre de mode :

- Donc une tâche peut être affectée à un processeur précis dans un mode précis.
- Une tâche ne peut être affectée qu'à un seul processeur à la fois dans un mode.

On prend la sémantique qu'un chromosome est partitionné en S parties qui représentent les processeurs et chaque partie et partitionnée en 2 modes et T parties qui représentent les tâches placées dans ce processeur.

Comme exemple prenant 2 processeurs s'exécutant en deux modes et 3 tâches, alors le chromosome 001010 000100 représente le mapping de 3 tâches : les tâches 2 et 3 sont affectées au processeur 1 mais chacune dans un mode d'exécution (3 au mode 1 et 2 au mode 2), la tâche 1 affectée au Processeur 2 dans le mode 2, la figure 4.6 ci-dessous illustre mieux ce codage.

Figure 4.8- Codage utilisé

4.8.2.3 Détermination de la population initiale

Les gènes des chromosomes sont choisis de telle façon qu'une tâche soit affectée à un seul processeur à la fois dans un mode précis. On doit aussi savoir comment on peut augmenter la vitesse de recherche de cette étape ou autrement dit, comment diminuer le temps de convergence durant cette phase. Comme on le sait toute heuristique ou méthode itérative dépend du choix de ses paramètres et de la solution initiale. Pour notre algorithme on a essayé de placer les tâches qui communiquent le plus sur des processeurs voisins ou carrément sur le même processeur afin de s'assurer que dans la population initiale on a quelques bons individus

4.8.2.4 Choix des opérateurs génétiques

4.8.2.4.1 Sélection

Après la fusion des deux populations (Evolution et Archive) on sélectionne aléatoirement deux chromosomes pour garder la diversité dans notre recherche des solutions respectant les contraintes. Alors si la sélection tombe sur un chromosome de l'Archive on aura au moins une solution dominante au départ.

4.8.2.4.2 Croisement

Le croisement est le processus selon lequel les gènes de deux chaînes élues sont inter changés afin de générer une nouvelle population. Il existe plusieurs types de croisement : croisement en un seul lieu, croisement en deux lieux.

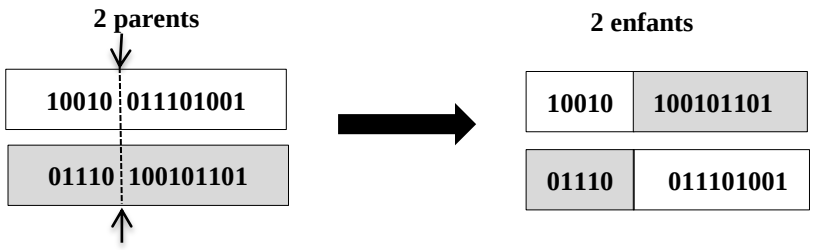


Figure 4.9- Croisement utilisé

4.8.2.4.3 Mutation

Pendant le processus de mutation, un seul gène est choisi aléatoirement parmi l'intervalle de chromosome avec une probabilité de *Pm*. La valeur du gène qu'on va muter est choisie au hasard .selon ce principe, il est possible qu'un gène ne subisse aucune mutation puisque sa valeur initiale peut être choisie de nouveau par le processus de randomisation (cf. figure 4.10).

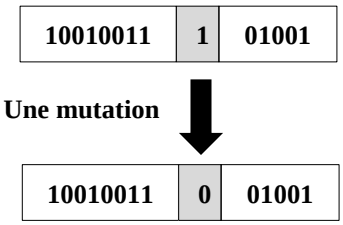


Figure 4.10- Mutation utilisée

4.8.2.5 Fonction d'évaluation

Un des éléments clés de l'algorithme génétique est la fonction d'évaluation (Fitness ou Fonction objective).

Le problème de placement des tâches d'une application sur une architecture MPSoC revient à minimiser la fonction objective. Il peut être formulé de la manière suivante :

Étant donné :

- Le graphe d'application (taille de la tâche par type de processeur, mémoire d'exécution requise par type de processeur, bande passante requise pour un message et taille du message).
- Le graphe d'architecture (Vitesse d'exécution par mode, consommation d'énergie par mode,

Équilibrage de charge (charge minimum et maximum), mémoire disponible dans un processeur, bande passante et latence des bus ainsi que l'énergie consommée due à la transmission).

On doit Minimiser Les coûts :

- Temps d'exécution de l'application.
- Énergie consommée due à l'exécution de cette application.

Donc par choix : On doit minimiser F(x) : f (Temps); f (Energie).

La détermination de la fonction Fitness (Fonction d'adaptation ou évaluation) passe par plusieurs étapes. Chaque fois qu'un chromosome est généré la fonction fitness de chacun de ces chromosomes doit être évaluée. Les étapes d'évaluation sont les suivantes :

On a en entrée :

-Un chromosome qui représente un placement des tâches de l'application sur l'architecture MPSoC cible.

Evaluation :

-Pour un chromosome, faire sortir les coûts de communication de tous ses messages en éliminant pour chaque message les chemins dont la bande passante est inférieure à celle requise par le message (en utilisant l'algorithme du plus court chemin), donc On calcule le temps d'exécution de chaque tâche selon le mode où a été affecté au sein du processeur; et puisque l'application (les tâches) est exécutée parallèlement donc pour calculer le temps global du traitement on utilise la *méthode du potentiel tâche* qu'on a parlé auparavant.

Concernant l'énergie consommée globale c'est l'addition des énergies consommées des différentes tâches placées sur les processeurs dans un mode particulier. Comme nous pouvons le constater, la phase d'évaluation comporte plusieurs étapes et de longs calculs. Remarquer que l'évaluation doit être faite pour chaque chromosome; si une population comporte 100 chromosomes, nous devrons réaliser 100 évaluations pour une génération. Si nous décidons de produire 40 générations, alors notre système aura calculé 4000 différents coûts, rang de matrice et fitness.

4.8.2.6 Mise à jour de la population d'évolution et d'archive

L'archive est une population qui ne conserve que les solutions optimales, La mise à jour de cette solution est faite ainsi :

1. Initialement; on a l'ensemble des solutions initialement générées, représentées par des chromosomes.

2. Après le calcul des coûts pour chaque chromosome.

3. On évalue la dominance par rapport aux solutions existantes.

4. Cette évaluation se fait un par un et c'est ce qui fait l'avantage de notre méthode.

5. Si la solution à tester est dominante par rapport à une autre appartenant à l'archive alors :

- On supprime cette solution dominée.

- Si on atteint la fin de la taille de notre archive alors

(a) On insère cette solution dominante.

(b) Sinon(le cas où notre solution est non dominante) on l'a fait remplacer par une autre solution dans la population d'évolution.

4.9 METHODE B&B POUR LE PLACEMENT DES TACHES RÉPÉTITVES

Comme indiqué dans le titre de cette section, nous avons opté pour la méthode branch and bound comme moyen pour résoudre le problème de mapping des tâches répétitives, notre choix pour cette technique est justifié par rapport à d'autres méthodes du fait qu'elle permet de prendre en considération le multi- objectif et permet d'éliminer des solutions partielles qui ne mènent pas à ceux que l'on recherche en utilisant certaines propriétés du problème en question.

Les tâches répétitives ont la particularité de présenter les mêmes caractéristiques : nombre de cycles d'exécution et la taille des données traitées. En plus ces tâches s'appliquent sur des données de même dimension et structures, qu'on appelle motifs [5]. La figure ci-dessous représente l'organigramme général de l'algorithme qui est basé sur la méthode exacte B&B hybridée avec une autre méthode d'optimisation des communications analogue à celle utilisé dans le AG.

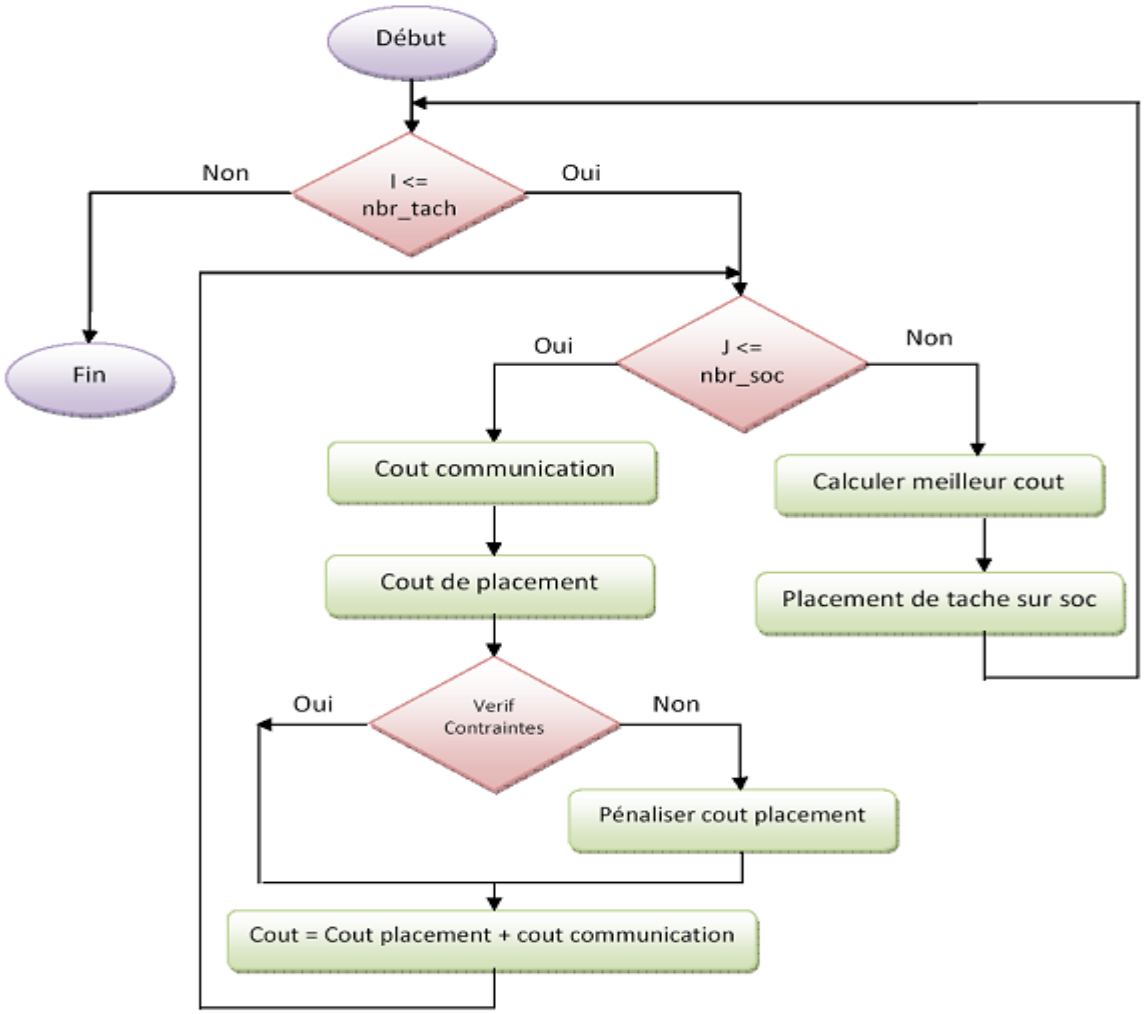


Figure 4.11- l'organigramme de l'algorithme basé sur la méthode exacte B&B

Pseudo Code de Branch and Bound

Début

Affecter $Task_1$ au RootPE

*Pour N= 2 à Max = 3*NbrPE faire*

Pour k=1 à NbrPE faire

Vérifier_Contraintes (k) ;

Pour m=1 à NbrMode

Calculer la durée de $Task_N$ affectée au processeur P_k^m ;

$D_{motif} = Max_{i \in motif}(D_i)$;

Si $D_{motif} \leq DL_{motif}$

Calculer l'énergie du motif : E motif

Couple = (D_{motif},E_{motif})
Si Couple est non dominé
Placement de $Task_N$ sur P_k^m
Fsi
Fsi
Fpour
Fpour
Fpour
Fin

4.10 GÉNÉRATION DE CHEMIN

Pour ajouter du plus à notre approche d'optimisation, nous avons choisi d'hybrider l'algorithme génétique et l'algorithme de B&B avec une autre méthode pour améliorer les performances. nous avons utilisé Dijkstra multi objectif comme une méthode nous donne le plus court chemin afin de résoudre les trois problèmes fondamentaux :

- Premièrement : Considérant notre topologie d'interconnexion MPSoC construisant une série de Processeurs triés en fonction de leur distance à l'égard de l'élément source.
- Deuxièmement : Compte tenu d'un ensemble de plusieurs courts chemins, le but est de trouver un chemin le plus court possible avec un minimum de poids (en charge communication) sur les arcs (les liens).
- Troisièmement : Pour équilibrer la charge de communication sur les liens dans un réseau donné, ce processus est appelé l'équilibrage de charge réseau. Nous allons expliquer ci-dessous.

L'équilibrage de charge réseau ou de communication, se fait par mise à jour de la charge de la

Communication (poids sur l'arc), d'une façon à éliminer les chemins qui ont une charge inférieure à celle requise par le message et ceci est fait pour chaque transmission d'un message au sein du réseau de communication (ou graphe).

Cette mise à jour graphique de l'interconnexion est utilisée à chaque appel de l'algorithme de

Dijkstra. En faisant cela, nous donnons un nouveau réseau graphique à chaque appel de l'algorithme et de ce fait, il peut arriver que pour l'interconnexion du même graphique,

l'algorithme rend différents chemins courts pour une destination en raison de la mise à jour de ces liens. Ainsi, de cette manière l'algorithme sélectionne et utilise de multiples courts chemins, avec un minimum de poids. Toutefois, l'algorithme de Dijkstra est capable de retourner un seul plus court chemin, il ne retourne jamais plusieurs courts chemins pour un réseau d'interconnexion graphique.

Algorithme de génération du chemin :

Début

Pour PE=1 à NbrPE faire
Si PE=RootPE alors return aucun chemin n'est exigé ;
Dijkstra (RootPE, PE, GridMatrix) ;

Retourner l'ensemble des liens constituant le chemin Ptsi ;

Mise à jour de la charge actuelle de données sur chaque lien de | Ptsi | ;

Définir les temps de début d'envoie et fin du paquet à travers chaque lien du chemin;

Envoie des données ;

Fpour ;

Fin.

4.11 RESULTAT

4.11.1 Front Pareto : Le Front Pareto est généré d'après les solutions existantes dans l'archive et Sont dominantes. Notre Front Pareto est un plan orthogonale qui a comme axe d'ordonnées le temps d'exécution et pour l'axe des abscisses l'énergie consommée. Les points représentés sont des placements avec coordonnées (Temps exécution, Energie consommée).

-On fait représenter ces solutions dans un graphe qui a les deux objectifs comme plan (ordonnées & abscisses) et ainsi nous aurons un Front Pareto qui permet au décideur de faire le choix de la solution qui lui convienne. Les figure 4.11 ci-dessous représentent le regroupement des solutions vers l'optimal d'une façon progressive en éliminant les solutions dominées et en se rapprochant vers le front Pareto.

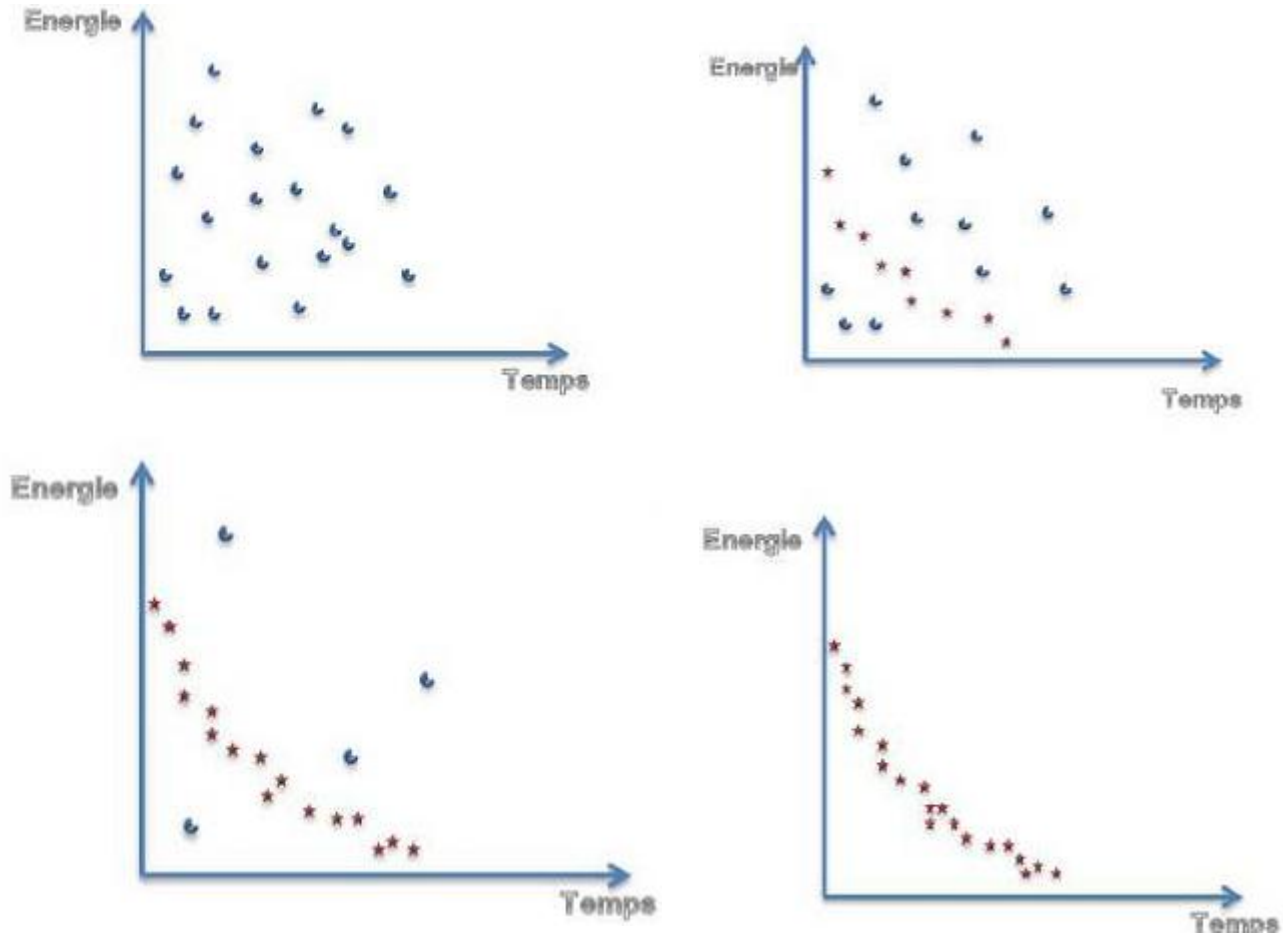


Figure 4.12- Regroupement des solutions durant les itérations

4.11.2 Diagramme de Gantt

- On peut avoir le diagramme de Gantt pour chaque solution qui représente un emplacement. Ce type de diagramme permet de visualiser l'enchaînement des différentes étapes du déroulement d'un projet et offre, ainsi, la possibilité :
- de prévoir suffisamment à l'avance les actions à entreprendre pour minimiser le temps de réalisation d'un projet.
- d'identifier les tâches critiques (tâches pour lesquelles aucun retard n'est possible sans retarder d'autant la date de fin du projet) et les marges (totale, libre et certaine) des autres.
- de faciliter le suivi de l'état d'avancement du projet (notamment évaluation de l'impact global d'un retard sur une tâche intermédiaire).
- de gérer au mieux les éventuels conflits de ressources.

- de servir d'outil de communication entre les différents acteurs impliqués dans la réalisation du projet.

4.11.3 Choix d'une solution par l'utilisateur

Le choix final d'une solution, parmi celles qui sont proposées, est laissé au décideur ou à l'utilisateur. Puisque les méthodes d'optimisation multi objectif donnent un ensemble de solutions et non pas une seule solution. Donc le choix final est réservé aux utilisateurs. Dans le principe du Système d'Aide à la Décision, le choix final est laissé à l'utilisateur.

4.12 CONCLUSION

Après avoir exposé les concepts qui servent de base à l'algorithme génétique et l'algorithme de B&B, nous avons réalisé une adaptation de ces algorithmes dans une démarche de recherche de bonnes solutions à notre problème de placement d'application intensive sur une architecture MPSoC. En plus, on a aussi exposé les concepts de l'algorithme de Dijkstra et son adaptation à notre domaine d'intérêts.

Dans le prochain chapitre, nous proposerons une implémentation de l'hybridation de l'algorithme génétique "AG», méthode de B&B et Dijkstra appliqués à notre problème.

Chapitre V

« Implémentation et Expérimentation »

5. 1 INTRODUCTION

Dans le chapitre précédent nous avons présenté nos approches proposées pour la résolution du problème de mapping multi-objectifs des applications intensives sur l'architecture MPSOC, notre contribution intervient dans une phase très importante dans la conception des systèmes embarqués, c'est celle de l'association application /architecture. Dans ce chapitre, nous allons voir l'implémentation de ces approches en commençant par l'environnement et les outils de développements, ensuite nous donnons le schéma fonctionnel et la description des modules.

5. 2 IDENTIFICATION DES BESOINS

D'après les problématiques posés, nous pouvons identifier les besoins que nous essayons de satisfaire grâce au logiciel de mapping :

- La possibilité de lancer une recherche permettant de trouver le placement optimal d'une application sur une architecture à moindre coût de communication (en utilisant l'algorithme du plus court chemin) sous les contraintes suivantes : équilibrage de charge, mémoire d'exécution disponible, bande passante des bus…etc., et cela par l'implémentation d'un algorithme génétique et la méthode B&B.
- la possibilité de charger et visualiser les graphes soit pour l'application ou pour l'architecture à partir d'un fichier XML.et même la possibilité de sauvegarder les graphes après leur modification.
- La possibilité d´étendre l'application par l'ajout des tâches ou des processeurs graphiquement.

L'application qu'on veut placer sur l'architecture est sous forme de taches (cercle) communicants via des messages (arc), permet aussi de paramétrer chaque tache (mémoire d'exécution requise et taille de la tache) et chaque message (Taille du message et bande passante requise). La même chose pour l'architecture : l'architecture du MPSoC sous forme de SOCs (carré) reliés par des bus (arc), elle permet aussi de paramétrer chaque SOC (charge minimum, charge maximum, mémoire d'exécution disponibles) et chaque bus (vitesse de transmission et bande passante et l'énergie consommée).

- minimiser l'occupation de mémoire par l'application
- minimiser le taux d'erreur commit par l'utilisateur.
- Visualiser les résultats de mapping sous forme de graphes front Pareto.

5. 3 ENVIRONNEMENT ET OUTILS DE MISE EN OEUVRE

L'implémentation est faite sur un système d'exploitation Windows® (Windows® 7 Service Pack 1) qui est un système multitâche reconnu pour sa stabilité et ses performances.

Pour le langage de programmation, nous avons choisi d'utiliser le langage JAVA sous l'environnement de développement Eclipse pour le développement de notre approche. Un choix qui se justifie par les nombreux avantages qu'il offre.

Java est un langage orienté objet qui présente beaucoup d'avantages : sécurité, robuste, indépendant du matériel, assurant une portabilité complète vers de très nombreux systèmes et son adaptabilité dans de nombreux domaines dont le domaine des systèmes embarqués.

Eclipse est un environnement de programmation visuelle orientée objet très Performant permettant le développement d'applications en vue de leur déploiement sous Windows ou sous Linux. Il trouve son origine au sein de la société IBM, qui a décidé en 2001 de mettre à disposition de la communauté Open Source l'ébauche d'une plate - forme de développement ouverte, entièrement écrite en Java, capable d'intégrer des extensions adaptées à diverses activités (débogage, modélisation, interfaces graphiques...).

L'objectif étant de créer un environnement de développement intégré polyvalent, Extensible, capable de travailler avec n'importe quel langage de programmation.

JAVA sous Eclipse propose un ensemble d'outils de conception pour le développement rapide d'applications selon le concept IP. Ceci lui rend un bon outil pour l'IDM (ou MDE).

5.4 PRESENTATION DU LOGICIEL ET DESCRIPTION DES MODULES

Comme cité auparavant, L'objectif de notre travail est de faire le mapping multi objectifs d'une application sur une architecture cible. L'entrée de notre application est constituée de deux graphes : un pour l'application et l'autre pour l'architecture. la sortie produit un front Pareto, qui est un ensemble des placements réalisables, qui respectent les contraints de Temps et de l´énergie. Cela par l'utilisation de l'AG pour placer les tâches irrégulières sur les

processeurs hétérogènes, et l'algorithme de B&B pour placer les tâches régulières sur des processeurs identiques. Le programme est constitué de deux grands modules : le premier regroupe les composants graphiques et le deuxième représente l'implémentation de tout ce qui est un calcul menant à la résolution de notre problème.

5.4.1 Interface graphique (partie graphique)

- ***Main_Frame :*** Elle permet la saisie graphique d'une architecture sous forme de Pe reliés par des bus, d'afficher et de modifier les propriétés de l'application et de l'architecture.

  Dans ce mode de saisie, l'utilisateur peut :
  - Introduire les graphes d'application et d'architecture.
  - L'extraction des graphes à partir de fichier d'entré au format XML.
  - modification et enregistrement des graphes
- ***Mapping_Frame :*** Elle englobe deux onglets, le premier est pour l'algorithme génétique et le deuxième pour la méthode Branch and Bound.

  Cette interface permet de :
  - Introduire les paramètres d'entrées pour les deux méthodes (AG, B&B)
  - Lancer le mapping irrégulier qui fait appel à l'AG
  - Lancer le mapping régulier qui fait appel à la méthode B&B
- ***Pareto_Frame*** **:** Une fenêtre pour afficher les solutions (Résultat de calcul) de chaque méthode sous forme de courbe (front de Pareto).

5.4.2 Les Calcules (partie texte)

Cette partie représente le code java qui correspond aux classes, aux méthodes et aux calculs implémentés pour résoudre le problème de mapping multi objectif.

- **Algorithme génétique** : méthode très connue dans le domaine d'optimisation multi-objectifs, son principe de fonctionnement est détaillé dans les chapitres 3 et 4. L'application et l'adaptation de cette dernière dans notre approche de résolution est illustrée par la figure 5.1.

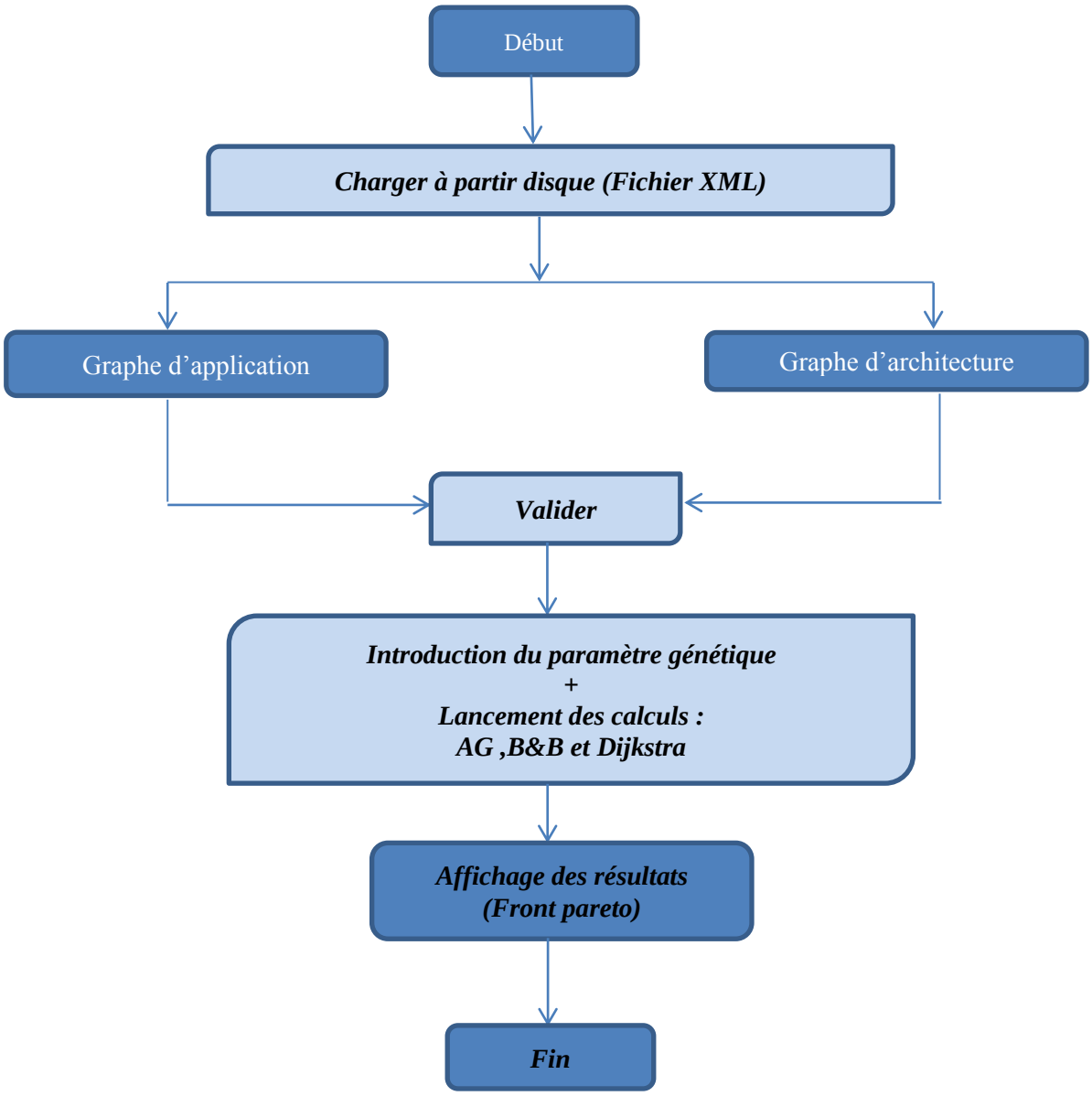


Figure 5.1 -Présentation globale de notre application

- **Branch And Bound** **:** méthode très connue dans le domaine d'optimisation multi-objectif, son principe de fonctionnement est détaillé dans les chapitres 3.
- **Algorithme de Dijkstra :** L'algorithme de Dijkstra pour le plus court chemin a été utilisé pour :

- Trouver le chemin le plus court possible entre *Pe* avec un minimum de poids (en charge communication) sur les arcs (les liens).

- Équilibrer la charge de communication sur les liens. Ce dernier se fait par mise à jour de la charge de la communication (poids sur l'arc), d'une façon à éliminer les chemins qui ont une charge inférieure à celle requise par le transfert de données.

- **Dominance :** une méthode basée sur le principe de dominance Pareto définie auparavant. Cette méthode est pour but d'éliminer toute solution (placement) trouvée, vérifiant les contraintes imposées (Mémoire, charge et Bande passante) et n'atteignant pas tout les objectifs voulu. Autrement dit et selon le principe de multi-objectifs, une solution qui est dominé par une autre.
- **Calcul_Temps :** cette méthode est implémentée pour calculer le temps d'exécution d'un placement.
- **Calcul_Energie :** même principe comme la précédente sauf que celle - ci calcule l'énergie consommée par un placement.

5. 5 MODELISATION UML

5.5.1 Diagramme de uses case

La Figure 5.2 présente le diagramme use-case de notre application :

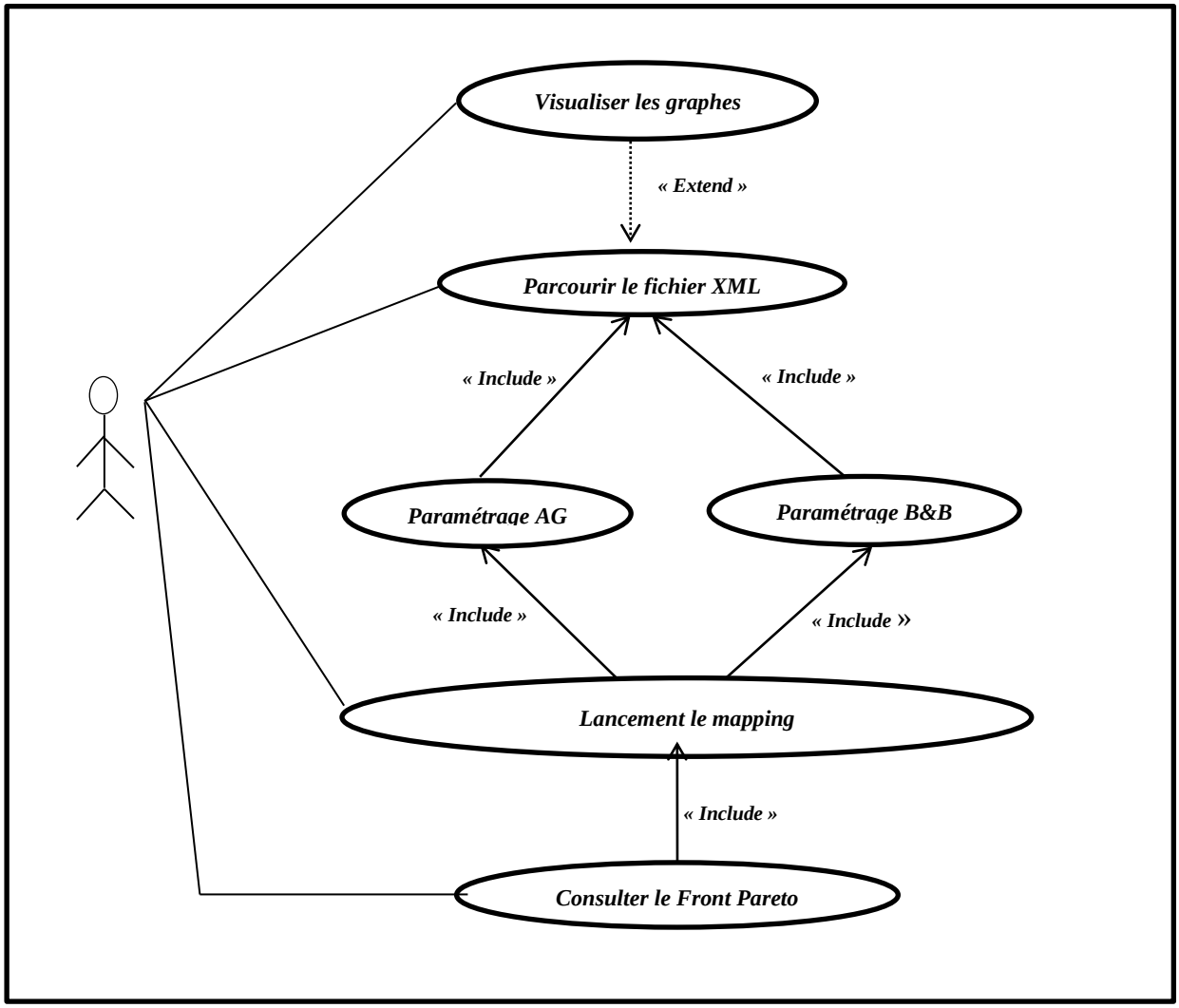


Figure 5.2- le diagramme use-case de l'application

5.5.2 Diagramme de classe (l'interface) :

La Figure 5.3 présente le diagramme de classe (interface) de notre application :

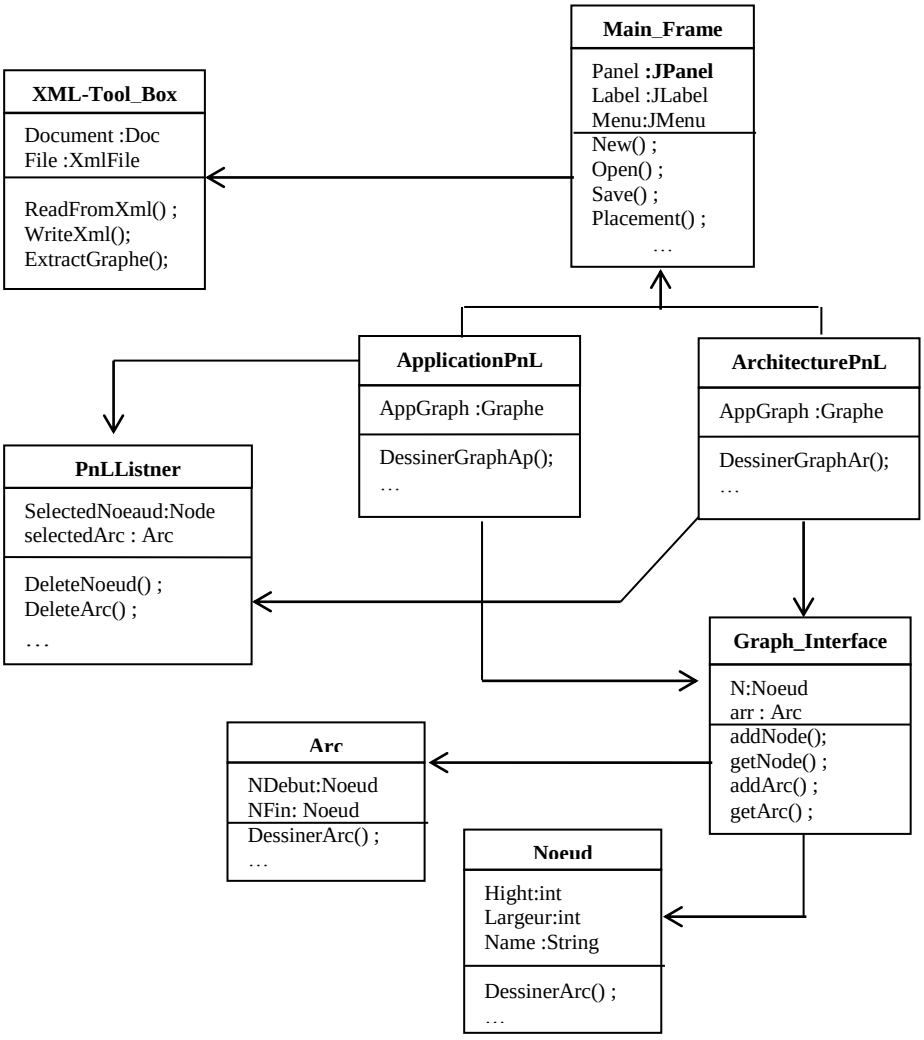


Figure 5.3- diagramme de classe (Interface)

5.5.3 Diagramme de classe(Mapping)

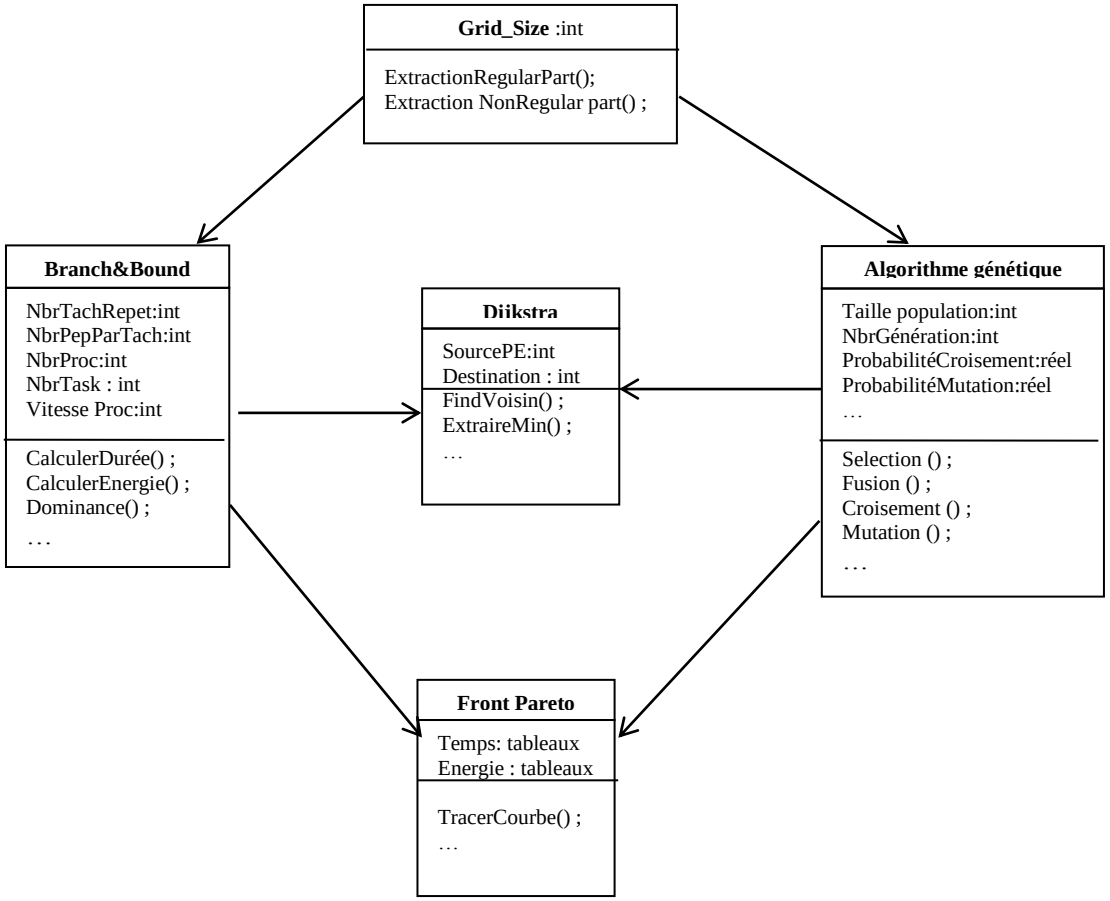


Figure 5.4- diagramme de classe (Mapping)

5.6 LES PRINCIPALE ETAPES D'UTILISATION DU LOGICIEL

Nous présentons maintenant les étapes et les schémas qui décrivent la manière d'utiliser notre logiciel. D'abord, l'utilisateur doit charger un exemple à partir d'un fichier XML (cf. figure 5.5), ensuite, on peut modifier les graphes en ajoutant des nœuds ou des liens. On peut aussi visualiser les matrices d'incidences d'architecture et d'application avec leurs caractéristiques.

L'utilisateur peut lancer l'algorithme génétique en ayant la possibilité d'insérer (modifier) ses paramètres de base comme la taille de la population initiale, le nombre de génération, probabilités de croisement et mutation.

Une fois toutes ces données sont saisies, il reste à appuyer sur le bouton placement pour lancer les calculs de recherche de placements (solutions) multi objectifs. Les résultats (cf. figure 5 .6) de ces derniers seront affichés dans une autre fenêtre avec le front Pareto leur correspondant.

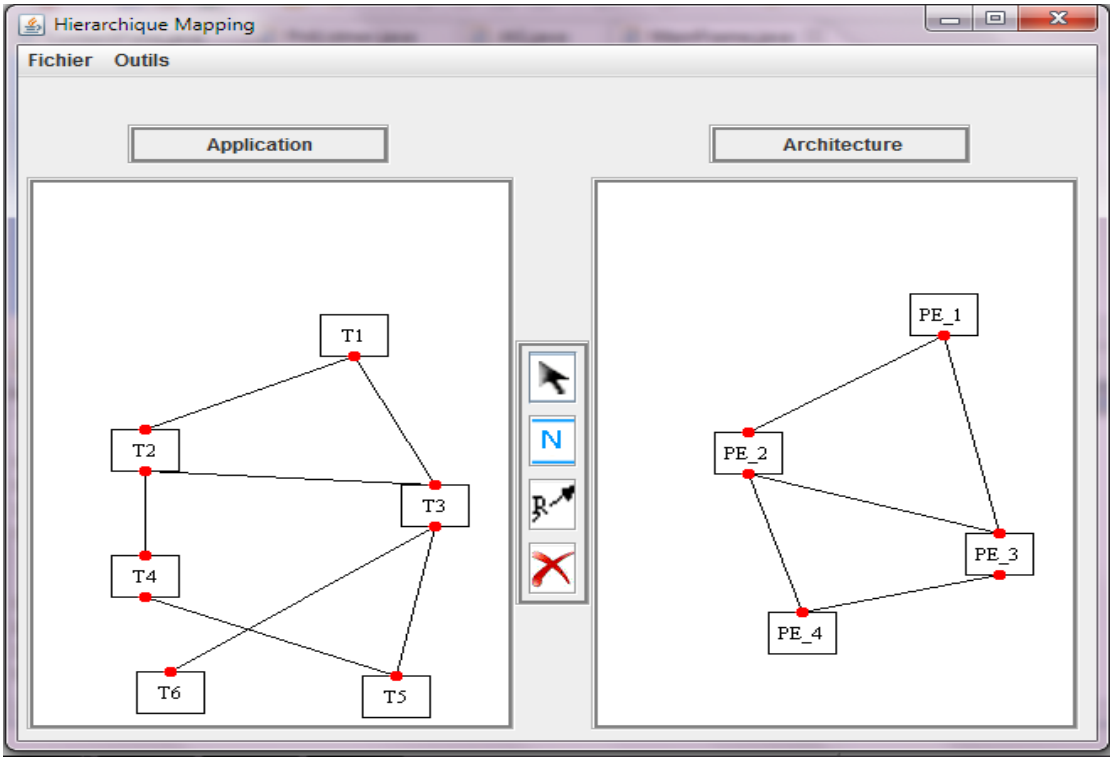


Figure 5.5- Lire des exemples à partir fichier XML

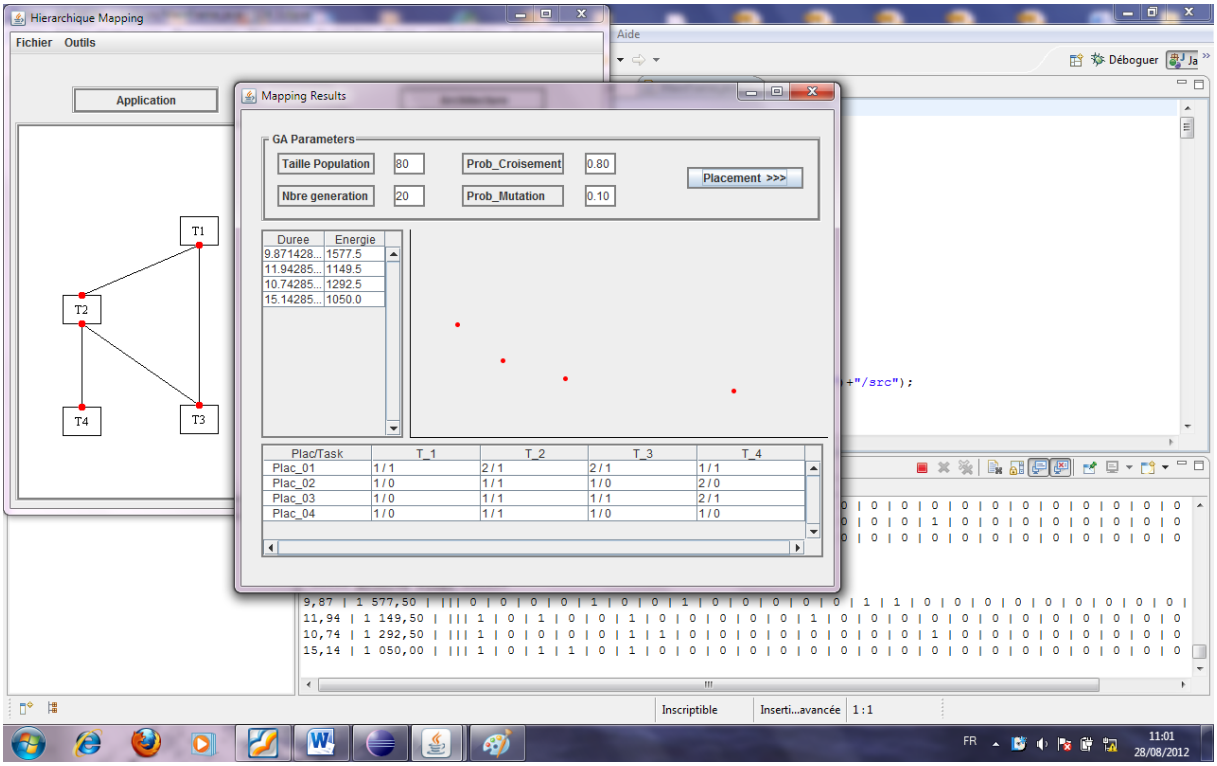


Figure 5.6 - Fenêtre principal du logiciel et l'affichage des résultats

Après l'exécution de notre approche hybridée (GA et B & B) en utilisant les propriétés et les paramètres de base indiqués dans le tableau 5.1, les résultats obtenus sont comme suit:

Algorithme	***Paramètre de base***	
Algorithme génétique	-Nombre de tache -Nombre de processeur -Architecture -Latence -Taille de la Population -Nombre de Génération -Probabilité de croisement -Probabilité de Mutation	21 9 Topologie en étoile 1 unité 80 40 80% 1 %
Algorithme brunch&baund	-Nombre de tache -Nombre de processeurs -Architecture -Latence	290 9 Mesh 2D (3*3) 1 unité

Tableau.5.1-Représentation des paramètres de base

5. 7 AFFICHAGE DES RESULTATS

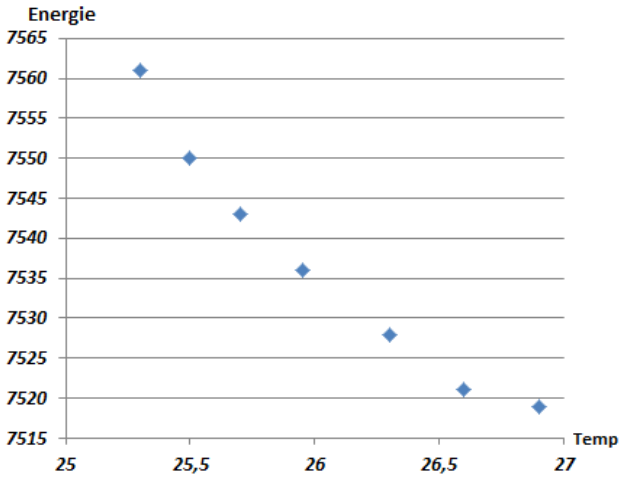


Figure.5.7-Front Pareto pour AG

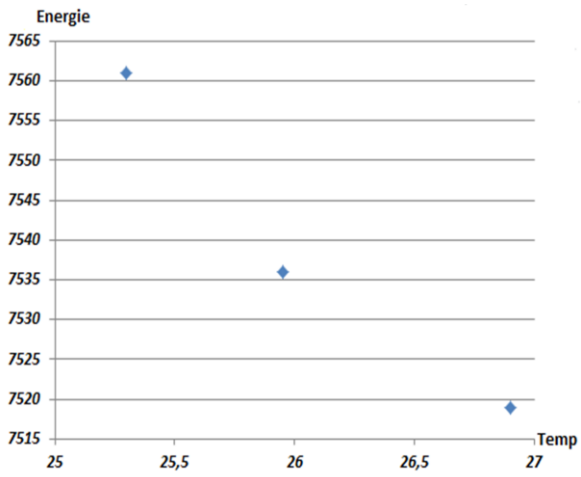


Figure.5.8- Front Pareto pour B&B

5. 8 EXPERIMENTATION POUR DETERMINER LES VALEURS DES PARAMETRES

Nous avons soumis l'implémentation de l'algorithme génétique détaillée précédemment à une série de tests. Ces tests visent principalement à la mesure de l'efficacité de l'algorithme et de la qualité des solutions qu'il fournit. Nous avons effectué des analyses de sensibilité relatives aux paramètres intrinsèques de l'algorithme génétique, puis mesuré l'évolution de la performance en fonction des variations de ces paramètres.

Les valeurs obtenues pour la configuration de référence sont :

- Taille de la population : 80 chromosomes
- Nombre de générations : 20
- Probabilité de croisement : 0,8
- Probabilité de mutation : 0,10

5. 8.1 Influence de la probabilité de croisement

Une série d'expériences a été réalisée pour étudier l'influence des variances de la probabilité de croisement. Nous donnerons les valeurs de probabilité suivantes : 0,2; 0,6; 0,80.

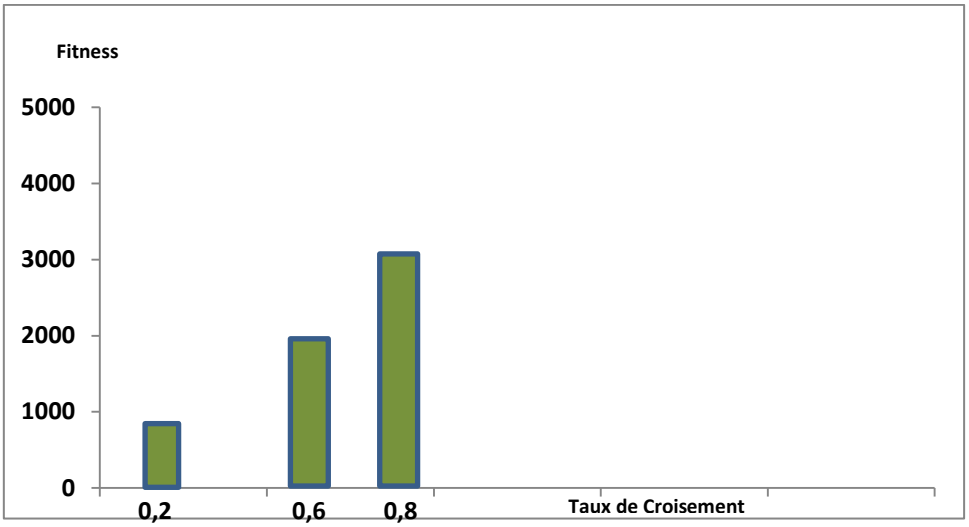


Figure. 5.9 - Influence des probabilités de croisement.

Nous remarquons qu'une probabilité de croisement autour de 0,80 donne une grande valeur de fitness donc une solution optimale.

5.1.2 Influence de la probabilité de mutation

L'opérateur de mutation a comme seul but d'amener une certaine diversité, les tests de variation de ce paramètre visaient à trouver la meilleure valeur de mutation.

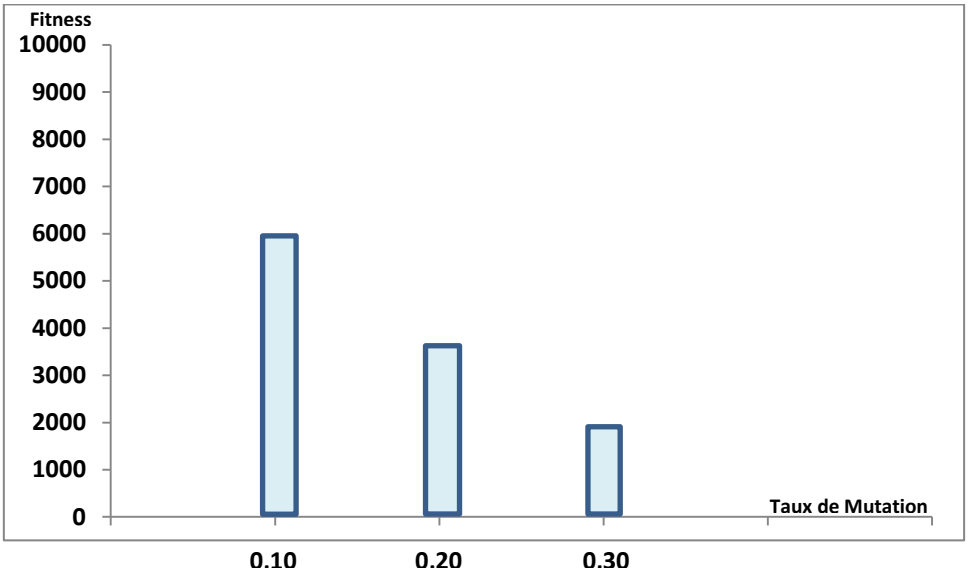


Figure 5.10 - Influence des probabilités de mutation

D'après la figure 5.10 nous constatons que les tests ont affirmés que la probabilité de 0.10 donne une grande valeur pour la fonction fitness, c'est-à-dire une solution meilleure.

5.9 MAPPING MULTI OBJECTIFS D'UNE APPLICATION TEMPS RÉEL

Nous commençons par présenter un benchmark réel sur lesquels nous avons effectué nos tests, puis nous étudions les résultats obtenus après la réalisation des tests sur ce benchmark.

Présentation du benchmark Picture in Picture (PiP)

L'application PiP est l'application la plus simple testée sur notre application. Cette fonction, Créer par Greg Dockery, permet de regarder une deuxième chaine sur l'écran d'une télévision. Une première chaine est affichée en plein écran alors que la seconde est affichée dans une Petite fenêtre. Elle requiert l'utilisation de deux tuners indépendants, un pour chaque chaine. Le benchmark est composé de 9 tâches connectées entre elles via 11 liens. La Figure 5.11 présente le graphe d'application le décrivant.

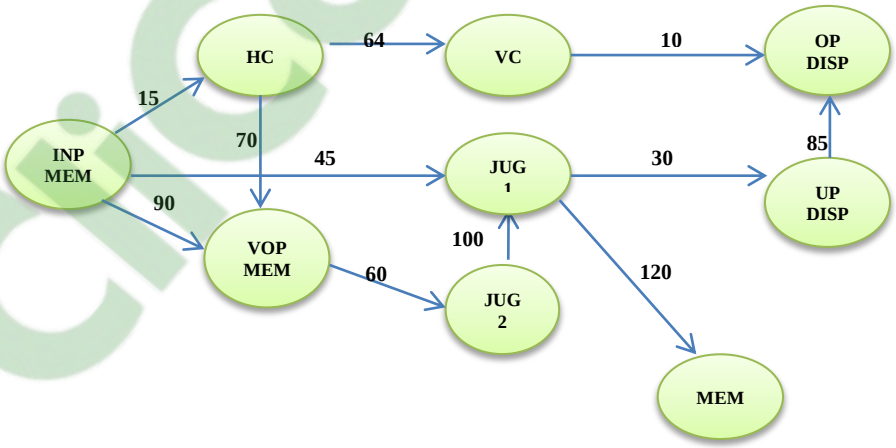


Figure 5.11-le placement et l'ordonnancement des différentes tâches.

Les communications ainsi que les volumes d'échanges sont donnés par la matrice suivante :

	T1	T2	T3	T4	T5	T6	T7	T8	T9
T1		90	15						
T2				60					
T3		70			64				
T4						100			
T5								10	
T6							30		120
T7									85
T8									
T9									

Tableau 5.2- matrice des communications inter-tâches

L'architecture cible est un Mesh-2D constitué de 5 processeurs de trois types différents. Le premier type un DSP, le deuxième est un FPGA, et le troisième est un GP. La topologie est illustré par le graphe 5.12 où les valeurs sur les arcs indiquent la latence, l'énergie et la bande passante du lien correspondant. Les caractéristiques des processeurs sont données par le tableau 5.3.

	Type du Proc	Vitesse 1 (mode 1)	Energie 1 (mode 1)	Vitesse 2 (mode 2)	Energie2 (mode2)	Mémoire
P1	GP	80	2,0	100	2,5	800
P2	FPGA	40	1,8	60	2,0	1000
P3	DSP	50	1,2	70	1,5	500
P4	FPGA	40	1,8	60	2,0	1000
P5	GP	80	2,0	100	2,5	800

Tableau 5.3- Les caractéristiques de l'architecture cible

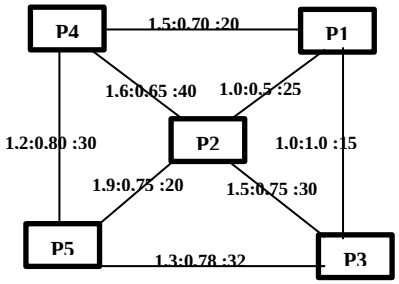


Figure.5.12- l'architecture cible

Les tailles des tâches sont données dans le tableau 5.4 suivant. Ces tailles sont mesurées en nombre de cycles selon le type de processeur :

Tâches	Nombre de cycles selon processeurs type 1	Nombre de cycles selon processeurs type 2	Nombre de cycles selon processeurs type 3
T1	200	240	230
T2	220	200	210
T3	210	240	170
T4	190	220	160
T5	150	180	200
T6	180	240	170
T7	230	190	240
T8	200	210	180
T9	220	160	200

Tableau 5.4- Les tailles des tâches selon le type du processeur

En utilisant L'algorithme génétique, on obtient toutes les bonnes solutions vérifiant les deadlines. Ces placements sont donnés dans le tableau 5.5 suivant où la première colonne désigne les tâches, les autres colonnes désignent les différents placements c'est-à-dire les processeurs auxquels ces tâches sont affectées, le cout de chaque placement (durée et l'énergie) est affiché dans les deux dernières lignes.

Tâches	Plac1	Plac2	Plac3	Plac4	Plac5	Plac6	Plac7	Plac8
T1	P1	P3	P3	P4	P5	P2	P1	P4
T2	P4	P5	P1	P5	P5	P1	P5	P3
T3	P3	P1	P4	P2	P1	P2	P1	P3
T4	P5	P1	P2	P3	P3	P5	P3	P1
T5	P4	P3	P3	P1	P4	P4	P2	P4
T6	P2	P4	P4	P3	P2	P3	P5	P1
T7	P3	P5	P1	P2	P3	P4	P5	P2
T8	P1	P2	P3	P4	P1	P2	P4	P5
T9	P5	P2	P5	P1	P4	P1	P2	P2
Durée	13,1	14,857	12,02	15,871	13,396	14,282	15,971	14,2
Energie	1645,0	1220,0	1337,0	1492,5	1387,5	1312,0	1214,0	1097,0

Tableau 5.5- Les meilleurs placements des différentes tâches

On constate que pour le moment le meilleur en temps d'exécution c'est pour le placement 3, L'ordonnancement est obtenu par le diagramme de Gantt dans la figure (5.13). Le temps de fin de l'application selon ce placement est égal au temps de fin de la dernière tâche, $T_{fin}=T_{fin}(T8)=12,02$ ut. Le temps d'exécution de l'application est égal au temps de fin de la dernière tâche moins le temps de début de la première tâche dans l'exécution.

Pour simplifier les calculs on a pris comme temps de début l'instant $T_{deb}(T1)=t0=0$ alors :

$D=T_{deb}(T1)-T_{fin}(T8)=12,02$.

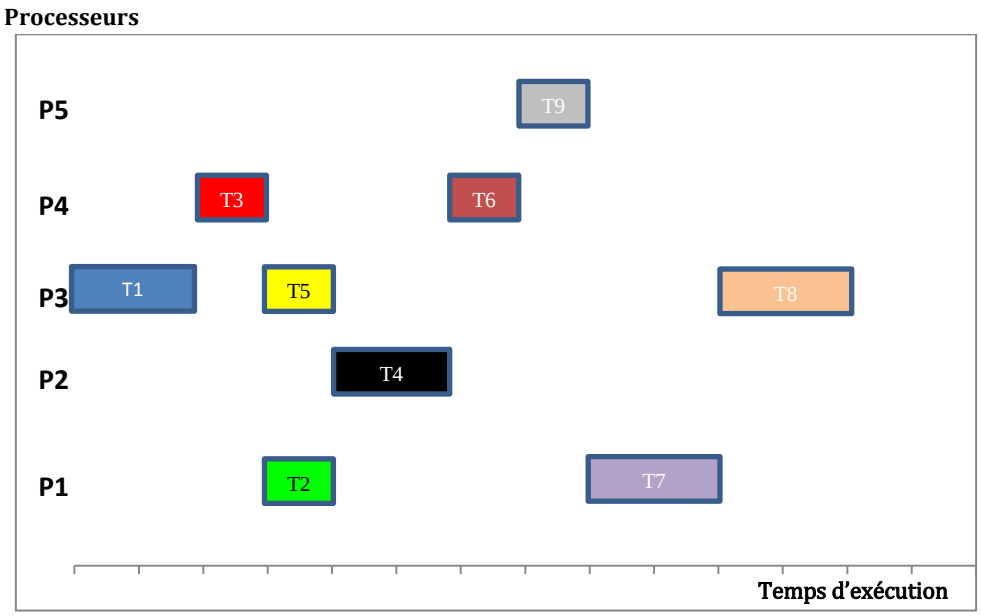


Figure 5.13-le placement et l'ordonnancement des différentes tâches

5.10 CONCLUSION

Dans ce chapitre, nous avons présenté les différents outils utilisés pour concevoir et réaliser notre approche, puis nous avons vu les différentes classes que nous avons implémentées. A travers les tests effectués, le but était d'étudier les performances afin de bien évaluer notre système de mapping. Nous avons montré et discuté les différents aspects de notre approche afin de montrer que cette démarche peut produire de bons résultats avec moins de complexité.

CONCLUSION GENERALE ET PERSPECTIVES

Conclusion générale et Perspectives

Les réseaux sur puce (network in chip (NOC**)**, nouveau concept d'interconnexion rendu possible grâce à l'évolution de la technologie sur silicium, est en passe de devenir une solution privilégiée pour simplifier l'intégration de composants complexes sur des systèmes sur puce. Comme toute nouvelle technologie, elle requiert des efforts de recherche importants, en particulier pour la simplification et l'accélération des phases de conception. Plusieurs outils automatisant les étapes les plus importantes ont été proposés. La plupart de ces solutions se focalisent sur le mapping car il représente une phase très importante dans la mise en œuvre de ces systèmes embarqués. La conception de ces derniers impose à respecter certaines contraintes logicielles et Matérielles, contrairement aux systèmes informatiques, ces contraintes doivent être prises en compte simultanément. Dans ce type de système où on fait du logiciel dédié et inversement pour le matériel. Les deux types de contraintes sont vérifiées en même temps. Lors de la phase de mapping, on doit prendre en considération les contraintes de consommation d'énergie, de temps, de mémoire, bande passante…etc. Ce problème est souvent désigné dans la communauté par l'AAS (Assignation,Affectation et Scheduling). Les contraintes dont on a parlé ne peuvent être traitées séparément d'où pour que l'AAS soit réaliste on doit considérer certaines d'entre elles comme des objectifs qu'on doit atteindre simultanément. C'est ce qui rend ce problème, un problème d'optimisation multi-objectifs.

Le but principal du travail présenté dans ce document est de proposer une technique de mapping d'une application sur une structure de réseau sur puce en minimisant le temps d'exécution et l'énergie consommée. Après avoir passé en revue les différentes solutions proposées, nous avons cherché à emprunter une nouvelle voie pour la résolution du problème.

Pour cela et d'après les recherches effectuées, on a utilisé une méthode hybridée basée sur deux algorithmes d'optimisation. Un algorithme génétique qui est un algorithme heuristique, et l'algorithme de Branch & Bound qui est un algorithme exact.

Ces deux algorithmes cherchent la bonne solution dans un espace de solution réalisable en respectant les contraintes de temps et d'énergie.

Nous considérons pour le moment que notre solution est très efficace parce qu'on a utilisé une meilleur méta heuristique et une méthode exact pour traiter les deux types de taches régulières et irrégulières et assurer le parallélisme de taches et de donnés (GILR : Globally

Irregular Locally Regular). En attendant des expérimentations et simulations plus poussées, nous considérons que notre solution est très efficace, et qui ne doit pas nous empêcher de faire d'autres expérimentations pour plus de validation et de prospecter d'autre perspectives. Pourquoi pas proposer d'autres méthodes à la place de AG , B&B et Dijkstra pour solutionner ce problème autrement .

BIBLIOGRAPHIE

Bibliographie

[1] Mimouni. " Concepts fondamentaux des systèmes embarqués", Mémoire de fin d'étude. Université d'Oran. 2007.

[2] Mobile Intel Pentium 4 Processor supporting Hyper-Threading Technology on 90-nm process technology, specification update. Document Number: 302441-001. juin 2004.

[3] Wikipédia. Système embarqué, 2012. http://fr.wikipedia.org/wiki/Système_embarqué.

[4] Patrice KADIONIK. "Introduction au système embarqué». Conférences à l'ENSEIRB.2005.

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

[6] Amine OUAFI. "Optimisation multi-objectif à l'aide d'un algorithme génétique pour les applications embarquées. Mémoire de fin d'étude. Université d'Oran. Juin 2009.

[7] Ivan DOYROV. "Conception des systèmes mono-puce multiprocesseur de la simulation vers la conception". Université Joseph Fourier GRENOBLE. 2006.

[8] Object Management Group. "Inc,editor. UML 2 Infrastructure (Final Adopted Specifcation)".Septembre 2003.

[9] Model Driven Engineering. "Planet MDE Model Driven Engineering". 2007.

[10] Jean-Marie FAVRE et Jacky ESTUBLIER et Mireille BLAY-FORNARINO. "L'ingénierie dirigée par les modèles", au-delà du MDA. Hermès Science, Lavoisier. Janvier 2006.

[11] Ashish Menna. Allocation, Assignation et Ordonnancement pour les systèmes sur multiprocesseurs. PhD thesis, Université des sciences et technologie de Lille, décembre 2006.

[12] WOLF Wayne and JERRAYA Ahmed Amine « Multiprocessor systems-on-chips ». Hardcover, Morgan Kaufmann Publishers, 2004.

[13] Oulhaci Abdalhak,Rajdal mohammed akli. "Ordonnancement hiérarchique d'application temps réel sur une architecture NOC ". Mémoire de fin d'étude. Université d'Oran. Juin 2009.

[14] BOUCHHIMA Aimen « Modélisation du logiciel embarque à différents niveau d'abstraction en vue de la validation et la synthèse des systèmes monopuces ». Thèse de doctorat. Institut National Polytechnique de Grenoble (INPG), 2006.

[15] MOSTEFA Meriem. "Placement des tâches répétitives sur une architecture régulière embarquée. Juin 2009.

[16] Object Management Group: A UML profile for MARTE, 2007. http://www.omgmarte.org

[17] L. Rioux, T. Saunier, S. Gerard, A. Radermacher, R. de Simone, T. Gautier, Y. Sorel, J. Forget, J. - L. Dekeyser, A. Cuccuru, C. Dumoulin et C. Andre : MARTE : A new profile

RFP for the modeling and analysis of real - time embedded systems. In UML - SoC'05, DAC 2005 Workshop UML for SoC Design, Anaheim, CA, Juin 2005.

[18] Mohamed Akli Redjedal: Optimisation multicritère pour le placement d'applications intensives sur système sur puce (SoC); pour obtenir le titre de Master. l'Université de Lille Mention : .Informatique. Juin 2010.

[19] P. Boulet. Contributions aux environnements de programmation pour le calcul intensif. Habilitation à diriger des recherches, Université des Sciences et de Technologie de Lille, Décembre 2002.

[20] Mouna makhlouti ,Méthode de conception rapide d'architecture massivement parallèle sur puce : de la modélisation a l'expérimentation sur FPGA ,octobre 2010.

[21] Ouassila Labbani. Modélisation à haut niveau du contrôle dans des applications de traitement systématique `a parallélisme massif. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, novembre 2006.

[22] S. Le Beux, Ph. Marquet, O. Labbani, and JL. Dekeyser.FPGA Implementation of Embedded Cruise Control and Anti-Collision Radar.In 9th Euromicro conference on digital system design, DSD, Dubrovnik, Croatia, August 2006.

[23] Radu Cornea, ShijairtMohapatra, NikilDutt,Alex Nicolau, and NaliniVenkatasubramanian. Power-aware multimedia-streaming in heterogenous multi-use environents. In Concurrent Information Processing and Computing Advanced Research Workshop, Sinaia, Romaina, uly 5-10 2003.

[24] Christophe Manas. Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones. PHD thesis, Université de Rennes, Décembre 1989.

[25] PraveenK.Murthy and Edward A.lee. Multidimensional synchronous data flow. IEEE Transaction on signalProcessing, juillet 2002.

[26] P.K.Murthy and Edward A.lee. A generalization of multidimensional synchronous dataflow to arbitrary sampling Lattices. Technical Report UCB/ERL M95/59, EECS Department of University of California, Berkeley, mais 1995.

[27] C.Mauras,Alpha, un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones, thèse de doctorat, université de rennes I, décembre 1989.

[28] R.karpR.Miller ,andS.winagrad,Theorganisation of computations for uniform recurrence equations .journal of the ACM,14(3):563-590 juillet 1967.

[29] William Thies, Michal Karczmarek et SamanAmarasinghe :StreamIt : A language for streaming applications. In Compiler Construction : 11th International Conference, CC 2002, Held as Part of the Joint European Conferences on Theory and Practice of Software.

[30] Michael Weeks «digital signal processing using MATLAB and Wavelets electrical engineering series».2008.

[31] E. A. Lee et D. G. Messerschmitt : Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. on Computers, janvier 1987.

[32] E. A. Lee et D. G. Messerschmitt : Synchronous Data Flow. Proc.of the IEEE, septembre 1987.

[33] Julien Soula. Principe de compilation d'un langage de traitement de signal. PhDthesis, Laboratoire d'Informatique Fondamentale de Lille, Université des sciences et technologies de Lille, décembre 2001.

[34] Calin Glitia. Optimisation des applications de traitement systématique intensives sur System on Chip. PhDthesis, Laboratoire d'Informatique Fondamenatale de Lille, Université des Sciences et Technologies de Lille, novembre 2009.

[35] Pierre Boulet : Formal semantics of Array -OL, a domains pecific language for intensive multidimensional signal processing. Rapport technique RR -6467. Université des Sciences et Technologies de Lille, mars 2008.

[36] AROUI Abdelkader, " Ordonnancement multi objectif d'application intensif sur une architecture régulières," pour obtenir le diplôme de magistère, université d'Oran, Algérie, février 2012.

[37] Philippe Dumont : Spécification multidimensionnelle pour le traitement du signal systématique. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Décembre 2005.

[38] Arnaud LIEFOOGHE : Méta heuristiques pour l'optimisation multiobjetif : Approches coopératives, prise en compte de l'incertitude et application en logistique, Pour obtenir le grade de Docteur. Université des Sciences et Technologies de Lille. Décembre 2009.

[39] Mohammed Amine OUKEBDANE & Mohamed Amine GHEZIZAT, Ordonnancement multicritères d'applications embarquées, pour obtenir le grade D'ingénieur d'état. Université d'Oran - Es- Senia; Faculté des Sciences Département d'Informatique. Juillet 2010.

[40] Wikipedia. Selection naturelle, 2012 http://fr.wikipedia.org/wiki/Selection_naturelle.

[41] BENYAMINA Redha, HOUARI Djalal, "Conception d'un système hybride R.N.MLP-AGs pour une classification de données de télédétection".

[42] Xavier Hüe, "Genetic Algorithms for Optimisation: Background and Applications", Edinburgh Parallel Computing Centre, University of Edinburgh, 1997.

[43] Wikipedia.Algorithmesgénétiques.2012http://fr.wikipedia.org/wiki/Algorithmes_génétiq ue.

[44] Nadira Benlahrache, " Optimisation Multi-Objectif Pour l'alignement Multiple de Séquences," pour obtenir le diplôme de magistère, université de Constantine, Algérie, Septembre, 2007.

[45] Souquet Amédée, Radet Francois-Gérard "Algorithmes Génétiques", 2004.

[46] Xavier Hüe, "Genetic Algorithms for Optimisation: Background and Applications", Edinburgh Parallel Computing Centre, University of Edinburgh, 1997.

[47] Youssef ATAT. "Conception de haut niveau des MPSoC's à partir d'une spécification Simulink".2007.

[48] M.DORIGO ET V.MANIEZZO ET A.COLORNI. "The Ant System: Optimization by a colony of cooperating agents". IEEE Transaction on System, Man, and Cybernetics- Part B, Vol.26, No.1, PP.1-13. 1996.

[49] N.HALLAM. "Diversity Preservation Techniques in Multiobjective Evolutionary Algorithms: Analysis and Proposal". Research Report CSIT-RR-200507-001, University of Nottingham. 2005.

[50] L.BENINI et D.BERTOZZI et A.GUERRI et M.MILANO. "Allocation, Scheduling and Voltage Scaling on Energy aware MPSoC". In Proc. Of CPAIOP-2006, pp.44-58, 2006.

[51] Luca BENINI et Giovanni De Micheli. "Networks on chips: A new SoC paradigm". Computer, 35(1) : 70-78. 2002.

[52] T.Lei, S. Kumar. "DA two-step Genetic Algorithm for Mapping Task Graphs to NoC Architecture", DSD. 2003.

[53] J.HU ET R.MARCULESCU. "Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints". In Proc. Design, Automation and Test in Europe Conf. Paris, France. Fevrier 2004.

[54] K.Deb. "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multiobiective Optimization: NSGA H Parallel problem Solving form Nature - PPSN VI", Springer Lecture Notes in Computer Science, p. 849-858. 2000.

[55] E.ZITZLER ET L.THIELE. "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach, Evolutionary Computation". Vol. 3, 1999, pp. 257271. 1999.

[56] E.Dijkstra. "In a Note on Two Problems in Connexion with Graphs". pages 269-271. Juillet 1959.

[57] Jaspal Subhlok and Gary Vondran. Optimal use of mixed task and data parallelism for pipelined computations. J. Parallel Distrib. Comput., 60(3):297 – 319, 2000.

[58] Soumen Chakrabarti, James Demmel, and Katherine A. Yelick. Modeling the benefits of mixed data and task parallelism. In ACM Symposium on Parallel Algorithms and Architectures, pages 74 – 83, 1995.

[59] Henri Bal and Matthew Haines. Approaches for integrating task and data parallelism. IEEE Concurrency, 6(3):74 – 84, 1998.

Résumé

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

La phase de mapping représente une phase centrale lors de la mise en œuvre d'un réseau sur puce. Elle permet de placer les éléments d'une application sur l'architecture. Le but est de minimiser le coût des communications, la consommation d'énergie ou la latence du système tout en respectant les contraintes de bande passante, de temps réel et la taille. Il devient nécessaire d'élaborer des outils et des méthodes qui automatisent ce processus.

C'est dans ce cadre que s'insère notre travail. Il consiste à proposer une nouvelle technique de placement d'une application de calcul intensif sur les différents éléments d'une architecture de réseau sur puce afin de minimiser le coût des communications. Cette nouvelle solution est basée sur une hybridation des méthodes évolutives (Algorithme génétique), algorithme d'optimisation par Branch and Bound (B&B) et l'algorithme de dijkstra multi objectif.

Mots clés :

Système Embarqué; NOC; Soc; Mpsoc; Mapping; Algorithme Génétique; Méthode B&B; GPNOCSIM; Algorithme De Dijkstra; GILR.