Titre: Title:	Modèle de placement pour les architectures nano-composantes
Auteur: Author:	Maimouna Amadou
Date:	2009
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Amadou, M. (2009). Modèle de placement pour les architectures nano-composantes [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/237/

UNIVERSITÉ DE MONTRÉAL

MODÈLE DE PLACEMENT POUR DES ARCHITECTURES NANO - COMPOSANTES

MAIMOUNA AMADOU
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE LôOBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2009

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

MODÈLE DE PLACEMENT POUR LES ARCHITECTURES NANO-COMPOSANTES

présenté par : AMADOU Maimouna
en vue de lôobtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury dôexamen constitué de :

M. BOIS Guy, Ph. D., président
Mme NICOLESCU Gabriela, Doct., membre et directrice de recherche
Mme BOUCHENEB Hanifa, Doctorat., membre et codirectrice de recherche
M. DAVID Jean-Pierre, Ph. D., membre

DÉDICACE

A mes parents

Amadou garba & ramatou moussa

REMERCIEMENTS

Jôaimerais, en premier lieu, remercier chaleureusement ma directrice de recherche, Professeur Gabriela Nicolescu. Je lui suis très reconnaissante pour môavoir accueillie dans son laboratoire de recherche. Ses orientations et son encadrement mônt permis de travailler sur ce projet qui nôaurait pas aussi bien abouti sans son aide précieuse. Jôai énormément apprécié sa patience, sa compréhension et la confiance quôelle a placée en moi.

Mes sincères remerciements vont également à ma co-directrice, professeur Hanifa Boucheneb pour môavoir accueillie, conseillée et guidée depuis mon arrivée à lôécole polytechnique. Jôai apprécié son encadrement, son écoute et sa compréhension.

Je voudrai remercier le professeur Ian OôConnor pour lôintérêt quôil a accordé à mes travaux. Il môa réservé un accueil chaleureux au se in de son laboratoire et a mis à ma disposition toutes les ressources nécessaires pour mener à bien ce projet. Jôai beaucoup apprécié son encouragement, son encadrement et son aide tout au long de ces travaux.

Je suis très reconnaissante envers les professeurs Guy Bois et Jean-Pierre David pour avoir accepté de faire partie de ce jury.

Je remercie vivement Alain, Azzedine, Bruno, Luiza, Sébastien et Taieb, mes collègues du groupe PIM, pour leurs suggestions, encouragements et conseils en tout genre.

Mes derniers remerciements, mais pas les moindres, vont à ce à qui je dois tout : mes parents, frères (Mahamadou, Boubacar) et sî urs (Fatouma, Mariama, Aminata, Rabiatou, Halimatou) ainsi que mes amis (Florence, Sidy Kounta, Gerry, Mayi, William, Fatoumata). Ils mônt toujours soutenue, appuyée et poussée en avant tout au long de ce projet.

RÉSUMÉ

Depuis la création de l'industrie des transistors CMOS, on assiste à un développement sans précédent de la miniaturisation. L'ITRS prévoit la limite des technologies basées sur le CMOS en 2020. Dans ce contexte, apparait de nouvelles disciplines, au cî ur de la nanotechnologie, qui permettent de définir de nouvelles technologies permettant de compléter et/ou remplacer les transistors CMOS. Ces nouveaux transistors ouvrent la voie vers un nouveau paradigme *d'architectures nano-composantes*. Ces architectures ont trois principales caractéristiques :

(i) Les cellules logiques sont dynamiquement reconfigurables. Ce qui donne la possibilité d'exécuter en pipeline plusieurs fonctions différentes;

(ii) La granularité est très fine. Ceci impose de considérer l'extensibilité des outils qui permettront l'exploitation de ces architectures;

(iii) Elles ont une structure hiérarchique particulière : Dans les architectures nano-composantes les cellules logiques sont organisées en matrices avec des connexions statiques et les matrices en réseau de matrices avec des connexions dynamiques. Ces architectures peuvent alors être paramétrées en fonction de la taille des matrices (nombre de cellules) et de la taille du réseau (nombre de matrices).

Pour prouver l'efficacité des architectures nano-composantes, il va falloir envisager la réalisation physique de systèmes complexes très performants basés sur ces technologies ainsi que l'utilisation des ces nano-systèmes. Comme l'accès au prototypage est très difficile et qu'il est souhaitable de réduire le temps de production des systèmes, la définition de nouveaux outils de conception assistée par ordinateur (CAO) s'avère nécessaire. Plusieurs outils CAO permettant la définition de systèmes basés sur les architectures conventionnelles existent. Cependant, ces outils ne prennent pas en compte les caractéristiques des architectures nano-composantes.

L'objectif de ce projet de recherche est d'aider à la conception de systèmes basés sur les architectures nano-composantes plus spécifiquement de définir un modèle de placement qui permettra :

(i) de placer des applications sur les architectures nano-composantes;

(ii) dœexplorer lœespace de solutions de conception défini par les paramètres des architectures, et

(iii) dœoptimiser quatre métriques importantes : le temps dœexécution de lœapplication, le coût de communication, le coût de reconfiguration et le nombre de cellules inactives.

Le modèle proposé, dans ce travail, a été défini en utilisant la théorie des graphes et les algorithmes génétiques. Il est réalisé en deux parties utilisant trois méthodes. Dans la première partie, lœapplication à placer est divisée en sous-fonctions en utilisant une méthode de partitionnement définie avec les concepts de théorie des graphes, puis la transposition de chaque sous-fonction sur une matrice est estimée en utilisant une méthode dœassignation matricielle définie avec les concepts de théorie des graphes. La seconde partie du modèle consiste à déterminer le placement global de toutes les sous-fonctions sur le réseau de matrices en utilisant une méthode de placement global exploitant les algorithmes génétiques.

Dans ce mémoire, le modèle de placement a été testé sur sept cas de test. Lœanalyse des résultats obtenus a permis de voir quœon peut atteindre, en moyenne, 60 % de taux dœoccupation globale pour un partitionnement sur des architectures ayant des matrices de 16 cellules. Lœexpérimentation de placement global nous a permis dœoptimiser le temps dœexécution des applications. Nous avons également pu définir des relations entre les différentes métriques de manière à aider dans les choix de compromis entre les métriques lors du paramétrage des architectures.

ABSTRACT

International Technology Roadmap for Semiconductors (ITRS) predicts that CMOS devices will reach their limits in 2022. Consequently, new devices and more efficient technologies are required. In this context, many efforts have been made to extend or replace conventional, CMOS devices. Some devices based on Field Effect Transistor (FET) nanotechnology such as the Dual Gate Carbon NanoTube FET (DG-CNTFET), the Nano Wire FET (NWFET) or the Grapheme FET (GFET) are promising candidates to replace CMOS devices. They lead to define new paradigm of non-conventional architectures (so called *nano-component architecture*). Nano-component architectures have three main characteristics:

(i) The logic cells are dynamically reconfigurable. This characteristic allows performing pipeline on several different functions;

(ii) The granularity is ultra-fine (at most 2-bit operation). This characteristic implies to take into consideration scalability to exploit those architectures;

(iii) The logic cells are organized with hierarchical structure and connectivity restrictions. In this structure, cells are organized in matrix and the matrices are organized in cluster.

Exploiting those characteristics, nanoï architecture are expected, compared to conventional architectures, to reduce the area and the cost and to improve the performance of a broad range of applications. In order to explore the potential of nano-architecture, new CAD tools are required. Those tools must take into account many parameters in nano-architecture definition: the number of cell in matrices, the number of matrices in cluster, the hierarchical structure, the connectivity restrictions, the fine granularity, the high reconfiguration, the pipeline and parallel executioné Although many CAD tools defined for conventional architecture have been proposed, they do not take into consideration nano-architecture parameters.

This research project aims to explore the system-level potential for nano-architectures; more specifically the aim is to define a mapping model that enables:

(i) An automatic application partitioning and mapping for nano-architecture;

(ii) The exploration of the design space defined by the nano-architecture parameters;

(iii) The optimization of several metrics: the computation speed, the communication cost, the reconfiguration cost and the number of non-used logical cells.

The proposed model is defined using the theory of graph and genetic algorithm. It is performed in two main parts accomplished through three complementary methods. In the first part, the application to map is partitioned into a set of sub-functions using a *partitioning method*. Each sub-graph resulting from the partitioning is mapped onto a matrix using a *matrix mapping method*. The second part of the proposed model, maps the set of sub-functions onto the cluster of matrices using a *global mapping method*.

In this work, seven test cases have been studied to validate the mapping model. Our results illustrate that application partitioning allows exploiting, in average, 60% of 16 cell matrix. We also observe that the application of global mapping enable a great optimization of the metrics (it gives the best execution time). Finally, the relations between metrics have been discussed in order to make a good compromise when performing application mapping onto nano-architectures.

TABLE DES MATIERES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ADD8/ADD16	Additionneur 8 bits/16 bits
ADSU8/ADSU16	Additionneur/soustracteur 8 bits/16 bits
AG	Algorithme Génétique
ALU2	Arithmetic and Logic Unit 2-bits
CAO	Conception Assistée par Ordinateur
CMOS	Complementary Metal Oxide Semiconductor
CNT	Carbon NanoTube
CNT_DRC_7T	Dynamic Reconfigurable Cell 7 Transistors
COMPM8/COMPM16	Comparateur 8bits/16bits
DAG	Directed acyclic graph
DG_CNTFET	Dual_Gate Carbone NanoTube Field Effect Transistor
ESP	Evolution Simulation Placement
FAFPGA	Genetic Algorithm for FPGA
FPGA	Field-Programmable Gate Array
GASP	Genetic Algorithm Standard cell Placement
GFET	Graphene Field Effect Transistor
ITRS	International Technology Roadmap for Semiconductors
NSGA	Non-dominated Sorting Genetic Algorithm
NSGA-II	Fast Non-dominated Sorting Genetic Algorithm
NWFET	Nano Wire Field Effect Transistor
SDK	Sun Development Kit
SBX	Simulated Binary Croisement
VEGA	Vector Evaluated Genetic Algorithm
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VPR	Versatile Packing, Placement and Routing

LISTE DES ANNEXES

INTRODUCTION

En 1965, Gordon Moore a énoncé que le nombre transistors dans les circuits intégrés doublera tous les 12 mois (Moore, 1965). L'énoncé de Moore s'est avéré justifié puisqu'on a assisté ces dernières décennies à une évolution technologique exponentielle. Ce phénomène est caractérisé par deux variables : le coût et le nombre de transistors (Hutcheson, 2009). D'un point de vue économique, on s'attend toujours à des composants de plus en plus performants et miniaturisés sans qu'il y ait une inflation de prix. Ce qui conduit non seulement à une grande production de transistors, mais aussi à leur miniaturisation pour pouvoir les exploiter à grand nombre. En effet, le transistor CMOS a franchi 12 ordres de grandeur physique depuis la création de l'industrie (Hutcheson, 2009). Dans ce contexte de miniaturisation, l'ITRS (International Technology Roadmap for Semiconductors) prévoit la limite des transistors CMOS à 2022 (ITRS, 2007). Pour atteindre la performance et la miniaturisation nécessaires aux technologies futures, il est important de définir de nouvelles technologies qui permettront de compléter voire même remplacer le transistor CMOS. Pour ce faire, plusieurs domaines sont en cours d'exploration. Le domaine de la nanotechnologie semble être l'un des plus promoteurs. Il a permis de définir de nouveaux transistors miniaturisés, notamment le transistor à base de graphène (GFET), le transistor à base de nanotube de carbone (DG-CNTFET), le transistor à base de nano fil (NWFET)é Ces transistors ouvrent la voie à un nouveau paradigme d'architectures non conventionnelles à granularité fine : *les architectures nano-composantes*. Ces architectures ont trois caractéristiques spécifiques :

- Les cellules logiques faites à base de nano-transistors sont dynamiquement reconfigurables c.-à-d. qu'elles peuvent être reconfigurées lors de l'exécution;
- Les architectures sont caractérisées par une granularité fine, c.-à-d. les cellules logiques traitent des opérations logiques prenant deux entrées codées sur un bit chacune;
- Les architectures ont une structure particulière à deux niveaux : les cellules sont organisées en matrices et sont connectées entre elles par des interconnexions fixes. Les matrices, à leur tour, sont organisées en réseau de matrices et communiquent entre elles par des connexions dynamiques.

Afin de prouver le potentiel des architectures nano-composantes pour la conception des systèmes électroniques performants, une évaluation exhaustive s'impose. Ceci implique des outils permettant l'exploration automatique de l'espace de conception.
De plus, étant donné que ces technologies sont très nouvelles l'accès et le coût pour le prototypage représentent des réels défis.
Pour résoudre ces problèmes, la définition des outils de conception assistée par ordinateur (CAO) est incontournable. Ces outils devront relever plusieurs défis :

- Accélérer la mise sur le marché des nouvelles technologies en réduisant le temps de conception;
- Faciliter l'accès au prototypage en permettant les prototypages virtuels;
- Faciliter la synthèse physique et garantir des systèmes de faible consommation;
- Permettre une bonne interaction entre les concepteurs niveau systèmes et les concepteurs niveau physique.

La conception des outils CAO, dans le domaine de la nanotechnologie, devra aussi prendre en compte les effets physiques liés à la réduction des dimensions et également définir des stratégies d'intégration et de traitement de l'information pour les nano-systèmes (Robert, 2004).

L'objectif principal de ce projet est de proposer une approche pour aider à la conception des systèmes basés sur les architectures nano-composantes. Plus spécifiquement, il s'agit de définir un modèle de placement qui permettra d'exécuter des applications sur les architectures nano-composantes. Le modèle doit prendre en compte les caractéristiques de ces architectures. Il doit également permettre d'optimiser leurs ressources tout en exploitant les performances qu'elles offrent.

Ce travail a principalement permis de développer un des premiers modèles de placement prenant en considération les caractéristiques des architectures nano-composantes. Plus spécifiquement il a permit de :

- Définir une méthode de partitionnement, utilisant la théorie de graphe, qui permet de diviser automatiquement une application en sous-fonctions;

- Développer une méthode d'assignation matricielle, utilisant la théorie de graphe, qui permet de placer une sous-fonction sur une *matrice*;
- Définition d'une méthode de placement globale, utilisant les algorithmes génétiques, pour permettre un placement global de toutes les sous-fonctions sur *le réseau de matrice*;
- Exploiter les performances de ces architectures tout en optimisant l'utilisation des ressources ;
- Faciliter l'exploration pour la définition des systèmes nano-composants.

Ce travail marque son originalité non seulement dans la nouveauté du domaine et du paradigme sur lequel nous travaillons, mais aussi sur les contributions qu'il apporte dans ce domaine. En effet ce projet de recherche contribue à la définition de nouvelles architectures candidates au remplacement des architectures conventionnelles (CMOS). Il fournit un outil qui aidera au prototypage de ces architectures tout en permettant leur exploitation.

Ce mémoire est organisé en quatre chapitres. Le premier chapitre présente les différentes techniques de placement, les travaux existants sur les architectures nano-composantes ainsi que le positionnement de notre projet par rapport à ces travaux. Le deuxième chapitre décrit les caractéristiques des architectures nano-composantes. Il décrit également les concepts de bases sur la théorie des graphes et les algorithmes génétiques exploités dans ce mémoire. Le modèle de placement ainsi que les différentes méthodes qui le composent sont expliqués dans le chapitre trois. Enfin le quatrième chapitre, présente l'implémentation et les résultats obtenus.

CHAPITRE 1. REVUE DE LITTÉRATURE

Dans ce chapitre, nous présenterons les travaux existants sur les méthodes de placement pour des architectures à granularité fine (reconfiguration niveau porte logique). Il existe plusieurs techniques de placement définies pour ce type d'architectures (heuristiques spécifiques). Parmi ces méthodes, on peut citer la méthode de constructions incrémentale et la méthode de placement par partitionnement (min-cut).

- **La construction incrémentale** est une méthode intuitive qui consiste à associer itérativement les opérations logiques (nœuds, modules) d'une application aux cellules logiques (locations) de l'architecture;
- **Le placement par partitionnement** consiste à réduire le problème de placement en partitionnant l'application à placer en sous-fonctions puis à effectuer le placement de ces sous-fonctions. Les méthodes de placement par partitionnement diffèrent, les unes des autres, par les algorithmes de partitionnement utilisés. Dans ce chapitre nous allons parler des algorithmes de partitionnement utilisés par les méthodes de placement par partitionnement.

Par ailleurs, le problème de placement étant un problème NP-Complet, les paradigmes d'optimisation générale permettent d'avoir de bons résultats de placement. Les plus utilisés pour le placement sont le recuit simulé et les algorithmes génétiques.

- **Le recuit simulé** permet une méthode de recherche locale qui part d'une solution initiale qu'elle améliore en plusieurs itérations;
- **Les algorithmes génétiques** permettent une recherche globale. Il utilise une approche qui part d'un ensemble de solutions qui sont diversifiées afin de rapprocher de la solution optimale.

Toutes ces techniques sont complémentaires et peuvent être combinées dans les méthodes de placement. Dans ce chapitre on présente chaque méthode de placement existante, à travers la technique qui la caractérise le mieux. La figure1.1 décrit la relation entre les travaux existants et les travaux effectués dans ce mémoire. L'organisation de ce chapitre y apparait également. La première section présente les travaux existants sur les heuristiques spécifiques dédiées aux architectures à grain fin. Les méthodes existantes basées sur les paradigmes d'optimisation générale sont présentés dans la seconde section. La troisième section est consacrée aux travaux

sur lesquels nous nous sommes basés ainsi quôà la situation du modèle de placement proposé parmi les méthodes de placement existantes. La quatrième partie résume le chapitre.

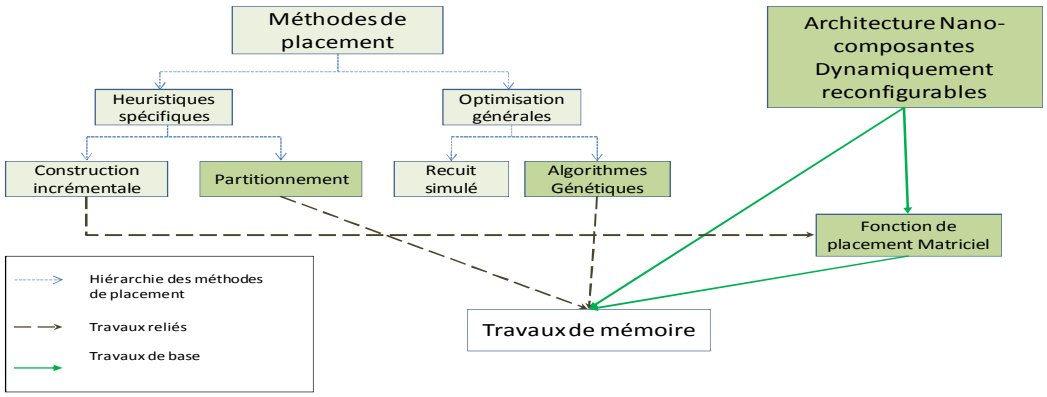


Figure 1. 1 Situation des travaux du mémoire

1.1 Les heuristiques de placement spécifiques aux architectures à grain fin

Cette section présente les méthodes de placement dédiées aux architectures à grains fins. La première sous-section présente les méthodes basées sur la construction incrémentale alors que la seconde sous-section présente les algorithmes de partitionnement utilisés dans les méthodes de placement par partitionnement.

1.1.1. La construction incrémentale (cluster growth)

Hanan et al (Hanan, et al., 1978) ont présenté une méthode de placement qui est incrémentale. Elle sélectionne les modules (nî uds ou opérations logiques de base) un à la fois en se basant sur une fonction dôévaluation qui permet de déterminer quel module sélectionné et où le placer. Une fois que le module est sélectionné et positionné, il ne peut pas être déplacé. La fonction de sélection définie dans cette méthode consiste à choisir le module qui a le plus de connexions avec les modules déjà placés. Ce processus est répété jusqu'à ce que tous les modules soient placés.

Ramineni et al (Ramineni, et al., 1993) ont proposé une méthode de placement incrémentale dont la fonction de sélection est basée sur la possibilité de routage. La méthode était adressée à des architectures spécifiques ayant une restriction de connectivité (Concurent logic

CLi6000/AMTEL). Dans cette méthode, un module est sélectionné s'il existe une connectivité pour le placer.

Kang (Kang, 1983) a proposé une méthode de placement basée sur un ordonnancement de tous les modules de la fonction logique. Une liste des modules, pas encore placés, est créée de manière incrémentale par ordre décroissant de coût de connexions. À chaque itération, l'ordonnancement est mis à jour jusqu'à ce que tous les modules soient placés. Cette méthode est intéressante dans le sens où elle facilite la sélection du module à placer.

Résumé : La méthode de construction incrémentale est très rapide. Elle donne de bons résultats pour le placement de petites applications, mais plus l'application est grande plus le résultat n'est pas nécessairement optimal.

1.1.2. Le placement basé sur le partitionnement

La méthode de placement par partitionnement consiste à diviser l'application à placer en ensemble de modules en leur réservant de l'espace sur l'architecture physique. Chaque ensemble de modules qui ne peut pas être placé est divisé récursivement jusqu'à ce qu'on puisse placer toute sous-fonction résultante du partitionnement. Une fois que le partitionnement est fait, on peut choisir n'importe quelle technique de placement pour placer les sous-fonctions qui vont devenir les unités atomiques à placer. On parle en général de méthode de placement, mais dans cette approche, les performances de la méthode dépendent beaucoup de l'algorithme de partitionnement utilisé. Dans cette sous-section, nous présentons différents algorithmes de partitionnement utilisés lors du placement par partitionnement.

Kernighan et al (Kernighan, et al., 1970) ont défini une méthode de partitionnement dont l'objectif est de réduire le coût de connexion et de routage de données. Ils ont défini un bloc comme un ensemble de modules. L'algorithme effectue un premier partitionnement aléatoire qui est amélioré pas la suite en utilisant une méthode itérative de changement de modules entre les blocs. Le changement de modules consiste en une permutation des modules entre deux blocs c.-à-d. si on a deux modules n_1 et n_2 et que n_1 se trouve dans le bloc b_1 et n_2 dans le bloc b_2, le changement de modules consiste à mettre n1 dans b_2 et n_2 dans b_1.

L'auteur de (Breuer, 1977) a proposé une méthode de placement basée sur le partitionnement de Keringhan et coll. Le bloc peut être partitionné par une coupure horizontale

ou verticale. Le partitionnement de Keringhan a été modifié de manière à prendre en compte deux nouveaux aspects :

- Certains modules ne peuvent pas être déplacés lors de la phase d'amélioration;
- Au lieu de partitionner une connexion entre deux modules, le partitionnement se fait entre deux blocs.

Dans leurs travaux, Fiduccia et al (Fiduccia, et al., 1982) ont défini une méthode itérative pour améliorer l'algorithme de Keringhan. Ils ont rajouté un nouveau critère qui consiste à assurer l'équilibre entre les deux blocs partitionnés (ensemble de modules dans chaque bloc). Ils définissent un concept de « gain de module » qui est utilisé pour déplacer un module d'un bloc à un autre. La valeur de ce gain est calculée par rapport au critère d'équilibre et au nombre d'interconnexions.

Cox et al (Cox, et al., 1986) ont défini une méthode de placement spécifique à l'architecture STAR (Standard Transistor Array). Cette architecture a été définie par Gould et al (Gould, et al., 1980) . Ils la présentent comme un ensemble de transistors CMOS superposés dans un réseau. 22 cellules standards qui facilitent la conception de circuit à partir de cette architecture y ont été définies. L'objectif de placement sur ces architectures est de réduire le nombre de connexions et le coût de routage. Pour cela, la méthode utilise l'ordonnancement linéaire pour partitionner les fonctions. Cela permet de bien organiser les modules d'un même bloc de manière à ce qu'ils soient placés dans le plus petit bloc physique possible. Cet ordonnancement doit aussi permettre de réduire le nombre de connexions et réduire le coût du routage.

Les algorithmes de partitionnement présentés ci-dessus prennent en compte la connexion entre deux modules ou deux blocs (ensemble de modules), mais ne prennent pas en compte le sens de la connexion. En effet le sens de la connexion est très important, car il permet d'éviter les partitionnements cycliques. Lorsque deux blocs résultant d'un partitionnement forment un cycle, cela rajoute beaucoup les communications et donc le coût de routage de données. Ce qui n'est pas souhaité, car l'un des plus grands objectifs du placement est de réduire le routage de données.

Les auteurs de (Cong, et al., 1994) et (Cong, et al., 1995) ont proposé une méthode de partitionnement qui exploite la dépendance logique afin d'éviter les retours cycliques après le partitionnement. Dans cette méthode, le partitionnement se fait dès le départ en plusieurs blocs

(au lieu de deux blocs dans les méthodes précédentes). Ils utilisent également la méthode de changement de modules pour améliorer l'algorithme défini. À ce niveau, même si un échange permet d'améliorer un partitionnement en termes de nombre de connexions, il n'est valide que s'il ne crée pas de cycles entre les partitions.

Résumé : Les méthodes de partitionnement donnent de bons résultats, cependant, elles peuvent être couteuses en temps d'exécution puisque d'une part elles utilisent une fonction de sélection qu'elles évaluent à chaque itération et de l'autre, elles utilisent des méthodes d'inter-changes pour améliorer le partitionnement. L'avantage de l'utilisation du partitionnement pour le problème de placement est qu'il permet d'éviter les problèmes d'explosion combinatoire puisque les placements se font à un niveau de granularité fine. Ils sont beaucoup utilisés lors de placement basé sur les paradigmes d'optimisation générale puisqu'ils permettent de réduire le problème et de réduire ainsi l'espace de recherche.

1.2 Le placement basé sur les paradigmes d'optimisation générale

Les méthodes d'optimisation métaheuristique permettent d'optimiser une large gamme de problèmes différents avec des recherches itératives successives. Ces approches peuvent prendre beaucoup de temps d'exécution, mais donnent d'excellents résultats (Shahookar, et al., 1991). Les paradigmes les plus utilisés sont ceux du recuit simulé et de l'algorithme génétique.

1.2.1. Le placement basé sur le recuit simulé

Cette approche est beaucoup utilisée pour le placement. L'idée de base est que l'algorithme prend en entrée un placement initial et essaye de le modifier en faisant des inter-changements entre les modules. Un paramètre T appelé température, qui tend vers zéro avec le temps, est utilisé pour déterminer la probabilité d'un mauvais mouvement.

Dans leurs travaux, Sechen et al (Sechen, et al., 1986), (Sechen, 1986) ont développé, Timberwolf, un outil de placement et routage très performant et très utilisé pour les architectures à cellules standards. L'objectif du placement est de minimiser le coût d'interconnexion pour le routage des données. Le placement des cellules est simulé de manière à minimiser les liens d'interconnexion (en estimations) en utilisant un algorithme de recuit simulé. À chaque itération on modifie le placement soit en inter-changeant deux modules ou en changeant de location

(cellule) à un module. La fonction objective est composée de deux valeurs indépendantes : la première est le coût d'interconnexion et la seconde une évaluation de pénalités qui consiste à identifier si la location accordée à une cellule est adéquate ou non.

Les auteurs de (Betz, et al., 1997) ont défini un outil VPR (Versatile Packing, Placement and Routing) pour les FPGA dont la méthode de placement est basée sur l'approche du recuit simulé. L'objectif est de minimiser le coût de routage de données. Ils ont exploré plusieurs fonctions objectives et ont misé sur une fonction qui fait la sommation du coût de toutes les interconnexions du circuit. La recherche par recuit simulé commence par un placement généré aléatoirement. À chaque itération, une permutation des modules, déjà placés, est effectuée. Le paramètre température est réévalué lorsqu'on constate que l'algorithme converge lentement. Ce qui permet d'améliorer la méthode en réduisant de temps d'exécution.

Wakabayashi et al (Wakabayashi, et al., 2002) ont développé un placement basé sur le partitionnement et l'approche de recuit simulé. L'objectif est de faire un placement qui permettra de regrouper au maximum les modules afin de réduire le coût d'interconnexion et d'optimiser l'utilisation de l'espace dans l'architecture. L'approche de recuit simulé est utilisée pour placer des blocs sur des modules physiques réservés. Elle consiste à la permutation des différents blocs afin de trouver le placement ayant le moins de coûts en connexions.

Résumé : Les méthodes de placement par recuit simulé donnent d'excellents résultats. Cependant, elles sont très couteuses en temps d'exécution. Aussi le recuit simulé fait évoluer une seule solution. Lorsque le problème de placement est multi-objectif, le recuit simulé fournit alors comme résultat une seule solution en essayant de l'optimiser. Ce qui représente un inconvénient puisqu'il existe dans ce cas une multitude de solutions qui ne se dominent pas (compromis entre les objectifs).

1.2.2. Le placement basé sur les algorithmes génétiques

Les algorithmes génétiques sont beaucoup utilisés pour les problèmes d'optimisations. L'idée de base est de suivre la loi de l'évolution naturelle. Chaque solution est représentée comme un individu et l'ensemble de solutions, comme une population. À chaque itération des opérateurs génétiques (sélection, croisement, mutation) sont utilisés pour diversifier la

population. La méthode de placement se diffère par la formulation du problème de placement et le choix des opérateurs.

Kling et al (Kling, et al., 1987) ont défini un outil de placement ESP (Evolution Simulation Placement) pour les architectures à cellules standards. L'objectif du placement est de réduire le coût d'interconnexions. L'algorithme commence par définir un placement initial. Chaque solution représente un placement (association modules-locations). La fonction objective associée à chaque placement est le coût d'interconnexion.

Shahookar et al (Shahookar, et al., 1990) ont développé GASP (Genetic Algorithm Standard cell Placement), un algorithme génétique pour résoudre le problème de placement. L'objectif du placement est aussi de réduire le coût d'interconnexions. Il diffère de l'ESP par les opérateurs utilisés et la manière de gérer le placement. La population initiale de l'algorithme génétique est définie aléatoirement. Chaque individu de la population est représenté par un tableau dont chaque élément contient un ensemble de quatre entiers (numéro de cellule, coordonnée de placement, numéro séquentiel). La fonction d'évaluation est le coût de connexion.

Les auteurs de (Venkataramana, et al., 1993) ont défini un algorithme génétique pour le partitionnement de circuits à placer sur FPGA (FAFPGA). Étant donné un circuit logique, l'objectif est de le partitionner en sous-circuits de manière à ce que le nombre d'entrées pour chaque sous-circuit ne dépasse pas une certaine valeur k et le nombre de dépendances entre les sous-circuits est minimisé. La population initiale est générée aléatoirement et chaque individu de la population est représenté sur deux chaines de caractères. La première représente le point de partitionnement et la seconde l'état du bloc. Deux fonctions objectives sont utilisées pour évaluer les deux objectifs.

Résumé : Le placement par algorithme génétique donne d'excellents résultats, même lorsque le problème est multi-objectif, il permet d'avoir des solutions globales avec des compromis. Cependant, il peut être couteux en temps d'exécution. Les auteurs de (Kling, et al., 1987) ont comparé l'outil ESP (algorithmes génétiques) à Timberwolf3.2, les résultats ont montré que pour la même qualité de placement l'outil ESP est plus rapide. La comparaison de Timberwolf3.2 et de GASP a montré qu'ils utilisaient globalement le même temps d'exécution pour la même qualité de placement.

Le tableau 1.1 présente une analyse comparative des différentes techniques présentées. Trois métriques ont été comparées : la qualité des résultats (optimalité), la capacité de la technique à réduire le risque d'explosion combinatoire (extensibilité) et la capacité de la technique à gérer les problèmes multi-objectifs (multi-objectifs). Ces métriques sont très importantes pour la définition du modèle de placement. Les algorithmes génétiques sont les plus adaptés pour les problèmes multi-objectifs. Les meilleures solutions de placement sont trouvées au niveau du recuit simulé et des algorithmes génétiques. Lorsqu'on parle d'architecture à grain fin, il faut toujours prévoir des risques d'explosion combinatoire. La méthode de partitionnement est la plus adaptée pour réduire ce risque.

Tableau1. 1 Analyse comparative des techniques de placement

Technique de placement	Optimalité	Extensibilité	Multi-Objectif
Construction incrémentale	**	*	*
Partitionnement (Placement)	**	***	*
Recuit simulé	***	*	**
Algorithmes génétiques	***	*	***
* Pauvre	** Moyen	*** Excellent	

1.3 Travaux de recherches et méthodes de placement existantes

Cette section est composée de deux sous-sections. La première présente les travaux qui ont permi de définir les architectures nano-composantes. Dans la seconde sous-section, on situe le modèle de placement proposé dans ce mémoire par rapport aux travaux existants.

1.3.1. Travaux de base

Nos travaux de recherches sont basés sur ceux de O'Connor et al (Liu, et al., 2007) , (O'Connor, et al., 2008). Ils ont défini des architectures nano-composantes à granularité très fine et à caractère dynamiquement configurable. Ces caractéristiques ont été obtenues en utilisant des transistors à nanotube de carbone deux grilles (DG_CNTFET : Dual_Gate Carbone NanoTube Field Effect Transistor). Il a été utilisé pour définir une famille de cellules nano-composantes

dynamiquement reconfigurables. Ces cellules ont été regroupées dans une structure hiérarchique à deux niveaux pour donner les architectures nano-composantes. Dans le premier niveau, les cellules sont regroupées dans une matrice et interconnectées avec des connexions fixes, dans le second niveau les matrices sont regroupées dans un réseau de matrices dynamiquement interconnectées. Une méthode a été définie (O'Connor, et al., 2008) pour permettre de placer une fonction sur une matrice. Cette méthode consiste à transformer structurellement la fonction de manière à ce quêelle sôadapte à la structure de la matrice sur laquelle elle va être placée puis de faire lôassignation de cellules aux modules de la fonction. Ils utilisent la méthode de construction incrémentale pour ce placement. La méthode définie cible seulement les fonctions qui peuvent être placées sur une matrice. Dans ce travail, on sôintéresse à comment placer des applications complexes sur ces architectures.

1.3.2. Situation du modèle de placement

Parmi les méthodes de placement présentées ci-dessus, certaines ciblent les architectures à cellules standards et dôautres, les architectures à cellules programmables. Les premières caractérisent lôarchitecture physique comme un ensemble de locations interconnectées sans prendre en compte plus de caractéristiques structurelles. Les méthodes qui ciblent les architectures FPGA prennent en compte plus de caractéristiques sur les architectures. Cependant, ces caractéristiques sont moins restrictives que celle des architectures nano-composantes que nous ciblons. Dans les architectures nano-composantes les cellules sont connectées de manière statique selon une topologie définie. Aussi la granularité très fine de ces architectures est due au fait quêune cellule logique traite seulement les opérations qui prennent des entrées codées sur 1 bit. Nos travaux sont alors différents de ceux cités plus haut puisquêils ciblent les architectures nano-composantes.

Ainsi, le modèle de placement défini dans nos travaux doit prendre en compte les caractéristiques de reconfiguration dynamique et la structure de ces architectures. Il prend également en compte le concept dôexécution en pipeline (possible grâce à la caractéristique de reconfiguration dynamique) et en parallèle des matrices de lôarchitecture (possible grâce à la structure de lôarchitecture). Les objectifs du placement sont de réduire le coût de communication, le nombre de cellules inactives, le coût de configuration et le temps dôexécution. Il sôagit dôun

problème de placement multi-objectif et les méthodes les plus adaptées, pour le résoudre, sont celles des paradigmes d'optimisations métaheuristiques.

Dans les travaux cités plut haut, on a discuté de deux paradigmes d'optimisation générale : le recuit simulé et les algorithmes génétiques. Il existe cependant plusieurs méthodes d'optimisations pouvant être utilisées pour les problèmes multi-objectifs. Les plus utilisées sont regroupées en deux classes :

- **Les méthodes de recherches locales avec la notion de voisinage et de mouvement** Il s'agit du recuit simulé, la recherche Tabou, recherche par escalade (Hill climbing). Ces méthodes traitent une seule solution en essayant de l'optimiser, lorsqu'il s'agit de problèmes multi-objectifs, il est plus pertinent d'avoir un ensemble de solutions qui permettront de faire des compromis entre les objectifs. Le recuit simulé a fait ses preuves dans le domaine du placement, mais compte tenu de cet inconvénient, il ne représente pas la meilleure solution pour notre problème;
- **Les méthodes de recherche globale basée sur les populations** Il s'agit des algorithmes génétiques, des colonies des fourmis, des essaims particulaires, etc. Ces méthodes traitent un ensemble de solutions et permettent l'utilisation du concept de front de pareto. La colonie de fourmis et les essaims particulaires se basent sur le concept d'auto-organisation (la majorité se dirige vers les bonnes solutions) ces méthodes peuvent facilement converger vers un optimum local ou un état bloqué puisque le résultat suit l'évolution de la masse.

La figure 1.2 illustre la classification des méthodes d'optimisation citées plus haut. Nous croyons que la méthode idéale qui permettra de faire un placement multi-objectif, en gérant un ensemble de solutions et en évitant les états bloqués, est celle des algorithmes génétiques. Elle a déjà fait ses preuves pour la résolution des problèmes de placement et offre un grand espace de solutions tout en permettant d'utiliser le concept de front de pareto.

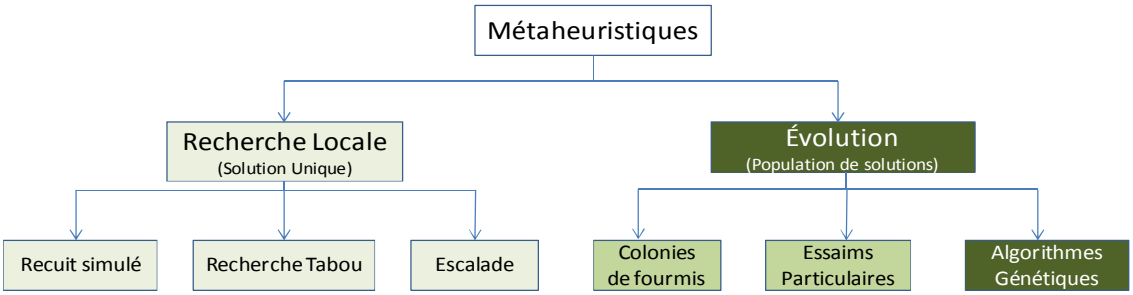


Figure 1. 2 Méthodes métaheuristiques

Néanmoins, pour le problème de placement traitant des fonctions logiques complexes, il peut y avoir des problèmes d'explosion combinatoire. Le tableau 1.1 montre que le partitionnement permet de réduire le risque d'explosion combinatoire. Pour éviter l'inconvénient d'explosion combinatoire nous allons réduire le problème en partitionnant la fonction à placer en sous-fonctions. Aussi, comme (Cong, et al., 1994), nous considérons le sens des dépendances entre les modules afin d'éviter des retours cycliques.

Le modèle de placement va consister donc dans un premier temps à partitionner une fonction en sous-fonctions acycliques, puis à placer les sous-fonctions sur l'architecture nano-composante. Pour que cela puisse être possible, il va falloir au préalable placer chaque sous-fonction sur une matrice nano-composante. O'Connor et al (O'Connor, et al., 2008) ont déjà défini une fonction à cet effet. Dans ce travail, nous avons repris la même méthode en utilisant la théorie des graphes pour faire la similitude entre la fonction et la matrice et faire la restructuration lors du placement en tenant compte des interconnexions.

Comparer aux modèles de placement présentés plus haut, notre modèle à deux contributions importantes :

- Le modèle de placement prend en compte les caractéristiques des architectures nano-composantes pour y placer des fonctions complexes.
- Le modèle de placement combine les algorithmes génétiques et le partitionnement. Ce qui permet d'avoir de bons résultats tout en réduisant, au préalable, le risque d'explosion combinatoire et les coûts de communications.

1.4 Résumé

Dans ce chapitre, les travaux reliés à notre domaine de recherche ont été présentés. Les méthodes de placements ont été présentées par classes de techniques de placement. La liste des techniques présentées n'est pas exhaustive, mais elle regroupe les principales techniques de placement à savoir la construction incrémentale, le placement par partitionnement, le recuit simulé et les algorithmes génétiques. Les techniques utilisées dans ce travail ont été positionnées par rapport à la littérature existante. Les travaux sur lesquelles est basé notre modèle ont aussi été présentés. Il s'agit d'un nouveau paradigme d'architecture nano-composantes dynamiquement reconfigurables. Ce paradigme va être décrit plus en détail dans le chapitre suivant.

CHAPITRE 2. CONCEPTS DE BASE

Ce chapitre décrit les concepts sur lesquels nous nous sommes basés pour définir le modèle de placement. En occurrence les architectures nano-composantes, quelques concepts de la théorie de graphes et les algorithmes génétiques.

Les méthodes de placement prennent toujours, en entrées, une application et la plateforme d'exécution (architecture). Dans ce travail, les architectures considérées pour le placement sont basées sur un nouveau paradigme d'architectures à granularité très fine et dynamiquement reconfigurables. Ce paradigme d'architecture a été conçu par le groupe « Conception microélectronique hétérogène » de l'Institut de Nanotechnologie de Lyon. Une étude de ces architectures a été faite dans ce travail afin de mieux les comprendre et pour prendre en compte leurs caractéristiques dans le modèle de placement. Par ailleurs, pour des raisons de flexibilité, nous avons modélisé le problème de placement en utilisant des graphes. Plusieurs concepts de la théorie de graphes ont été exploités afin de définir le modèle de placement. Enfin, le problème de placement traité dans ce mémoire est un problème multi-objectif. Pour le résoudre, nous avons choisi le concept des algorithmes génétique qui est bien adapté pour ce genre de problèmes.

Le chapitre est organisé en quatre sections. La première décrit les architectures nano-composantes. Les concepts, de la théorie de graphes, utilisés sont présentés dans la seconde section. La troisième section est consacrée aux algorithmes génétiques. Enfin, un résumé conclut le chapitre.

2.1 Les architectures nano-composantes

Dans le but de poursuivre la loi de Moore, plusieurs dispositifs ont été définis en vue de compléter voire même remplacer les transistors CMOS. Parmi ces dispositifs, ceux de la nanoélectronique semblent être très promoteurs (Transistors à nano fils NWFET, transistors à nanotube de carbone DG-CNTFET,é). Ils ont ouvert la voie vers de nouvelles familles d'architectures nano-composantes à caractères dynamiquement reconfigurable. Dans ce travail, nous allons nous concentrer sur les architectures faites à base de nanotube de carbone double grilles (DG-CNTFET). Les architectures nano-composantes peuvent être définies comme un ensemble de cellules logiques dynamiquement reconfigurables faites à base de transistors à

nanotube de carbone (DG-CNTFET). Cette section décrit ces architectures et leurs caractéristiques.

2.1.1. Le transistor à nanotube de carbone double grille (DG-CNTFET)

Le transistor DG-CNTFET présente une propriété ambivalente. Cela signifie qu'il peut opérer soit en transistor N-type (porteur de charges négatives) ou P-type (Porteur de charges positives). Comme son nom l'indique il est composé de deux grilles (au lieu d'une dans les transistors en général) : une grille aluminium qui est celle qui a été rajoutée et une grille silicium qui est la grille arrière. La grille aluminium est celle qui régit le transport de charge par le nanotube. La grille arrière quant à elle régit le comportement du transistor :

- Lorsque son potentiel est suffisamment négatif, le dispositif fonctionne en P-type ;
- Lorsque son potentiel est suffisamment positif, le dispositif fonctionne en N-type ;
- Lorsque son potentiel est flottant, le dispositif est dans un état d'arrêt.

Cette ambivalence du transistor DG-CNTFET a été exploitée pour définir des cellules logiques dynamiquement reconfigurables.

La figure 2.1 est une image de Lin et al (Lin, et al., 2005) qui présente une vue transversale d'un CNTFET double grilles. La grille aluminium (grille avant/région B) est placée sous le nanotube de carbone entre la source et le drain (région A). La grille silicium est placée à l'arrière.

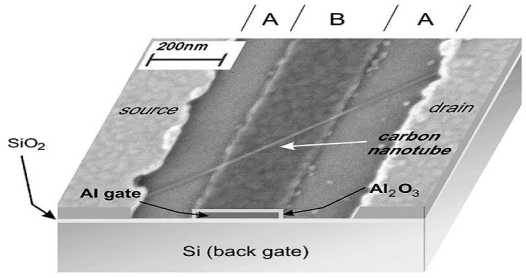


Figure 2. 1 Image de la vue transversale d'un CNTFET à double-grille (Lin, et al., 2005)

2.1.2. Les cellules logiques dynamiquement reconfigurables

Dans leurs travaux, Liu et al (Liu, 2008) ont développé des cellules logiques dynamiquement reconfigurables : CNT_DRC_7T (Dynamic Reconfigurable Cell 7 Transistors). La structure de ce type de cellules est représentée à la figure 2.2. Elle est composée de 7 transistors DG-CNTFETs organisés en deux couches : la première permet de faire une opération logique élémentaire et la seconde sert soit de suiveur ou dôinverseur pour acheminer le résultat. Toutes les variables impliquées dans le fonctionnement de la cellule sont représentées à la figure2.1

- La cellule logique prend deux entrées booléennes A et B sur lesquelles se font une opération logique ;
- Lôopération à faire dépend des trois entrées de contrôle OP1, OP2 et OP3 qui représentent des potentiels des grilles arrières de 3 des 7 transistors ;
- Quatre signaux dôhorloge non chevauchants sont utilisés pour le pré-chargement (PC1, PC2) et lôévaluation (EV1, EV2) ;
- Y représente la sortie du circuit qui peut être dupliquée.

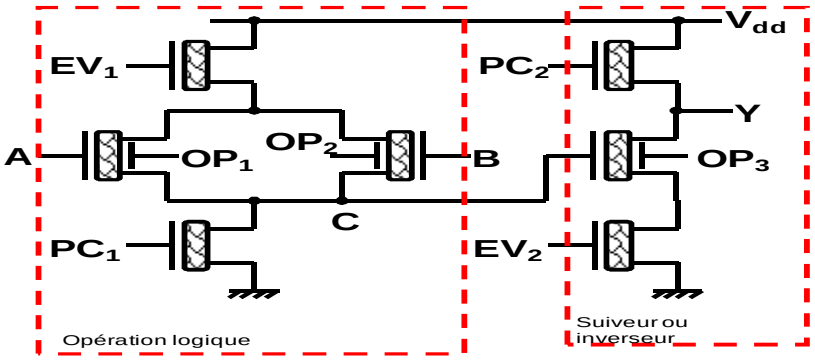


Figure 2. 2 Structure dôune cellule CNT_DRC_7T (Liu, et al., 2007)

Cette cellule peut exécuter 14 opérations logiques différentes selon les valeurs données aux entrées de contrôle pour déterminer le comportement des transistors correspondants : i) +V tension positive pour un comportement P-Type de transistor; ii) 0 tension nulle pour un comportement flottant; iii) ï V négative pour un comportement N-Type. Le tableau 2.1 présente la table de vérité associée.

Tableau 2.1 Table de vérité de la cellule CNT_DRC_7T

Configuration			***Opération***
Op1	*OP2*	*OP3*	**Y**
+V	+V	+V	$\overline{A+B}$
+V	+V	-V	$A+B$
+V	0	+V	$\overline{A}$
-V	-V	+V	$A.B$
-V	-V	-V	$\overline{A.B}$
+V	-V	+V	$\overline{A}.B$
+V	-V	-V	$A+\overline{B}$
0	+V	+V	$\overline{B}$
0	0	0	1
0	0	-V	0
-V	+V	+V	$A.\overline{B}$
-V	+V	-V	$B+\overline{A}$
+V	0	-V	A
0	+V	-V	B

2.1.3. Les matrices à intra-connexions fixes

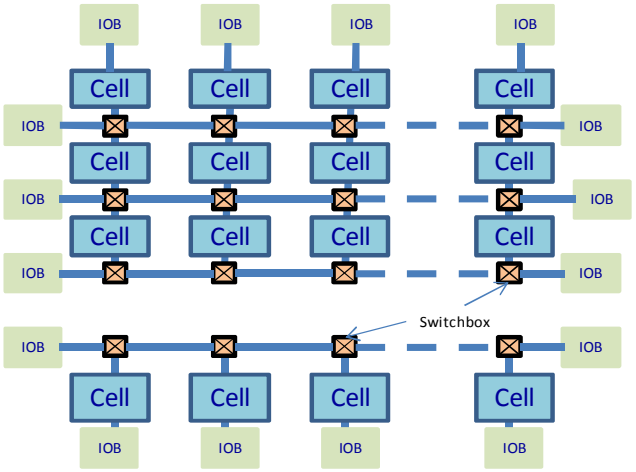


Figure 2. 3 Approche conventionnelle de connexion de cellules

Une des questions encore ouvertes sur les architectures nano-composantes est la manière de connecter les cellules pour les exploiter efficacement. Si lôapproche conventionnelle est

utilisée, les cellules seront connectées entre elles par des broches de connexions (switchbox) comme à la figure 2.3. Cette connexion est couteuse puisqu'on parle de cellules nano-composantes et qu'elles sont relativement petites par rapport aux broches de connexion (Liu, et al., 2007).

Liu et al (Liu, et al., 2007) ont défini une solution pour des cellules DG_CNTFET. Il s'agit d'une approche matricielle où les cellules seront regroupées dans des matrices. À l'intérieur des matrices, les cellules sont connectées avec des connexions fixes. Les matrices quant à elles seront connectées avec les broches de connexion. La figure 2.5 montre la structure de l'approche matricielle. Une matrice est alors caractérisée par ses dimensions *Largeur* × *Profondeur* et le type de topologie qui définit les interconnexions intra-matrice. Chaque cellule est caractérisée par sa position (couche x et la position dans la couche y).

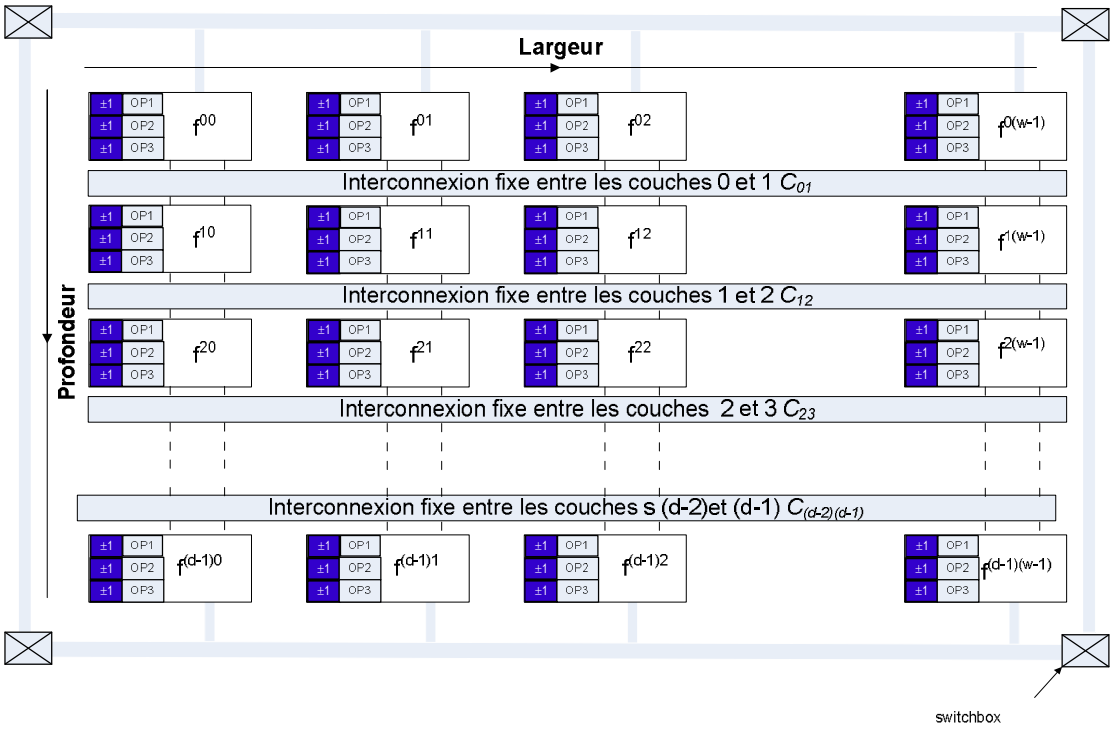


Figure 2. 4 Approche matricielle de connexion des cellules DG_CNTFET

Le symbole de la cellule CNT est représenté dans la figure 2.5. On y met en évidence les entrées booléennes A et B, les entrées de contrôles OP1, OP2, OP3 et la sortie dupliquée Y. f^{xy} représente ses coordonnées (x, y) dans la matrice.

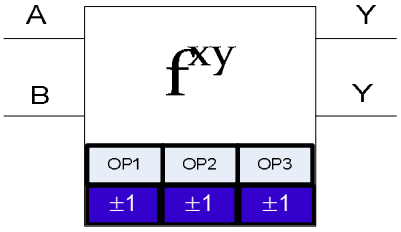
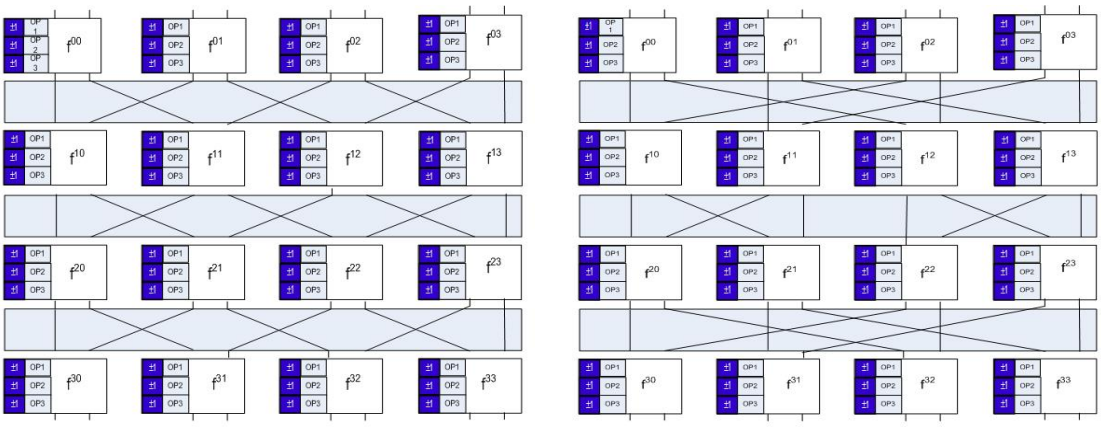


Figure 2. 5 Symbole de la cellule logique DG_CNTFET

Les cellules sont organisées par *couches*, les cellules dûne même couche *i* ne communiquent pas entre elles. Elles sont dépendantes des cellules de la couche supérieure *i+1* (si elle existe) et les cellules de la couche *i-1* (si elles existent) dépendent dœlles.

La figure 2.6 présente des exemples de topologies fixes pour les matrices. Dans la topologie Omega, la cellule f^{01}, par exemple, reçoit toujours les données de f^{00} et f^{02} et transmet ses résultats à f^{20} et f^{22}. Dans la topologie Banyan, la même cellule recevra ses données de f^{01} et f^{03} et transmettra ses résultats à f^{20} et f^{21}.



a. Topologie Omega Modifiée b. Topologie Banyan

Figure 2. 6 Exemple de topologie

2.1.4. Les caractéristiques des architectures nano-composantes

Le paradigme dœarchitectures nano-composantes présente trois principales caractéristiques :

- ✓ Les architectures ont un caractère de reconfigurations dynamiques ultrafines. Cela signifie que les cellules logiques de cette famille d'architecture peuvent être reconfigurées lors des exécutions.
- ✓ La granularité des architectures est très fine, car les cellules logiques traitent des opérations ayant des entrées codées sur 1 bit;
- ✓ La structure des architectures est particulière et favorise l'exécution en parallèle des matrices. Il s'agit d'une structure à deux niveaux (figure 2.6) :
  - ➢ Le niveau cellule (Matrices) : Les cellules sont interconnectées par des connexions fixes ;
  - ➢ Niveau matrice (Réseau de matrices) : Les matrices sont connectées de manières dynamiques.

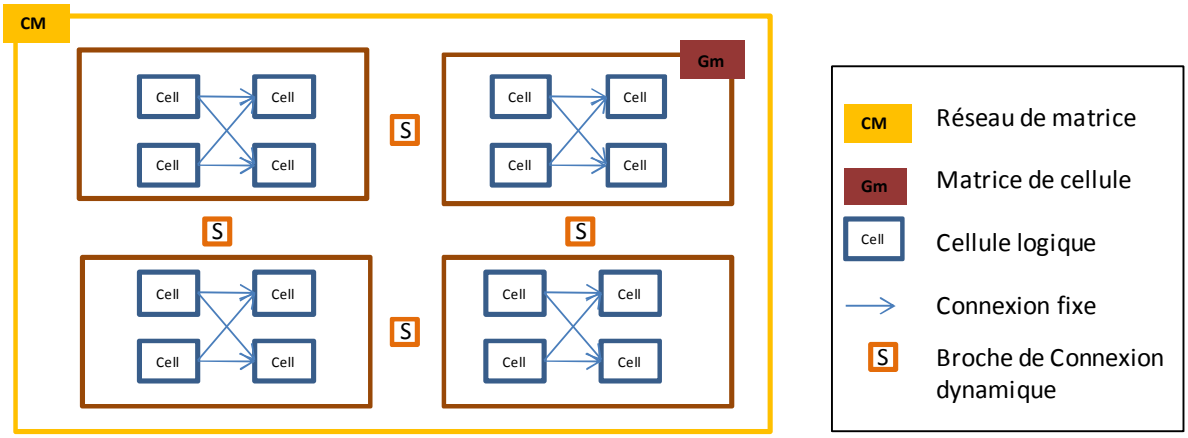


Figure 2. 7 Hiérarchie des architectures nano composantes

Les architectures nano-composantes dépendent alors de trois paramètres : le nombre de cellules dans les matrices, les topologies d'interconnexion dans les matrices et le nombre de matrices dans le réseau.

2.2 Les concepts de théorie de graphes utilisés

La théorie de graphes est utilisée dans divers domaines pour modéliser des situations concrètes comme les interconnexions routières, ferroviaires ou aériennes, les connexions dans les circuits électroniques, les structures chimiques, etc.

Cette section décrit les différents concepts de théorie de graphes utilisés dans le cadre de ce mémoire. La première sous-section définit les graphes acycliques orientés ainsi que leurs utilisations. La seconde sous-section présente deux algorithmes différents pour parcourir un graphe. La dernière sous-section est consacrée aux algorithmes de plus courts chemins.

2.2.1. Les graphes acycliques orientés

Définition 2.1 : Un *graphe* G permet de représenter des objets (nœuds) et leurs relations (arêtes). Il est défini par le couple G = (V, E) tel que :

- V est un ensemble fini de nœuds (sommets) ;
- E est un ensemble d'arêtes E : V×V reliant des nœuds de V.

Définition 2.2 : Un graphe G'= (V', E') est dit *sous-graphe* de G= (V, E) si V' est inclus dans V et E' est l'ensemble des arêtes de E dont les deux nœuds se trouvent dans V'.
La figure 2.8.a présente un exemple de graphe G avec ses nœuds V et ses arêtes E. Un sous-graphe de G est également présenté dans la figure 2.8.b.

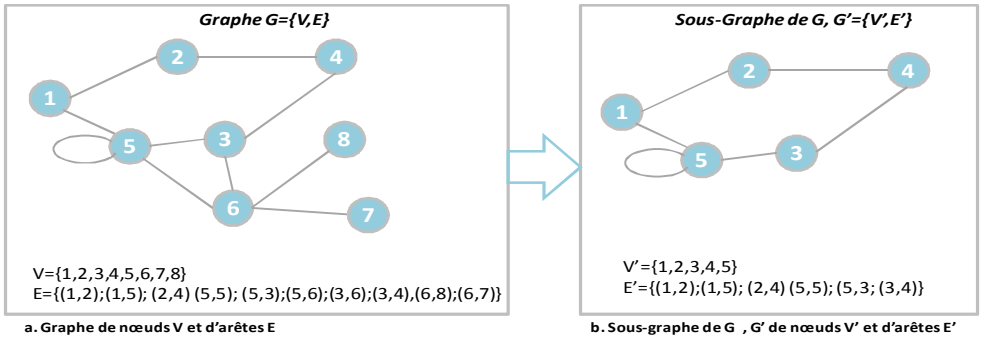


Figure 2. 8 Exemple de graphe non orienté et de sous-graphe

Définition 2.3 : Un graphe G (V, E) est dit *orienté* lorsque, pour une arête (x, y), le sommet y est en relation avec x sans que x soit nécessairement en relation avec y. L'arête est alors appelée ***arc*** (x, y) où le sommet x est l'origine de l'arc et y l'extrémité. La figure 2.9.a présente un graphe orienté cyclique qui contient un cycle élémentaire c.à.d. une boucle d'un nœud sur lui-même ***(5,5).***

Définition 2.4 : Un nœud x est ***voisin*** d'un autre nœud y dans un graphe G=(E, V) si et seulement si, il existe une ***arête*** (x, y) dans V. Par exemple, dans la figure 2.9.b les *voisins* du nœud *6* sont *5-3-8-7*.

Définition 2.5 : Un nî ud x est ***prédécesseur*** dôun autre nî ud y dans un graphe G=(E, V) si et seulement si, il existe un ***arc*** (x, y) dans V. Par exemple, dans la figure 2.9.b le *prédécesseur* du nî ud *6* est *3*.

Définition 2.6 : Un nî ud x est ***successeur*** dôun autre nî ud y dans un graphe G=(E, V) si et seulement si, il existe un ***arc*** (y, x) dans V. Par exemple, dans la figure 2.9.b les *successeurs* du nî ud *6* sont *5-8-7*.

Définition 2.7 : On définit par ***graphe acyclique orienté*** (de lôanglais Directed Acyclic Graph : DAG), un graphe orienté G= (V, E) qui ne contient pas de cycle c.-à-d. pour tout sommet x de V il nôexiste pas des arêtes qui permettent directement ou indirectement de le relier à lui-même. Un graphe acyclique orienté est présenté dans la figure 2.9.b.

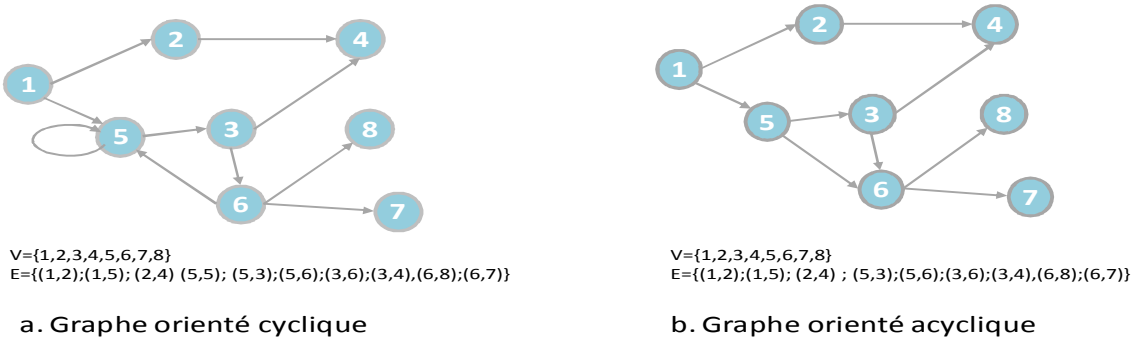


Figure 2. 9 Graphe orienté cyclique et acyclique

La notation DAG est souvent utilisée en informatique pour représenter la théorie de langages, les modèles par couches, les architectures informatiques, les applications informatiques, etc.

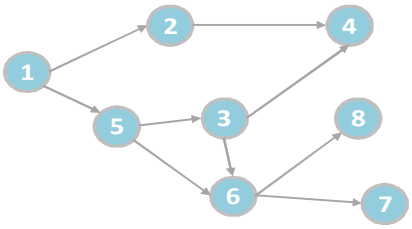
2.2.2. Les algorithmes de parcours des graphes

Un parcours de graphe consiste à explorer le graphe par ses nî uds ou ses arcs et à les ordonner pour effectuer des traitements. Selon lôordre souhaité, plusieurs algorithmes de parcours existent. Les plus courants sont le parcours en largeur et le parcours en profondeur.

2.2.2.1. Le parcours en largeur

Le parcours en largeur dôun graphe G= (V, E) consiste à partir dôun nî ud x de V, de traverser tout nî ud, y de V, directement accessible à partir de x à condition que tous les arcs

entrants à y aient été visités. Le parcours se fait ainsi récursivement jusqu'à ce que tous les nœuds de V soient traversés. La figure 2.10 montre un exemple de solutions possibles du parcours en largeur en commençant par le nœud *1* du graphe. Les nœuds accessibles à partir de 1 sont *2 et 5*, l'itération suivante va « découvrir » les nœuds accessibles à partir de 2 : *4* et à partir de 5 : *3 et 6*. Les successeurs de 3 ayant déjà été explorés, la dernière itération va découvrir les successeurs de 6 : *8 et 7*.

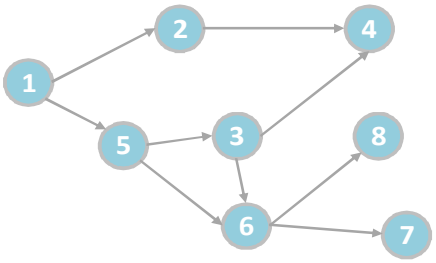


Parcours en largeur:
Solution Possible 1 :{1,2,5,4,3,6,8,7}
Solution Possible 2: {1,5,2,6,3,4,7,8}

Figure 2. 10 Parcours en largeur

2.2.2.2. Le parcours en profondeur

Contrairement au parcours en largeur, le parcours en profondeur explore à fond un nœud x pour découvrir tous les nœuds qui sont accessibles *directement ou indirectement* à partir de x. L'exploration se fait ainsi jusqu'à ce qu'on tombe sur un nœud qui n'a plus de successeurs non explorés, puis on fait un retour en arrière vers le nœud père pour reprendre l'exploration à partir d'un autre nœud. La figure 2.11 illustre un exemple de solutions pour le graphe DAG présenté plus haut. Le parcours commence par le nœud 1, on prend un de ses successeurs le nœud *2* par exemple et on explore aussi les voisins du nœud 4. Le nœud *4* n'ayant pas de successeurs on revient à 1 pour prendre le nœud *5*. On prend un de ses successeurs le nœud *3* par exemple. Le nœud 4 ayant déjà été exploré, on parcourt le nœud *6* puis ses successeurs *8 et 7*.



Parcours en profondeur:
Solution Possible 1 :{1,2,4,5,3,6,8,7}
Solution Possible 2: {1,5,6,7,8,3,4,2}

Figure 2. 11 Parcours en profondeur

2.2.3. Les algorithmes de plus courts chemins

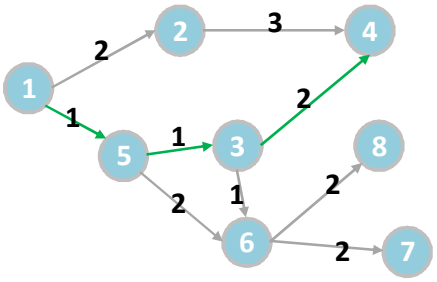
Le cheminement est un problème très courant pour minimiser ou maximiser directement ou indirectement les coûts des fonctions économiques (temps, argents, ressourcesé). On le retrouve notamment dans les domaines de transport, le routage de paquet dans les réseaux, les interconnexions de communications, etc. Lôobjective est de trouver un chemin entre une source et une destination afin dôoptimiser une fonction économique.

Définition 2.8 : On appelle un *chemin* C dans un graphe G= {V, E}, une liste de nî uds appartenant à V, C= {x_1, x_2,é x_k}, telle que pour tout couple (x_i, x_{i+1}), il existe une arête correspondante dans E.

Le plus court chemin entre deux nî uds x_1 et x_k est calculé dans un graphe pondéré c.-à-d. toutes les arêtes du graphe ont une valeur. La longueur dôun chemin est alors la somme des poids des arêtes/arcs qui le composent.

Longueur (C={x_1, x_2,é , x_k}) = $\sum_{i=1}^{k-1}(x_i, x_{i+1})$

Le plus court chemin entre deux nî uds est le chemin dont la longueur est la plus petite. Dans le graphe de la figure 2.12, le plus court chemin entre le nî ud *1* et le nî ud *4* est le chemin de longueur 4 passant par *5*,*3*.



Plus court chemin entre les nî uds 1 et 4
Solution Possible 1 :C={1,5,3,4} de longueur *4*

Figure 2. 12 Plus court chemin

Plusieurs algorithmes ont été définis pour déterminer le plus court chemin entre deux nî uds n_1, n_2 dans un graphe pondéré. Le plus utilisé est celui de Djikstra. L'idée est d'explorer le graphe et de calculer, au fur à mesure, les distances des nî uds parcourus de n_1 jusqu'à atteindre n_2.

Un algorithme appelé *K-plus courts chemins (K-Shortest paths)* permet d'avoir en sortie K différents courts chemins ordonnés par ordre décroissant de longueur de chemins (Lau, 1989). Il donne en sortie K plus courts chemins s'ils existent sinon il donne les plus courts chemins existants. Par exemple dans la figure 2.12, pour les plus courts chemins entre les nœuds 1 et 4, même si le paramètre K est supérieur à 2, on aura en sortie que les deux chemins existants dans l'ordre suivant : {{1, 5, 3,4.} ;{1, 2,4}}.

2.3 Les Algorithmes génétiques

Cette section est consacrée aux algorithmes génétiques. La première sous-section décrit le fonctionnement général des algorithmes génétiques. La seconde est consacrée aux différents aspects d'un problème qui doivent être formulés pour appliquer les algorithmes génétiques. Les opérateurs génétiques sont présentés dans la troisième sous-section. La dernière sous-section décrit la manière dont les algorithmes génétiques gèrent les solutions dans les problèmes multi-objectifs.

2.3.1. Fonctionnement général

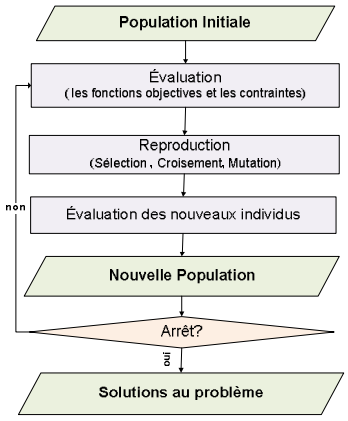


Figure 2. 13 Fonctionnement des algorithmes génétiques

L'algorithme génétique commence avec une population initiale dont les individus sont évalués en fonction des objectifs et des contraintes du problème à résoudre. Cette population est ensuite diversifiée en utilisant des opérateurs génétiques (sélection, croisement, mutation) pour reproduire les individus. La reproduction est faite à plusieurs reprises jusqu'à ce que le critère

dôarrêt (nombre dôitérations en générale) de lôalgorithme soit atteint. La figure 2.13 résume le fonctionnement général des algorithmes génétiques.

2.3.2. Formulation du problème dôoptimisation

Les algorithmes génétiques travaillent sur la formulation dôun problème et non sur le problème lui-même. La formulation consiste en deux étapes essentielles : la représentation des individus et la formalisation des fonctions dôévaluations.

2.3.2.1. Codage des individus

Une population est composée de chromosomes ou individus et chaque individu se caractérise par un ensemble de gènes. Chaque individu représente une solution possible au problème. Représenter un individu revient alors à définir ce qui caractérise une solution possible au problème. Nôimporte quel type de représentation peut être défini à condition quôon sôassure de la définition des opérateurs génétiques pouvant traiter la représentation. Les représentations les plus courantes sont :

La représentation binaire : Il sôagit dôune suite de 0 et 1. Un exemple dôutilisation est dans le cas où chaque gène représente une caractéristique qui peut être présente (1) ou non (0) dans lôindividu. La figure 2.14.a illustre un exemple de deux individus codés en binaire.

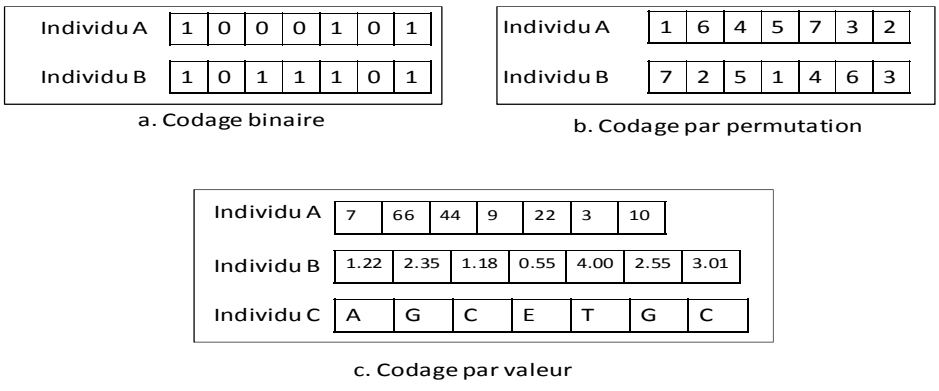


Figure 2. 14 Codage des solutions

La permutation : Il sôagit de la permutation de valeurs entières égales au nombre de gènes dans les chromosomes. Ce codage est beaucoup utilisé pour les problèmes dôordonnancement de ressources. Des exemples du codage par permutation sont présentés dans la figure 2.14.b.

La représentation par valeurs : Dans ce codage on associe à chaque gène une valeur prise dans un ensemble fini ou infini. Ces valeurs dépendent du problème étudié. La figure 2.14.c montre des exemples de codage en entier, réel, et caractère.

2.3.2.2. Formalisation des fonctions d'évaluation

La formulation consiste en l'expression sous forme mathématique des objectifs d'optimisation de la manière la plus fidèle possible. Ces fonctions d'évaluation sont utilisées pour déterminer le degré de pertinence de chaque solution. L'évaluation de chaque solution est indépendante du reste de la population. Elle permet de s'assurer qu'on garde les individus les plus pertinents en éliminant les moins pertinents progressivement de la population.

Dans certains problèmes, les solutions doivent satisfaire des contraintes pour faire parties des solutions finales. Ces solutions sont représentées sous forme de fonctions de contraintes qui permettent de garantir la cohérence de chaque solution.

Le problème est alors formulé comme suit :

- S ={X_1, X_2, X_3,é , X_n} un ensemble d'individus représentant une population. X étant un ensemble de composants X ={x_1, x_2, x_3,é , x_k}
- L'objectif est de minimiser F (X)= {f_1 (X), f_2 (X),é , f_m(X)} pout tout X de S
- Telles que les contraintes G (X)= {g_1 (X), g_2 (X),.. , g_m(X)} soient satisfaites

2.3.3. Opérateurs génétiques

Les algorithmes génétiques utilisent trois opérateurs pour générer de nouvelles solutions :

(i) L'opérateur de sélection qui permet de choisir des solutions parentes sur lesquelles la reproduction va être faite pour générer des nouvelles solutions.

(ii) L'opérateur de croisement qui permet de croiser les deux solutions parentes et créer de nouvelles solutions.

(iii) L'opérateur de mutation qui permet de diversifier les nouvelles solutions afin qu'elles ne ressemblent pas trop aux solutions parentes. L'utilisation des coefficients de probabilistes permet de faire des bonnes diversifications en introduisant l'effet du hasard.

2.3.3.1. L'opérateur de sélection

La sélection consiste à choisir des individus qui permettront de générer de nouveaux individus. Plusieurs méthodes existent pour sélectionner des individus destinés à la reproduction. On citera les deux méthodes classiques les plus utilisées.

La sélection par tournoi : Cette méthode consiste à choisir aléatoirement une paire d`individus dans la population. Le meilleur des deux individus sera sélectionné pour la reproduction avec une probabilité p supérieure à 0.5. Cette méthode est en général satisfaisante. La figure 2.15 illustre ce type de sélection. Les paires d'individus (1,5) et (2,4) sont choisis, puis on sélectionne les individus 5 et 2 pour la reproduction.

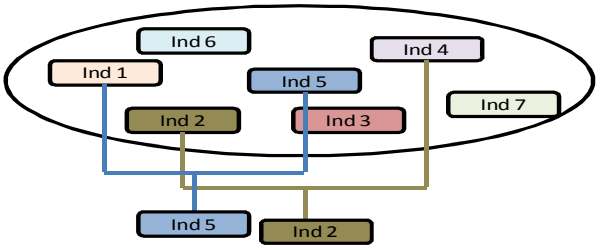


Figure 2. 15 Sélection par tournoi

La sélection par la roulette : Les individus de la population se voient allouer des longueurs proportionnelles à leurs performances. Ils sont représentés dans une roulette dont la surface totale représente les performances des individus et chaque individu est représenté par sa longueur sous forme de portion de roulette. Un nombre tiré aléatoirement permet de déterminer l'individu sélectionné selon la portion de roulette à laquelle il correspond. Cette méthode favorise les meilleurs individus ce qui n'est pas nécessairement souhaité puisque la reproduction avec de mauvais individus peut rapprocher de la solution optimale. La figure 2.16 illustre une population de 7 individus dont les performances sont représentées en roulette.

Individu	Fonction Objective	Longueur
1	20	0.20
2	5	0.05
3	30	0.3
4	10	0.1
5	15	0.15
6	7	0.07
7	13	0.13

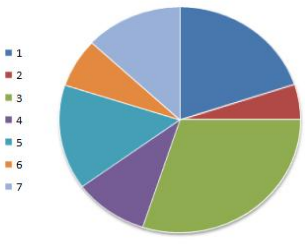


Figure 2. 16 Sélection par roulette

2.3.3.2. Lôopérateur de Croisement

Le croisement consiste, à partir de deux parents P_1 et P_2 sélectionnés, à définir deux nouveaux individus C_1 et C_2. Ce qui permet de diversifier lôespace de solutions. Il existe plusieurs méthodes de croisement.

Le croisement à un point (slicing crossover) : Il a été initialement défini pour le codage binaire. Le principe consiste à tirer aléatoire une position pour chaque parent et à échanger les sous-chaines des parents à partir des positions tirées. Ce qui donne naissance à deux nouveaux individus C_1 et C_2 Figure 2.17.

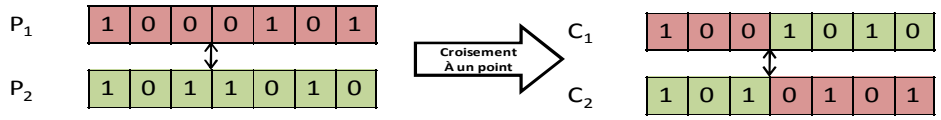


Figure 2. 17 Croisement à un point

Le croisement à deux-points (2-points crossover) : elle reprend le mécanisme de la méthode de croisement à un point en généralisant lôéchange à 3 ou 4 sous chaines (Figure 2.18).

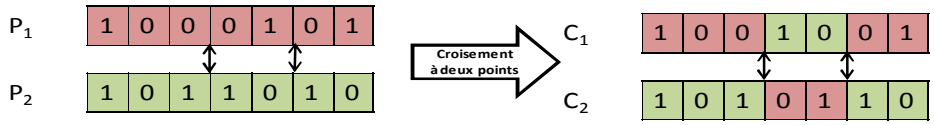


Figure 2. 18 Croisement à deux-points

Le croisement barycentrique : Les méthodes de croisement à un point et deux-points sont très efficaces pour le codage binaire et le codage à variables discrètes comme les permutations. Le croisement barycentrique permet des croisements plus appropriés pour les codages nôayant pas recours aux variables binaires. Pour cela on sélectionne une position i et on récupère les gènes

P_1 (i) et P_2 (i) des deux parents à cette position. À partir de ces gènes, on définit les gènes des enfants à cette position C_1 (i) et C_2 (i). Les relations entre P_1 (i) et P_2 (i) pour calculer C_1 (i) et

C_2 (i) dépendent du type de croisement. Le croisement binaire simulé (Simulated Binary Crossover), basé sur le croisement à un point, définit cette relation de la manière suivante :

$$\begin{cases} C_1(i) = 0.5[(1+\beta)P_1(i) + (1-\beta)P_1(i)] \\ C_2(i) = 0.5[(1-\beta)P_2(i) + (1+\beta)P_2(i)] \end{cases}$$

Où β représente un facteur de dispersion défini à partir d'une variable aléatoire uniformément répartie et d'un paramètre qui caractérise la forme des dispersions des enfants.

2.3.3.3. L'opérateur de Mutation

La Mutation permet de réduire la ressemblance des individus parents à leurs enfants afin de diversifier l'espace de solution et d'éviter les optimums locaux. Elle consiste à définir une valeur de remplacement de certains gènes des individus à muter. Cette valeur est définie soit aléatoirement soit par un calcul uniforme.

La mutation ne doit cependant pas être appliquée souvent, car elle peut orienter vers une recherche aléatoire.

2.3.3.4. L'élitisme

À chaque itération, des individus de la population peuvent être remplacés par de nouveaux individus. Lors de ce remplacement, il y a de grandes chances que les meilleurs individus soient perdus. Pour éviter cela, l'élitisme permet à chaque itération de copier un ou plusieurs individus dans la nouvelle population avant de faire la reproduction.

2.3.4. Gestion des solutions dans les Algorithmes Génétiques multi-objectifs

Dans les problèmes multi-critères, les objectifs sont en général contradictoires. Les solutions peuvent optimiser certains objectifs sans pour autant être bonnes sur d'autres objectifs. Selon les situations, il est possible de faire des compromis et de choisir des solutions avantageuses sur certains objectifs même si elles ne sont pas bonnes sur d'autres. Il est alors important de pouvoir fournir, comme solutions des problèmes multi-objectifs, un ensemble de choix possibles.

Les algorithmes génétiques sont très adaptés pour régler les problèmes multi-objectifs. Ils sont basés sur une approche évolutionnaire manipulant une population de solutions et faisant des explorations sur différentes régions de cette population. Il existe deux approches pour gérer les problèmes multi-objectifs : les approches Pareto et les approches non Pareto.

2.3.4.1. Les approches non Pareto

Schaffer (Schaffer, 1985) a défini le premier algorithme génétique multi-objectif : VEGA (Vector Evaluated Genetic Algorithm). Il utilise un processus de recherche qui traite séparément les objectifs. Cette approche se fait en deux étapes. Dans la première, les objectifs sont traités séparément. Si la taille de la population est de *n* et quôon a *k* objectifs, une sélection de *n/k* individus est effectuée pour chaque objectif. On obtiendra alors K sous-populations chacune contenant *n/k* meilleurs individus pour chaque objectif. Ces sous-populations sont regroupées en une nouvelle population sur laquelle vont se faire les opérations de croisement et de mutation. Les approches non Pareto sont faciles à implémenter, mais donnent parfois des solutions extrêmes qui ne permettent pas dôavoir de bon compromis (Konak, et al., 2006).

2.3.4.2. Les approches Pareto

Lôapproche Pareto consiste à regrouper les solutions en un ensemble de solutions non-dominées qui constituera le front de Pareto.

Définition 2.9 : Soit un problème multi-objectif avec {f_1, f_2,é , f_k} k objectifs à minimiser. Une solution **x** *domine* une autre solution **y** si et seulement si :

$$\begin{cases} \forall i \in [1..k], fi(x) \leq fi(\\ \exists j \in [1..k] tel\ que\ fj(x) < \end{cases}$$

La figure 2.19 présente des exemples de dominance et de non-dominance pour un problème à deux fonctions (f1, f2) conformément à la définition 2.6.

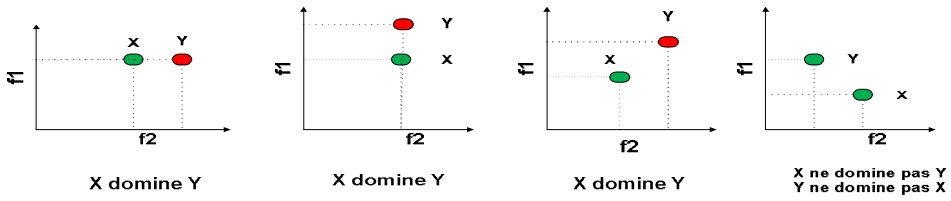


Figure 2. 19 Notion de dominance

Le front de Pareto optimal (premier front) est lôensemble de solutions non dominées. Lôobjectif de lôoptimisation multi-objective est alors de déterminer un front de Pareto optimal pour un problème multi-objectif. Un exemple de front de Pareto pour un problème à deux objectifs est présenté dans la figure 2.20.

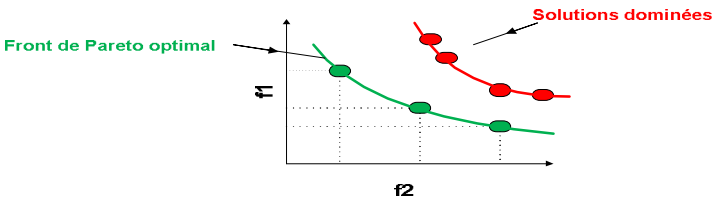


Figure 2. 20 Front de Pareto

Il existe plusieurs algorithmes basés sur l'approche Pareto. L'un des algorithmes les plus adaptés pour les problèmes multi-objectifs est le NSGA-II (Fast Non-dominated Sorting Genetic Algorithm).

Les auteurs de (Srivinas, et al., 1994) ont d'abord défini le NSGA (Non-dominated Sorting Genetic Algorithm). Il utilise la notion de non-dominance et définit plusieurs fronts dans lesquels les solutions sont classées selon leur performance. Les solutions du premier front sont meilleures que celles du front 2 qui sont quant à elles meilleures à celles du front 3, etc. Les solutions qui sont dans un même front *i* ne se dominent pas entre elles, mais sont dominées par celles du front *i+1*. Pour diversifier la population, le NSGA utilise une fonction de partage de la fonction objective qui augmente la chance des solutions se trouvant aux fronts les plus hauts (les plus mauvaises solutions) d'être sélectionnées. Cette fonction consiste à réduire artificiellement les fonctions objectives de ces solutions tout en leur associant des coefficients de pénalités (Konak, et al., 2006). Cette méthode a été beaucoup utilisée et plusieurs inconvénients en sont ressortis (Deb, et al., 2002) :

- La fonction de tri pour classer les solutions en fronts est très couteuse en temps d'exécution.
- L'algorithme n'est pas élitiste et il arrive de perdre de bonnes solutions.
- Le NSGA utilise une fonction de partage qui doit être bien paramétrée par l'utilisateur.

Pour résoudre tous ces inconvénients, (Deb, et al., 2002) a défini le NSGA-II. Ils ont proposé un algorithme de tri de la population en plusieurs fronts qui a une complexité de $O(KN^2)$ au lieu de $O(KN^3)$ pour celle du NSGA (K étant le nombre d'objectifs et N la taille de la population).

Deb et al (Deb, et al., 2002) ont remplacé la fonction de partage nécessitant des paramétrages externes à une fonction de remplacement basée sur une notion de distance de

remplacement (crowding distance) qui guide vers une sélection assurant une diversité. Pour cela, on associe à chaque solution i deux caractéristiques :

- i_{rank} qui représente le rang de non-dominance de la solution conformément à son front. Si la solution est au front 1 son i_{rank} est de 0.
- $I_{distance}$ qui représente la taille du plus grand « cuboïde » contenant i sans contenir les autres solutions de la population. Pour déterminer lâdistance, on calcule la distance moyenne sur chaque objectif entre la solution i et les solutions les plus proches de part chaque objectif. La figure 2.21 illustre le calcul de la $i_{distance}$ pour un problème à 2 objectifs.

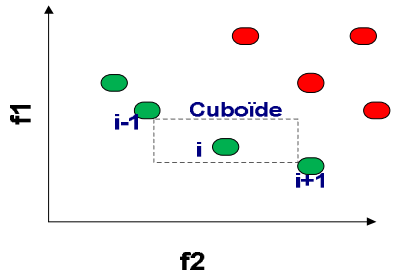


Figure 2. 21 Calcul de $i_{distance}$ (Deb, et al., 2002)

Un opérateur de comparaison $Ó_n$ a été défini pour guider le processus de sélection. Pour deux solutions i, j :

$$i \, Ó_n j \text{ si } (i_{rank} < j_{rank}) \text{ ou } ((i_{rank} = j_{rank}) \text{ et } (i_{distance} > j_{distance}))$$

Lâalgorithme NSGA-II utilise une technique élitiste. La figure 2.22 illustre son fonctionnement. À chaque itération t, la population P_t est reproduite pour créer un ensemble de nouvelles solutions Q_t. P_t et Q_t sont alors regroupées en un nouvel ensemble R_t de taille 2N ($R_t = P_t \cup Q_t$). Un tri est alors effectué pour générer la nouvelle population P_{t+1}. Les solutions sont ajoutées dans P_{t+1} par front jusqu'à ce que P_{t+1} ait atteint la taille N. toutes les autres solutions de R_t qui nâont pas pu être ajoutées dans P_{t+1} seront éliminées et P_{t+1} sera utilisée comme population pour la prochaine itération t+1.

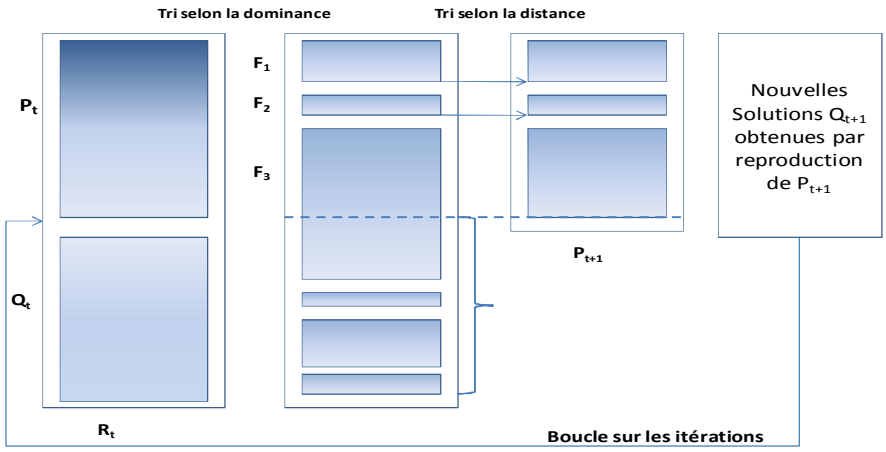


Figure 2. 22 Gestion des solutions par NSGA-II

2.4 Résumé

Ce chapitre a présenté trois concepts différents. Le premier est le concept d'architectures nano-composantes définies par (O'Connor, et al., 2008) sur lesquelles nos travaux sont basés. Ils ont exploité une caractéristique d'ambivalence des dispositifs de la nanotechnologie pour définir une famille de cellules dynamiquement reconfigurables. Ces cellules ont ouvert la voie vers un nouveau concept d'architecture à granularité ultrafine. Une structure matricielle a été définie pour les architectures nano-composantes DG_CNTFET.

Le problème de placement étudié dans ce mémoire a été formulé en utilisant les graphes. À cet effet, quelques concepts de théorie de graphe ont été étudiés. Il s'agit : (i) de la notion de graphe orienté acyclique ; (ii) la notion de parcours dans les graphes à savoir le parcours en profondeur et en largeur et (iii) la notion de cheminement notamment les algorithmes de plus court chemin et de K-plus courts chemins.

Le dernier concept étudié est celui des algorithmes génétiques, leur fonctionnement, la formulation mathématique qui leur est associée, les opérateurs de sélection, de croisement et de mutation qui les caractérisent et enfin les différentes approches pour la gestion de solutions.

L'utilisation de tous ces concepts sur le problème de placement est présentée dans le chapitre suivant.

CHAPITRE 3. MODÈLE DE PLACEMENT PROPOSÉ

Le modèle de placement proposé dans ce mémoire permet de placer une application logique sur les architectures nano-composantes tout en essayant d'optimiser leurs performances. Ce modèle prend en compte les caractéristiques des architectures nano-composantes qui ont été présentées dans le chapitre 2 :

- les cellules logiques sont dynamiquement reconfigurables ce qui permet de pouvoir faire des exécutions en pipeline puisqu'à chaque cycle, une cellule peut être reconfigurée pour traiter une des 14 opérations logiques qu'elle supporte;
- les architectures ont une granularité très fine, car les cellules prennent deux entrées codées sur 1bit et donnent une sortie, pouvant être dupliquée, codée 1 bit. Ceci contraint à avoir des applications dont les opérations logiques opèrent sur les mêmes types de données, et
- les architectures ont une structure hiérarchique à deux niveaux : les cellules sont organisées en matrices et les matrices en réseau de matrices. Ce qui permet de faire des exécutions parallèles sur plusieurs matrices.

Lors du placement, quatre métriques doivent être optimisées :

- Le nombre de cellules (matrices) inactives lors de l'exécution de l'application
- Le temps d'exécution de l'application
- Le coût de chargement des cellules logiques c.-à-d., leur configuration par les entrées de contrôles à chaque cycle.
- Le coût de communication entre les matrices

Ce chapitre est organisé en six sections. La première section présente la formulation du problème de placement. La seconde donne une vue générale du modèle et des deux parties qui le composent. La méthode de placement au sein d'une matrice est présentée dans la troisième section. La méthode de partitionnement est décrite dans la quatrième section, la cinquième section est consacrée à la partie de placement global en utilisant les algorithmes génétiques. Un résumé conclut le chapitre.

3.1. La formulation du placement

Cette section consiste à modéliser le problème de placement de manière claire afin de pouvoir aisément manipuler les différents concepts qui sôy rapportent. Nous avons choisi la modélisation sous forme de graphe DAG (Direct Acyclique Graph) pour représenter les concepts.

3.1.1. Lôarchitecture physique

Lôarchitecture nano-composante peut être considérée comme un réseau de matrices. Ce réseau est représenté comme un ensemble $\mathbf{C_M} = \{m_1, m_2, \text{é } m_k\}$ de matrices homogènes. Il se caractérise alors par le nombre de matrices, leurs dispositions au sein du réseau ainsi que leur taille. Chaque matrice m est représentée par un graphe acyclique orienté (DAG) $G_m = \{V_m, E_m\}$ où V_m représente lôensemble des cellules de la matrice m et E_m les connexions entre les cellules qui se caractérise par la topologie spécifiée (Omega-modifiée, Banyané). Les connexions entre les matrices nôont pas été modélisées puisque les matrices communiquent entre elles avec des connexions dynamiques sans quôil y ait restriction de connectivités entre elles. Lôinformation importante pour la communication entre les matrices est leurs dispositions au sein du réseau. La modélisation de lôarchitecture nano-composante est illustrée par la figure 3.1.

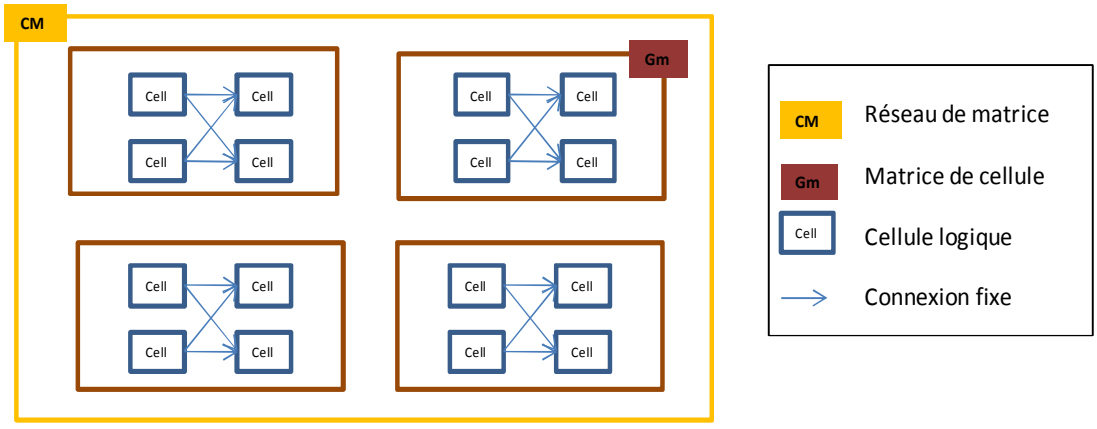


Figure 3. 1 Formulation de lôarchitecture physique

3.1.2. Lôapplication logique

Lôapplication logique consiste en un réseau dôopérations booléennes prenant en entrées des données binaires codées sur 1 bit. Les opérations peuvent être binaires (OR, AND, NANDé), unaire ($\overline{A}$, Bé) ou sans entrées (0,1).

La notation DAG est utilisée pour représenter le flot de données de lôapplication $G_f = (V_f, E_f)$. V_f est lôensemble des opérations logiques et E_f, les dépendances de données entre les opérations.

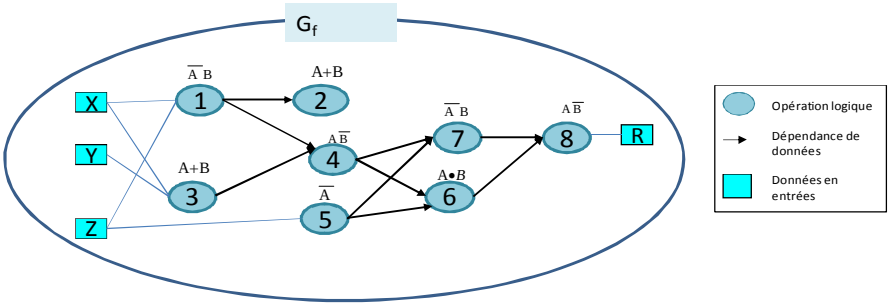


Figure 3. 2 Formulation de l'application

Définition 3.1 : *La profondeur dôune application* est la longueur du plus long chemin dans son graphe DAG. Dans lôexemple de la figure 3.3.a, la profondeur de lôapplication est de 4. Elle est donnée par le chemin 3-4-7-8.

Définition 3.2 : Le *niveau* dôune opération se calcule de manière récursive. Les opérations qui nôont pas de prédécesseurs ont le niveau zéro. Le *niveau* dôune opération dans le graphe est le plus grand niveau de ses prédécesseurs plus 1. Dans la figure 3.3.b les opérations 1 et 3 sont de niveau 0, les opérations 2, 4 et 5 sont de niveau1, etc.

Comme expliqué dans le chapitre 2, les cellules sont organisées en couche dans les matrices, lôidentification du niveau dôune opération logique, permettra de savoir dans quelle couche de la matrice elle peut être placée (un niveau dans la sous-fonction correspond à une couche dans la matrice).

Définition 3.3 : La *largeur* dôune application est le plus grand nombre dôopérations ayant le même niveau. Dans la figure 3.3.b, il yôa 2 opérations de niveau 0, 3 opérations de niveau 1, 2 opérations de niveau et 1 opération de niveau 3. La largeur de lôapplication est donc 3. Elle est donnée par 2-4-5.

La décomposition dûne application G_f fournit un graphe de dépendance $G_d = (V_d, E_d)$ où V_d représente lôensemble des sous-graphes obtenu après la décomposition de lôapplication $V_d = \{G_{f0}, G_{f1}, \text{é} , G_{fi}\}$ et E_d représente les dépendances entre ces sous-graphes.

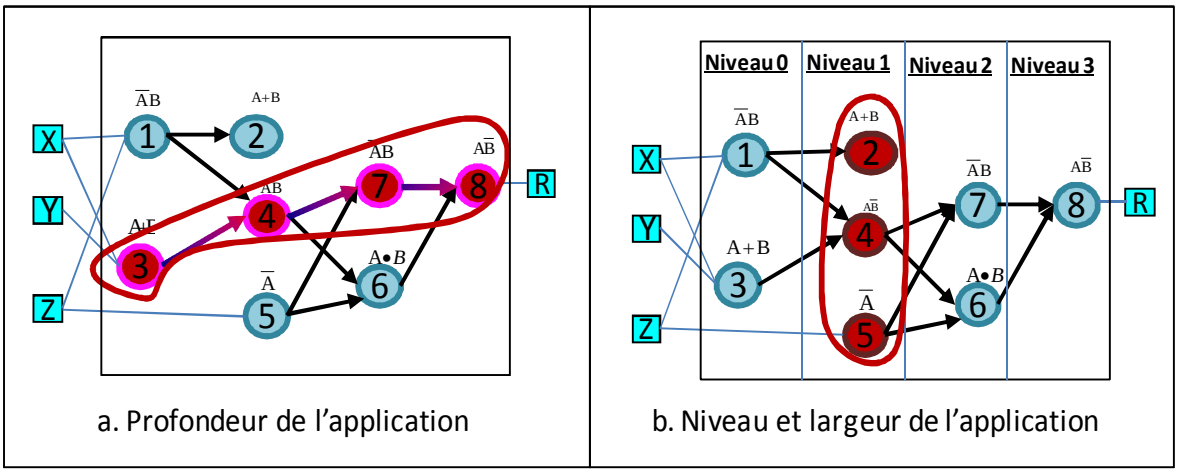


Figure 3. 3 Caractéristiques de l'application

3.1.3. Objectif de placement

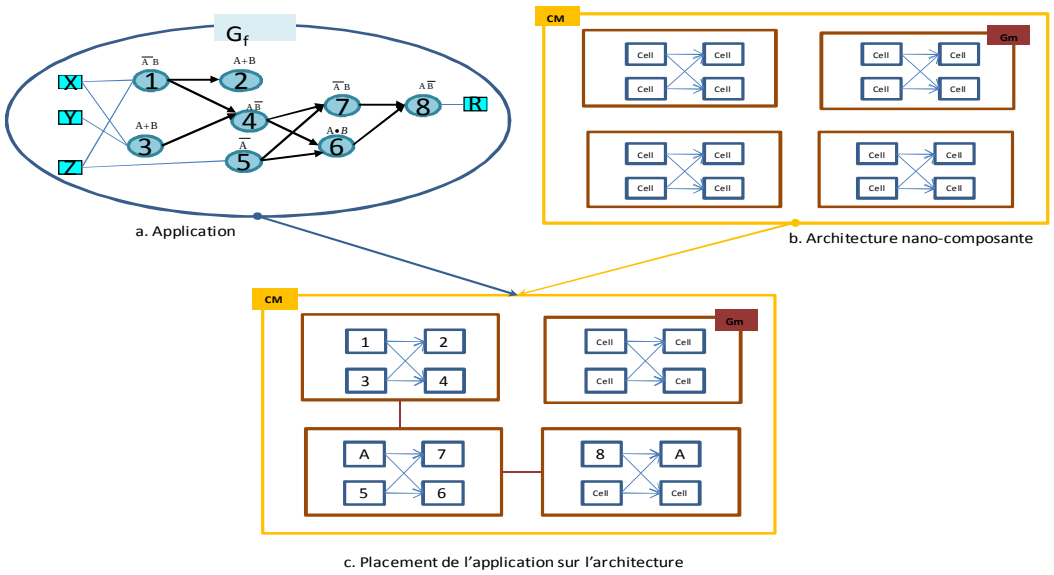


Figure 3. 4 Objectif de placement

Lôobjectif du placement est de trouver un modèle de placement qui permet de placer une application, sous forme de flot de données, $G_f = (V_f, E_f)$ sur un réseau de matrices $Cm = \{m_1, m_2, é\ m_k\}$ en optimisant lôexploitation des ressources de lôarchitecture.

La figure 3.4 illustre lôobjectif quôon souhaite atteindre : à partir dôune application représentée sous forme de graphe et une architecture sous forme dôensemble de matrices, on va essayer dôassocier à chaque opération logique de lôapplication une cellule logique dans une des matrices de lôarchitecture et déterminer les connexions entre les matrices.

Par la suite du chapitre toute notation G_{fi} référera à un graphe dôapplication ou un sous-graphe et toute notation G_{mi} à une matrice. Le graphe dôapplication fera référence à lôapplication globale et le sous-graphe à une sous-fonction qui est une partie de lôapplication.

3.2. Description du modèle de placement

Les applications à placer sont en générales très complexes et ne peuvent pas être placées sur une seule matrice. Les interconnexions intra-matrices étant fixes, il faudrait sôassurer que chaque opération logique placée sur une cellule est connectée à ses successeurs et à ses prédécesseurs. Cette situation rend le partitionnement de lôapplication encore plus spécifique puisque chaque sous-graphe obtenu après le partitionnement, doit pouvoir être placé sur une matrice. Deux types de placement sont alors effectués :

- Lôassignation matricielle en tenant compte des interconnexions fixes et des nombres de cellules dans une matrice.
- Le placement global en définissant lôordre dôexécution de chaque sous-graphe et la matrice sur laquelle il va sôexécuter.

Le Modèle de placement est présenté à la figue 3.5. Il est réalisé en deux parties :

1) **Partitionnement de lôapplication et assignation matricielle :** Dans cette partie, le modèle prend en entrée le graphe de lôapplication G_f, les informations sur la taille de la matrice (on parle dôarchitectures homogènes). Le graphe est partitionné en un ensemble de sous-graphes qui vont être placés sur les matrices.

Deux méthodes basées sur les concepts de théorie de graphe sont alors utilisées à ce niveau :

- La méthode de partitionnement qui permet de diviser une application en sous fonctions
- La méthode d'assignation matricielle qui permet de placer une sous-fonction sur une matrice

2) **Placement global :** Une fois que le graphe de l'application est partitionné et qu'on s'est assuré que chaque sous-graphe est « plaçable » sur une matrice, on passe au placement global. Ayant un ensemble de *m* sous-graphes et un ensemble de *n* matrices, l'objectif sera de trouver le meilleur ordre d'exécution, en tenant compte des ressources disponibles. Dans cette seconde partie, on utilise une méthode de placement global basée sur les algorithmes génétiques.

La sortie du modèle de placement est le code de reconfiguration des matrices pour l'exécution des sous-graphes ainsi que leurs ordres d'exécution sur les matrices.

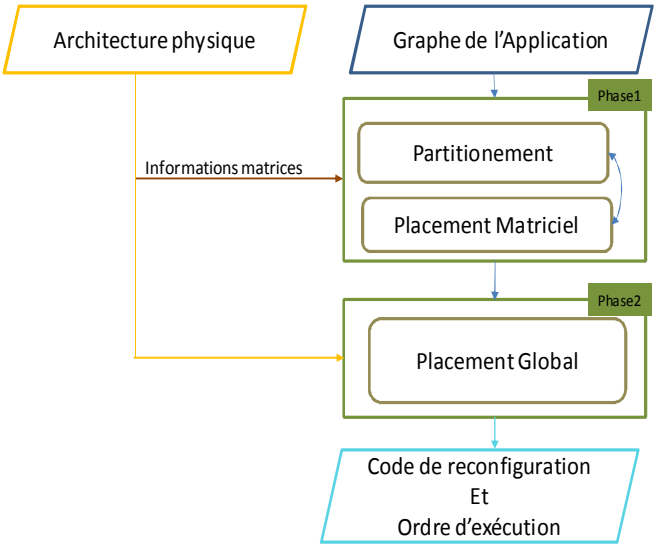


Figure 3. 5 Modèle de placement proposé

3.3. La méthode d'assignation matricielle

Cette section explique le fonctionnement de l'assignation matricielle entre une sous-fonction de l'application et une matrice de l'architecture.

Définition 3.4 : Une sous-fonction est *compatible* avec une matrice si les connexions entre les opérations logiques de la sous-fonction peuvent être retrouvées dans la topologie de la matrice.

Définition 3.5 : Un *chemin libre* dans une matrice est un chemin qui traverse des cellules libres c.-à-d. des cellules sur lesquelles aucune opération logique n'a été placée.

3.3.1. Objectif de l'assignation matricielle

L'assignation matricielle se fait entre une sous-fonction résultant du partitionnement et une matrice de l'architecture. L'objectif de l'assignation matricielle est d'associer à chaque opération logique, de la sous-fonction, une cellule logique nano-composante, dans la matrice, de manière à assurer le routage de l'information selon la topologie d'interconnexion. Pour ce faire, il est important de tenir compte des restrictions des connexions et de l'organisation des cellules par couche au sein des matrices.

Deux conditions doivent être vérifiées pour qu'une fonction soit placée sur une matrice :

i) Pour toute matrice de taille *Largeur* × *Profondeur*, la largeur de la sous-fonction (cf. *définition 3.3*) ne doit pas dépasser *Largeur* et la profondeur (cf. *définition 3.1*) de la sous-fonction ne doit pas dépasser *Profondeur*.

ii) Pour toute matrice donnée, sa topologie doit être compatible (cf. *définition 3.4*) avec la structure de la fonction.

Lorsque ces deux conditions ne sont pas vérifiées, le placement peut échouer. Une métrique de la méthode d'assignation matricielle est *le taux de couverture*. C'est le pourcentage de sous-fonctions qui ont été placées avec succès sur des matrices. La méthode d'assignation matricielle doit optimiser cette métrique en s'assurant que toutes les éventualités ont été étudiées avant de décréter qu'il y a eu *échec de placement* c.-à-d. une fonction donnée n'a pas pu être placée sur une matrice.

3.3.2. Fonctionnement de l'assignation matricielle

Le fonctionnement de la méthode de placement est illustré par la figure 3.6. Il prend en entrées le graphe de la sous-fonction et le graphe de la matrice physique sur laquelle va se faire le placement. Le graphe est optimisé, s'il y a lieu, pour éviter une redondance de données lors du placement. Le graphe est ensuite traversé dans un parcours en profondeur modifié pour lister les arêtes du graphe. Cette liste est utilisée itérativement pour placer les sommets de chaque arête sur la matrice physique.

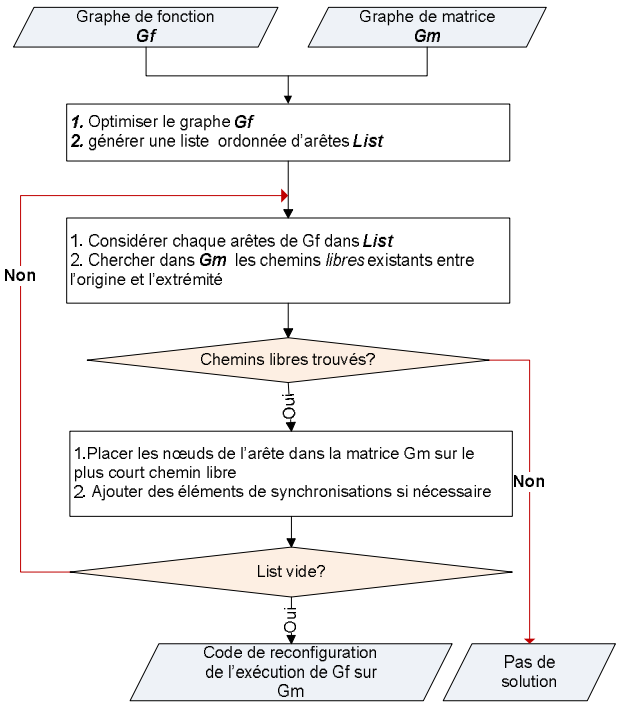


Figure 3. 6 Fonctionnement de lôassignation matricielle

Définition 3.6 : soit un arc (x_1, x_2) reliant deux nî uds x_1 de niveau n_1 et x_2 de niveau n_2 ($n_1<n_2$). Lorsque $n_2-n_1>1$, pour placer ces nî uds sur une matrice, il faudra passer par une ou plusieurs couches intermédiaires puisque x_1 sera placé sur la couche n_1 et x_2 sur la couche n_2. On appelle *éléments de synchronisations*, les suiveurs qui seront rajoutés pour faire passer la donnée. La figure 3.7 présente un graphe et lôajout dôéléments de synchronisations dans ce graphe. Lôarc (5,4) entre les opérations 5 de niveau 0 et 4 de niveau 3 traverse deux niveaux. Pour faire passer le résultat de 5 vers lôopération 4, il va falloir passer par les niveaux 1 et 2 dôòù lôajout des opérations **6** et **7** comme éléments de synchronisations.

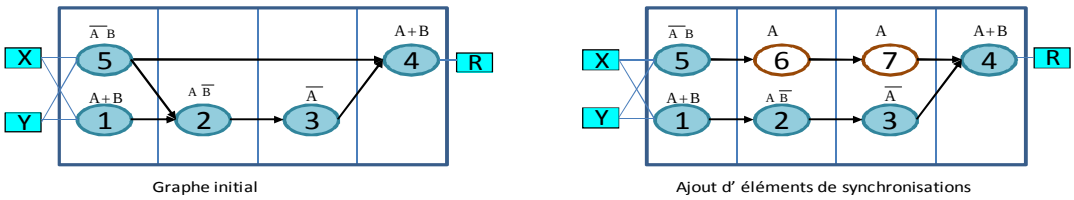


Figure 3. 7 Ajout d'éléments de synchronisations

Ces éléments de synchronisations vont être rajoutés si nécessaire lors du placement. Cela va être plus simple puisque leur emplacement sera déjà déterminé par le chemin libre trouvé dans

le graphe de la matrice. Cependant, il existe deux cas particuliers dans lesquels lâajout des éléments de synchronisations lors du placement sera redondant. Une étape dâoptimisation consistera à éviter cela.

3.3.2.1. Génération de liste ordonnée dâarcs

Le placement des opérations de la fonction logique sur la matrice physique se fait selon un ordre défini. Lâordre proposé est un parcours en profondeur modifié afin de permettre un raffinement du placement. La modification vient du fait que pour une opération ayant deux entrées, lâalgorithme doit sâassurer que les deux entrées peuvent être acheminées vers lâemplacement de lâopération sinon lâemplacement de lâopération doit être changé. La figure 3.8 présente un exemple de liste ordonnée. Lâordre des opérations est défini en parcours en profondeur et en parcours en profondeur modifié. La différence entre les 2 types de parcours se trouve au niveau des arcs (2,4) et (1,2). Il sâagit en général dâune cellule logique à deux prédécesseurs et deux successeurs. Lorsque lâopération 4 est placée sur une cellule, on doit sâassurer que lâautre prédécesseur de la cellule exécute lâopération 2.

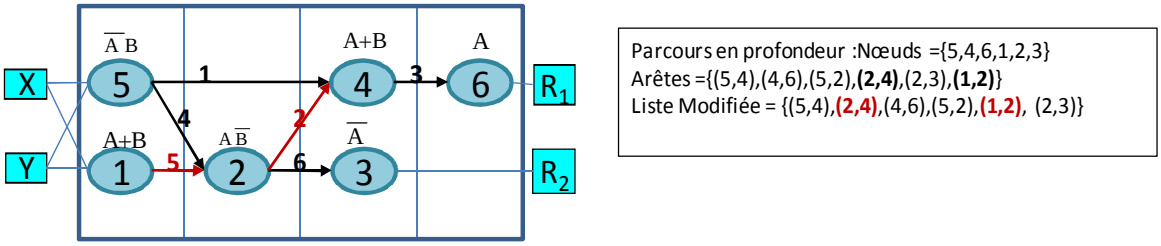


Figure 3. 8 Liste ordonnée

3.3.2.2. Recherche de chemins libres et placement

Le graphe de matrice physique est un graphe pondéré par la taille des communications entre les nî uds. Pour deux nî uds n_1 et n_2 placés respectivement à la position *i* au niveau *e* et à la position *j* au niveau *e+1*. Le poids de lârc qui les relie $W_{n1, n2}$ est *j-i+1*.

Pour chaque arc du graphe de fonction, lâalgorithme cherche les chemins possibles (et par ordre de plus courts chemins *K-plus courts chemins, cf. chapitre 2*). Le premier chemin libre est retenu et le placement des nî uds se fait selon ce chemin. Si les nî uds n_1 et n_2 sont à des étages respectifs i, j de la matrice physique (i<j), le chemin trouvé permet au besoin de rajouter (j-i-1) éléments de synchronisations (suiveur).

La figure 3.9 présente une fonction logique et son placement sur une matrice de topologie Oméga-modifiée. Cette figure met en évidence la correspondance entre le niveau des opérations logiques et les couches dans les matrices : les opérations de niveau i sont placées dans la couche i de la matrice. Dans la figure 3.9 les opérations **5** et **1** ont le niveau 0 et sont placées dans la couche 0, lôopération **2** a le niveau 1 et est placée dans la couche 1é .

Lôajout des éléments de synchronisations entre deux opérations se fait dynamiquement lors du placement. Dans la figure 3.9, un élément de synchronisation a été ajouté pour faire passer le résultat de lôopération **5** vers lôopération **4**.

Après avoir placé tous les arcs, la prochaine étape consiste à rajouter, si nécessaire, des éléments de synchronisations pour rendre les entrées et résultats accessibles. Dans lôexemple de la figure 3.9, ce type de synchronisation a été rajouté pour faire passer le résultat de lôopération **4**.

Lorsquôil reste des cellules inactives dont le chemin permet un routage cohérent de données, dôautres applications indépendantes peuvent être rajoutées jusqu'à saturation de la matrice.

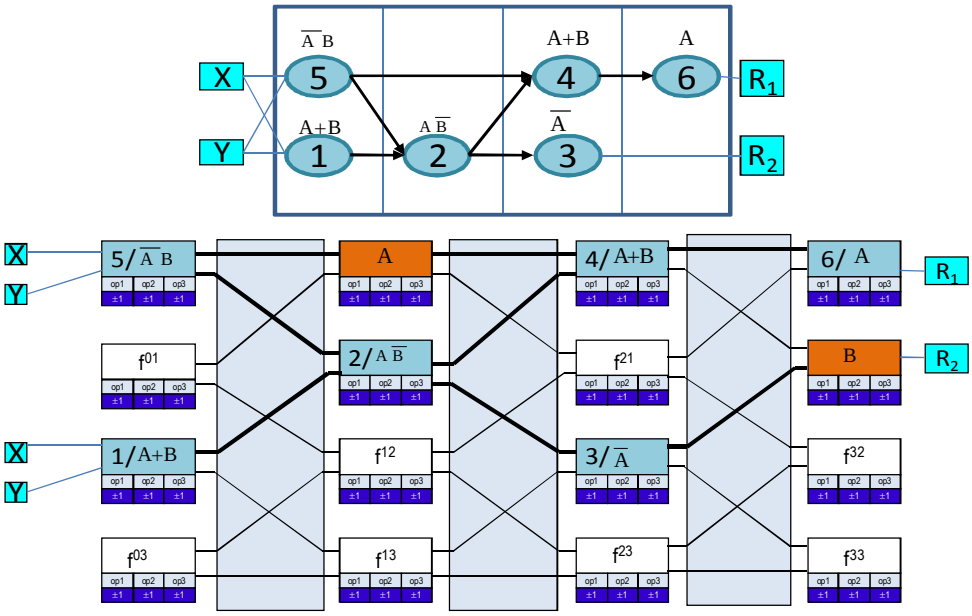


Figure 3. 9 Assignation sur une matrice de topologie Oméga-modifiée

La méthode dôassignation matricielle permet dôassurer un routage cohérent des données à travers une matrice. Dans le modèle proposé, elle est utilisée pour placer les sous-graphes découlant de la méthode de partitionnement sur les matrices de lôarchitecture.

3.4. La méthode de partitionnement

3.4.1. Objectif du partitionnement

Le partitionnement consiste à diviser une fonction complexe, quôon ne peut pas placer sur une matrice, en un ensemble minimum de sous-graphes « plaçable » sur une matrice. Il doit permettre de :

- **Minimiser la communication entre les matrices** Plus il yô aura de dépendances de données entre les sous-graphes, plus il yô aura de communication entre les matrices. Minimiser la communication entre matrices, à ce niveau, revient alors à minimiser le nombre de dépendances entre sous-graphes;
- **Minimiser le nombre de matrices utilisées** Il sôagit de minimiser le nombre les sous-graphes, car moins on en a, plus on réduit les dépendances;
- **Éviter les communications cycliques** les cycles entre les graphes vont causer des retours entre les matrices. Ce qui va engendrer de grands coûts de communication inutile.

La méthode de partitionnement donne en sortie un ensemble S = {G_{f0}, G_{f1},é , G_{fk}} de sous-graphes qui doit satisfaire les conditions suivantes :

i) Les sous-graphes sont disjoints et lôensemble des sous-graphes donne lôapplication G_f :

$$G_{fi} \check{\mathbf{z}} G_{fj} = \qquad \forall\, i,j \in [1,2\ldots k] \;\; et \;\; _{fi} = G_f$$

ii) Il nôy pas de dépendances cycles entre les sous-graphes : Si G_{fi} dépend de G_{fj}, G_{fj} ne doit pas dépendre de G_{fi} $\qquad , \forall\, i,j \in$

iii) Chaque sous-graphe G_{fi} doit pouvoir être placé sur une matrice.

3.4.2. Fonctionnement du partitionnement

Lôorganigramme de la méthode de partitionnement est illustré par la figure 3.10. Il consiste en un partitionnement itératif du graphe dôapplication en trois étapes :

- le tri des nî uds du graphe de lôapplication;

- la génération itérative des sous-graphes à partir de la liste ordonnée, et
- le placement de ces sous-graphes sur une matrice.

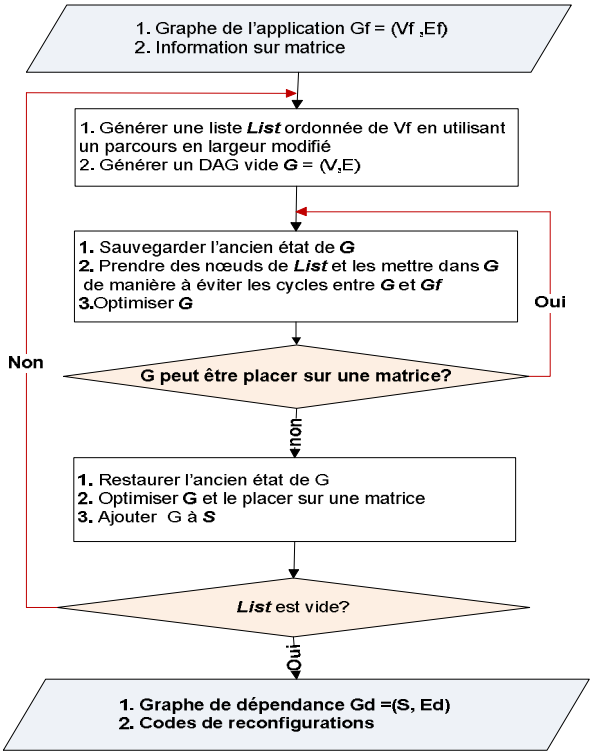
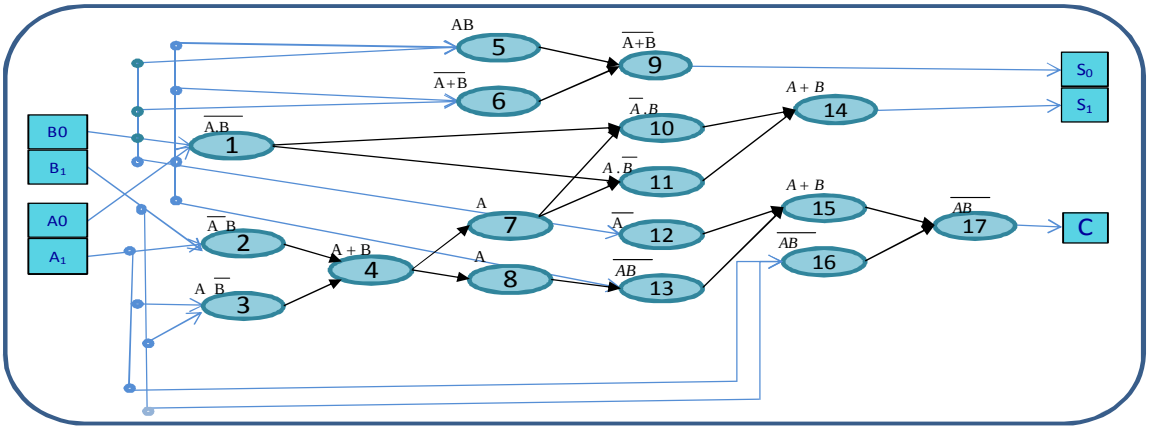


Figure 3. 10 Organigramme de la méthode de partitionnement

3.4.2.1. Le parcours du graphe dôapplication

Le tri des nî uds du graphe de lôapplication G_f= (V_f, E_f) donne en sortie une liste ordonnée nommée *List*. Il est réalisé par un parcours en largeur modifié. Il diffère du parcours en largeur dans la mesure où, au lieu de traverser les successeurs dôun nî ud on traverse tous ses *voisins* (successeurs et prédécesseurs).

Le parcours commence par un nî ud dans V_f (en générale le premier), puis on traverse tous les nî uds voisins de ce dernier. Il se fait de manière récursive jusqu'à ce que tous les nî uds de V_f soient parcourus. La figure 3.11 présente un exemple de liste triée en utilisant ce parcours. Le parcours commence par lôopération ***1***, puis traverse ses voisins directs qui sont ***10*** *et* ***11***. La prochaine itération découvre les voisins de 10 qui sont ***7*** *et* ***14*** qui sont également les voisins de 11. Le même parcours est effectué récursivement jusqu'à ce que tous les nî uds soient traversés. Le résultat complet du tri est présenté dans la liste *List*.



Vf = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}
List= {1,10,11,7,14,4,2,3,8,13,15,12,17,16,5,9,6}

Figure 3. 11 Exemple d'application logique et de liste triée

3.4.2.2. Génération du sous-graphe à partir dôune liste ordonnée

Cette section explique comment se fait la génération des sous-graphes. Au début de chaque génération de sous-graphe, la liste de nî uds de lôapplication est ordonnée en utilisant le parcours présenté dans la section précédente. Une fois que les nî uds sont ordonnés un nouveau graphe DAG G= (V, E) est créé. Ce graphe va servir pour générer un sous-graphe du graphe dôapplication.

Initialement, les ensembles V et E sont vides. On va commencer par définir V puis déduire les arcs de lôensemble E à partir de V. Les sous-graphes générés doivent pouvoir être placés sur une matrice. Pour cela, la méthode de partitionnement prend en entrées les caractéristiques des matrices et le partitionnement se fait en fonction de ces caractéristiques. Pour éviter de faire appel à la fonction de partitionnement, à chaque itération, on examine dans un premier temps si le sous-graphe vérifie la première condition qui stipule que la taille du sous-graphe (largeur, profondeur) ne doit pas dépasser celle de la matrice.

La génération de sous-graphes se fait de manière itérative. Les nî uds de la liste ordonnée *List* sont progressivement ajoutés dans la liste du nouveau sous-graphe V. Trois situations peuvent se produire :

- **Situation1 :** Le nî ud, en tête de la liste *List* quôon souhaite transféré dans G, crée un cycle élémentaire entre le graphe G et le reste du graphe dôapplication G_f. Dans ce cas, tous ses

prédécesseurs doivent être transférés dans G. La validité du transfert est ensuite vérifiée : Si la largeur du graphe est plus petite que celle des matrices physiques, le transfert est accepté ; sinon il est rejeté.

La figure 3.12.a illustre un exemple de détection de cycle ainsi que sa résolution. Lorsque le nœud **10** du graphe G_f est transféré dans G, il crée un cycle puisque ces prédécesseurs **7**, **4**, **2**, **3** sont toujours dans G_f. Pour résoudre ce cycle, tous ces nœuds ont été rajoutés dans G. La figure 3.12.b illustre ce rajout.

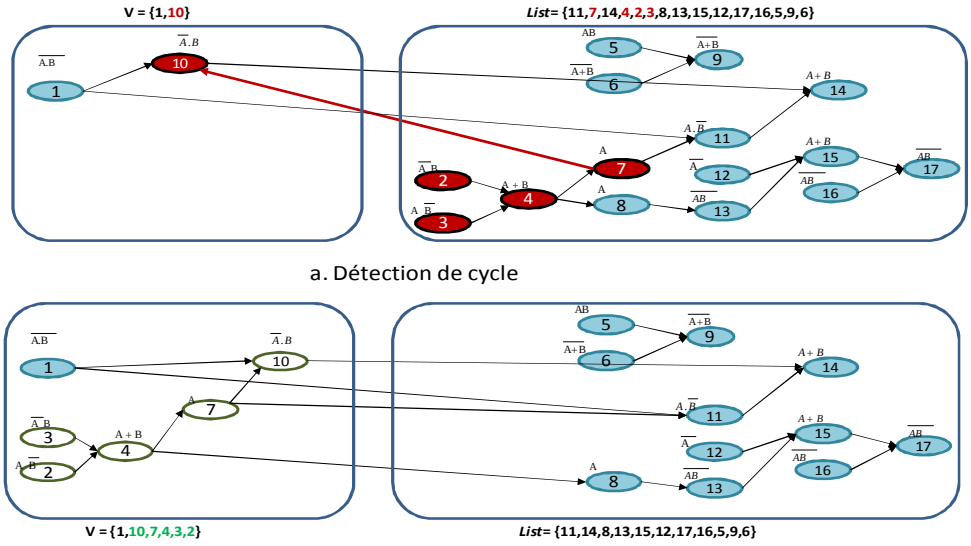


Figure 3. 12 Détection et correction de cycle lors du partitionnement

- **Situation2 :** Le niveau du nœud en tête de la liste *List* est plus grand que la profondeur de la matrice ciblée. Dans ce cas, le transfert n'est pas effectué et on passe au nœud suivant dans *List*.

La figure 3.13 illustre cette situation. Le nœud **14** aura le niveau 4 s'il est rajouté dans G. Si on considère que les matrices de l'architecture sont de taille 4×4 (de largeur 4 et de profondeur 4), le nœud 14 ne pourra pas alors être rajouté dans le graphe G puisqu'il y'aura dépassement de la profondeur de la matrice.

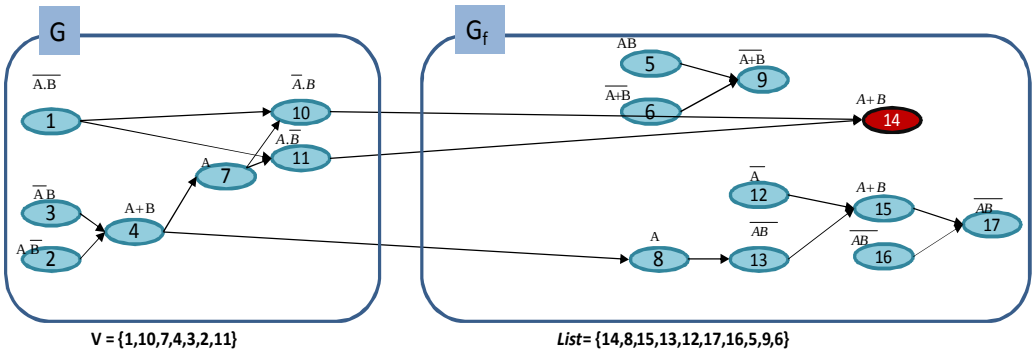


Figure 3. 13 Niveau de nî ud plus grand que la profondeur des matrices

- **Situation3 :** Le nî ud courant rajouté ne crée pas de cycle ni de dépassement de la taille des matrices. Dans ce cas le transfert est validé.

La figure 3.14 présente ce type de situation, le nî ud **8** a été ajouté dans le sous-graphe sans créer un cycle et sans dépasser la largeur limite souhaitée.

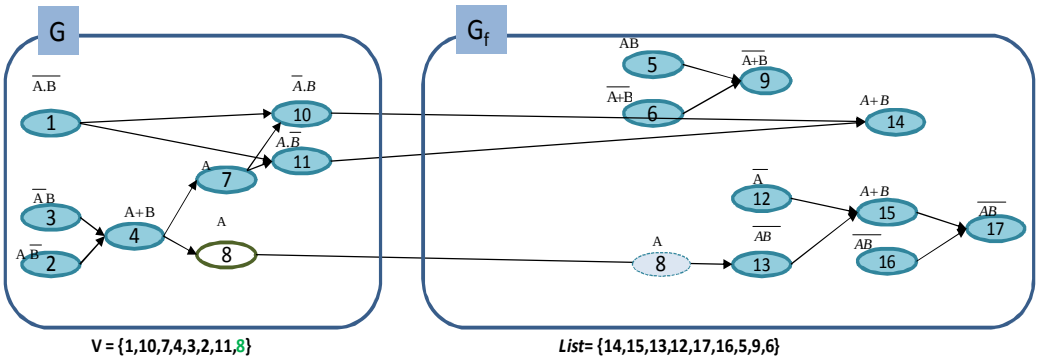


Figure 3. 14 Nî ud sans cycle et sans dépassement de largeur et de profondeur

3.4.2.3. Optimisation de la génération de sous-graphe

Lôoptimisation consiste à vérifier à chaque génération de sous-graphe, sôil yôa des nî uds qui pourront être inter-changés, en fonction des dépendances, pour réduire le nombre de dépendances ou agrandir (rajouter de nî uds) le sous graphe sans rajouter de nouvelles dépendances (ce qui réduira les communications entre les matrices lors du placement global). Dans lôexemple de la figure 3.15, les opérations qui seront concernées pour lôinter-change sont les opérations **13** et **14**. Lôajout de lôopérations **13** dans G est possible puisquôelle agrandit le sous-graphe sans rajouter de dépendances. Lôajout de lôopération **14** aurait réduit les

dépendances ; mais elle nôest pas possible puisquôelle va créer un dépassement de profondeur si la matrice est de taille (4×4).

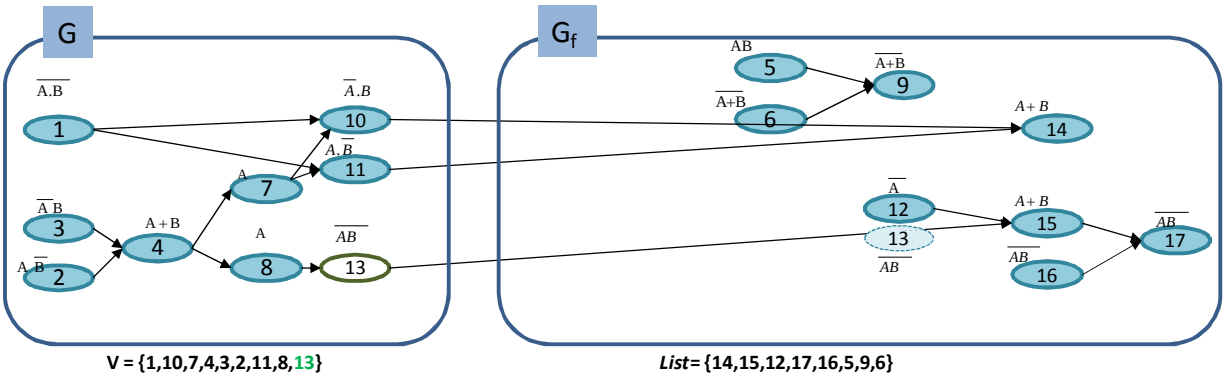


Figure 3. 15 Inter-change de nî uds

Le partitionnement est itératif. Le graphe dôapplication est partitionné tant quôil ne peut pas être placé sur une matrice. À la fin du partitionnement, les sous-graphes sont regroupés en un graphe de dépendances dont les nî uds sont les sous-graphes et les arêtes les dépendances entre les sous-graphes. Les sorties du partitionnement seront alors ce graphe de dépendance ainsi que les informations détaillées de dépendance et de reconfiguration entre les sous-graphes (résultats des assignations matricielles).

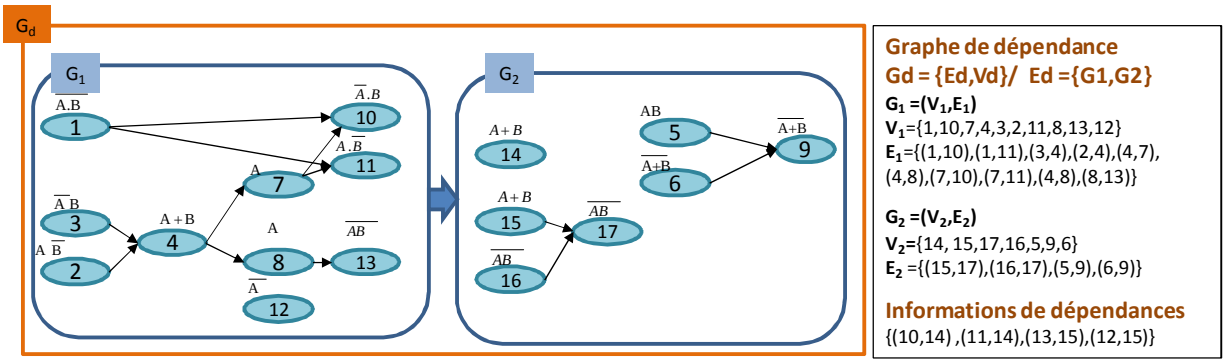


Figure 3. 16 Sorties de la méthode de partitionnement

3.5. La méthode de placement global

3.5.1. Objectif du placement global

Une fois que le graphe d'application a été partitionné en un ensemble de sous-graphes à placer sur des matrices, il faut déterminer sur quelle matrice de l'architecture chaque sous-graphe va être exécuté et à quel moment. L'objectif du placement global est de déterminer ces informations en essayant de minimiser les ressources. Il prend en compte les possibilités d'exécution en pipeline et en parallèle des matrices. Le placement global est dit optimal s'il permet de minimiser :

- **Le nombre de cellules inactives** Il s'agit du nombre de matrices non utilisées lors de l'exécution de l'application. Le mieux, pour optimiser cet objectif, est de faire l'exécution de plusieurs applications en même temps.
- **Le coût de communication entre les matrices** Il s'agit du coût d'envoi de donnée d'une matrice à une autre en fonction des dépendances entre les sous-graphes. Pour évaluer ce coût on supposera que : i) le coût de communication entre deux matrices est proportionnel à la distance entre l'emplacement des matrices dans le réseau de matrice ; ii) il est plus facile de communiquer entre deux matrices distinctes que de reboucler sur une même matrice. Cette hypothèse permet d'évaluer le coût de communication entre les matrices distinctes d'un réseau de matrices.
- **Le coût de reconfiguration des cellules logiques** Ce coût est donné par le nombre de reconfigurations à effectuer lorsqu'on charge un sous-graphe sur une matrice. Lorsque deux sous-graphes i, j doivent être exécutés successivement sur une même matrice, si une cellule de la matrice a été configurée pour une opération donnée lors de l'exécution du sous-graphe i et que l'exécution du sous-graphe j demande la même configuration pour la même cellule, la cellule va exécuter cette opération sans qu'on ait besoin de refaire la reconfiguration. Ainsi lorsque des sous-graphes ayant des configurations en commun sont exécutés successivement, on peut gagner en coût de reconfiguration. La figure 3.17 illustre deux placements différents avec des codes de configurations en commun. Lorsque les cellules logiques sont inactives d'une exécution à une autre, on considère également qu'il s'agit d'une configuration en commun.

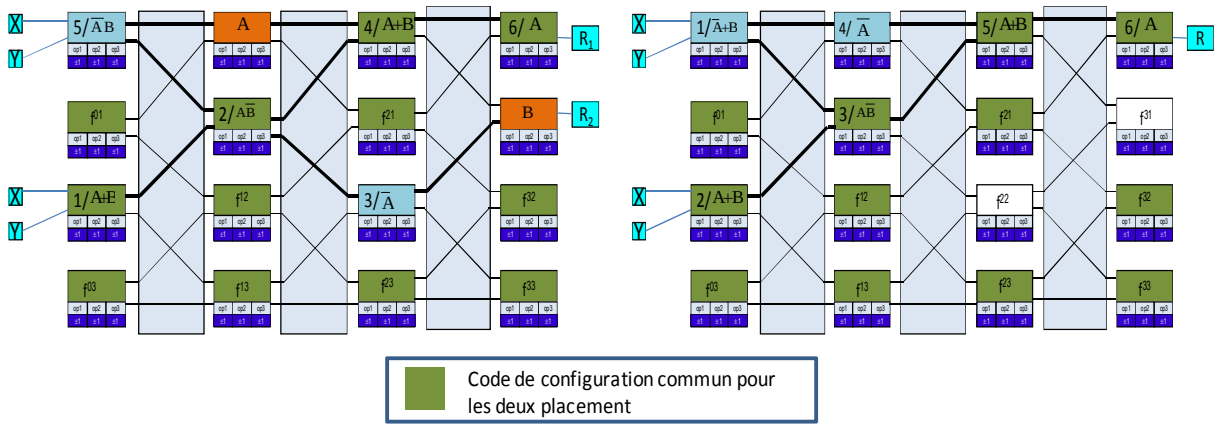


Figure 3. 17 Coût de reconfiguration (chargement des cellules)

- **Le temps d'exécution global de l'application :** Il s'agit d'une estimation du nombre de cycles pendant lesquels l'application va être exécutée. À ce niveau, on considère le temps absolu d'exécution des opérations logiques sur les cellules.

Le placement global a été géré en utilisant les algorithmes génétiques (cf. chapitre 2/section 2.3).

3.5.2. Fonctionnement du placement global

Comme expliqué au chapitre 2, les algorithmes génétiques traitent la formulation d'un problème et non le problème lui-même. Le problème de placement a été dans un premier temps formalisé, puis traité en utilisant les opérateurs génétiques et un algorithme de gestion de solution.

3.5.2.1. Formulation du problème de placement globale

a. Codage des solutions

Les solutions de placement sont codées avec un codage par valeur entière. Chaque solution est composée de l'ordre d'exécution des sous-graphes d'une part et de l'autre du numéro de la matrice sur laquelle chaque sous-graphe va être exécuté. Si l'application a été partitionnée en *n* sous-graphes, la solution de placement est codée sur un tableau de taille $2 \times n$. Les *n* premiers éléments représentent l'ordre d'exécution des sous-graphes (sous-fonctions) et le reste le numéro de la matrice sur laquelle va se faire l'exécution

La figure 3.18 représente un exemple de solution pour un graphe de dépendance donné. Le tableau est divisé en deux parties : la première partie présente lôordre dôexécution des sous-graphes et la seconde partie, les matrices sur lesquelles ils vont être exécutés. Lôordre dôexécution dôun graphe G_i est représenté à la case *i* et la matrice sur laquelle il va être exécuté à la case *i+n* (n étant le nombre de sous-graphes). Par exemple, dans la figure 3.18.b, lôordre du graphe G_0 est à la case 0 et la matrice sur laquelle il va être représenté à la case 7.

Lôordre dôexécution ne représente pas exactement le cycle dôexécution, mais une approximation qui permettra dôestimer le temps dôexécution de lôapplication.

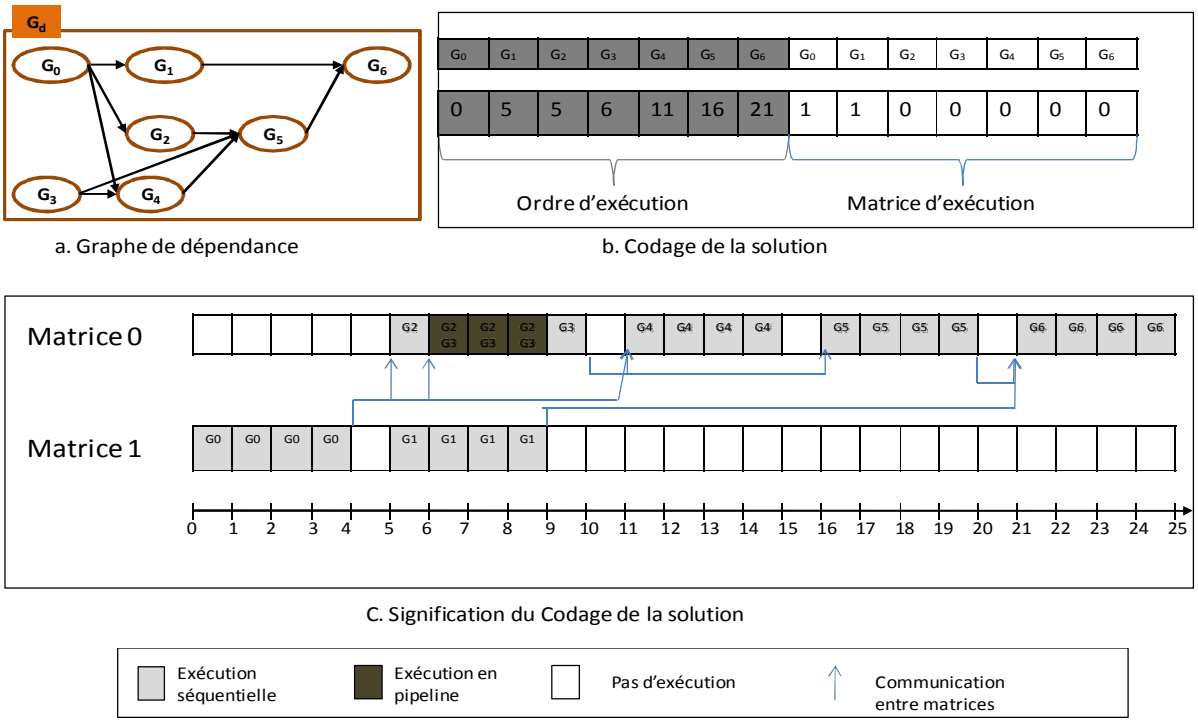


Figure 3. 18 Codage des solutions de placement

Dans la solution représentée à la figure 3.18.c, le graphe G_0 commence lôexécution de lôapplication sur la matrice 1, si on considère une architecture avec une matrice 4×4, les graphes G1 et G2 vont sôexécuter en parallèle sur les matrices 1 et 0. G4 va être chargé en pipeline après G2 sur la matrice 0, G4, G5 et G6 vont être exécutés séquentiellement sur la matrice 0.

b. Évaluation des solutions

La fonction dôévaluation prend en compte les quatre objectifs dôoptimisation du placement cités dans les objectifs du placement global.

- **Temps dôexécution :** Soient *sol* le tableau contenant la solution de placement et *size* la taille du tableau. La première partie du tableau *sol* (de 0 à *size*/2-1) contient lôordre dôexécution (ordonnancement) des sous-graphes et la seconde (de *size*/2 à *size*-1) la matrice dôexécution. Le temps dôexécution de toute lôapplication correspond à la fin de lôexécution du dernier sous-graphe dans lôordonnancement de la solution.

Pour calculer ce temps, on cherche dans la solution, le dernier graphe qui a été ordonnancé c.-à-d. la plus grande valeur *(Max (sol (0..size/2))* dans la partie ordonnancement du tableau *sol*. Le temps dôexécution de lôapplication sera alors la fin de son exécution c.-à-d. le temps où il a commencé à être exécuté plus le temps de son exécution (profondeur de la matrice dôexécution).

$F_{time}(sol) = Max\ (sol\ [0..size/2]) + profondeur\ (sol[indiceMax+size/2])$

Max détermine lôordre dôexécution du dernier sous graphe exécuté et *indiceMax* représente lôindice du sous-graphe auquel lôordre correspond.

Dans lôexemple de la figure 3.19, la taille du tableau est 14. On va chercher le maximum de la première partie du tableau (7 premiers éléments). Ce maximum qui est 21, correspond au début dôexécution du graphe G_6. Le temps dôexécution est alors la fin de lôexécution du graphe G_6. Comme la matrice a une profondeur de 4, il sera égal à 21+4 donc *25*.

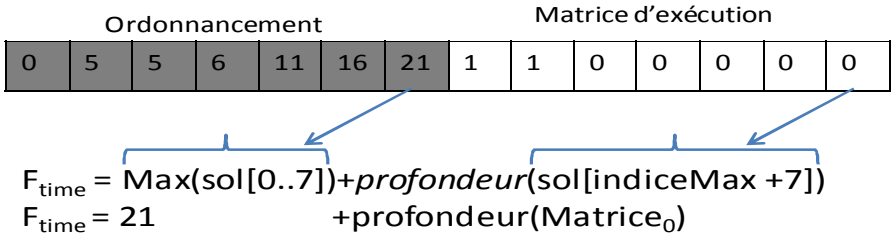


Figure 3. 19 Indentification du dernier sous-graphe exécuté

- **Nombre de cellules inactives :** Lorsque deux sous-graphes i, j doivent être exécutés successivement sur la même matrice au cycle C_i respectivement C_j, le nombre de cellules inactives entre i et j est :

$$nb_{idle}\ (i, j, Matrice) = (C_i\text{-}C_j\text{-}1)*Largeur\ (Matrice)$$

Le nombre de cellules inactives de chaque solution, sur une plateforme de n matrices et chaque matrice ayant Mk graphes à exécuter, est défini par lôéquation :

$$F_{idle} = \sum_{k=0}^{n} \sum_{i=0}^{Mk-1} nb_{idle}(i,i+1,Matrice_k)$$

- **Le Coût de reconfiguration** : Le coût de reconfiguration est calculé de la même manière que la fonction précédente. Lorsque deux graphes i, j doivent être exécutés successivement sur la même matrice, le nombre de reconfigurations effectuées entre i et j est : chargement [i] [j]. Le coût de reconfiguration de chaque solution, sur une plateforme de n matrices et chaque matrice ayant Mk graphes à exécuter, est défini ci-dessous.

$$F_{load} = \sum_{k=0}^{n} \sum_{i=0}^{Mk-1} \text{chargement}[i][i+1]$$

- **Le coût de communication** : Le coût de communication prend en compte la distance de routage de données entre les matrices. Dans le réseau de matrice, chaque matrice m est caractérisée par son emplacement c.-à-d. ses cordonnées m (i, j) dans le réseau. La distance entre deux matrices m_1 (i_1, j_1) et m_2 (i_2, j_2) est estimée comme suit :

$$D(m_1, m_2) = (|i_2-i_1|+|j_2-j_1|)$$

Pour chaque sous-graphe *s* le coût de communication est la somme des distances entre la matrice où il est placé et les matrices où ses voisins sont placés (successeurs et prédécesseurs).

$$Com(s) = \sum_{k=1}^{vois} D(s, v[k])$$ */vois* est le nombre de voisins et *v [k]* le k^{ième} voisin.

Le coût de communication dûne solution est donné par la somme des coûts de communication de chaque sous-graphe.

$$F_{com} = \sum_{s=1}^{n} com(s)$$ où n est le nombre de sous-graphes

c. Évaluation des contraintes des solutions

Cette évaluation consiste à vérifier si une solution respecte les dépendances. Cela signifie quûun sous-graphe dépendant ne doit pas sôexécuter avant les sous-graphes dont il dépend.

G_{dep} = ($NB_{depviolation}$=0), où $NB_{depviolation}$ est le nombre de dépendances violées.

3.5.2.2. Algorithme génétique appliqué au problème de placement

Lôapplication de lôalgorithme génétique au problème de placement est représentée à la figure 3.20. Il prend en entrées le résultat du partitionnement ainsi que lôarchitecture physique c'est-à-dire les matrices et lôinformation sur leur emplacement. La sortie de lôalgorithme génétique est lôinformation globale et détaillée du placement de lôapplication sur lôarchitecture physique notamment le code de lôordre dôexécution des sous graphes sur les matrices, la période

dôexécution et le code de configuration de chaque matrice pour chacune des exécutions pour laquelle elle est utilisée. Lôalgorithme est composé de trois processus :

- La création de la population initiale qui se fait de manière pseudo-aléatoire,
- La reproduction itérative de la population avec les opérateurs génétiques choisis,
- La gestion de la solution avec le NSGA-II.

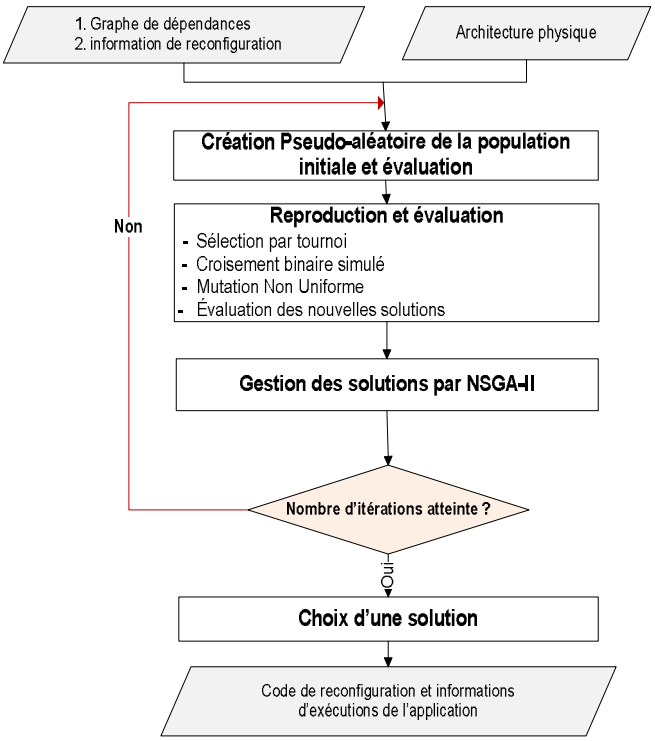
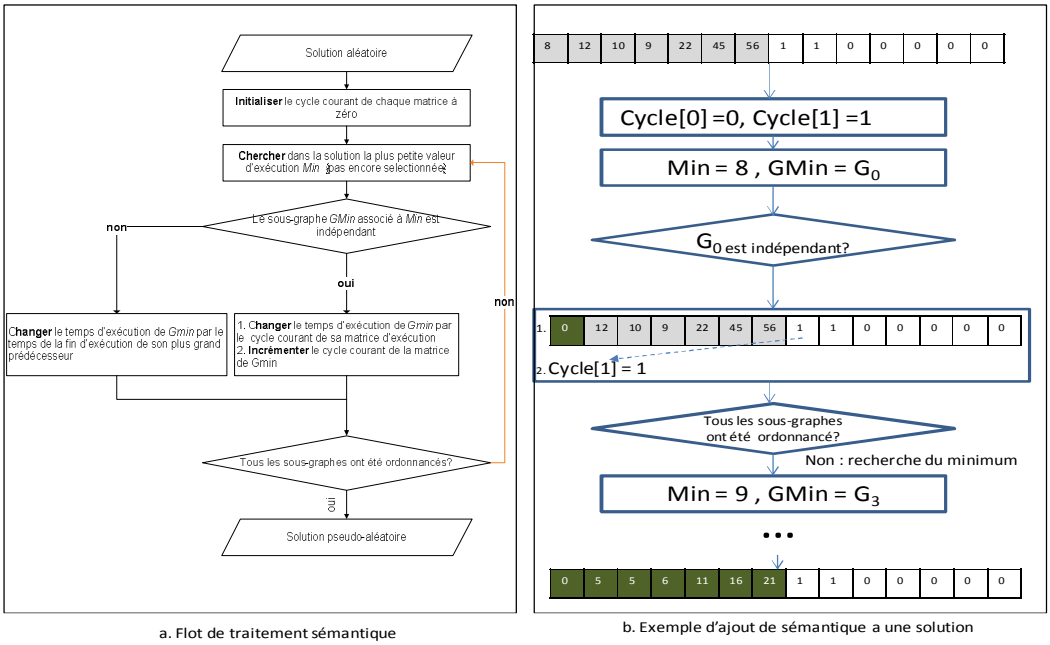


Figure 3. 20 Algorithme génétique pour le placement global

a. Création pseudo-aléatoire de population initiale

Une solution est représentée sous forme de tableau divisé en deux parties. La génération de solutions se fait en deux étapes. La première partie du tableau est une génération aléatoire de lôordre dôexécution des graphes. Ça peut être un entier quelconque entre 0 et un maximum prédéfini. La seconde partie du tableau est aussi générée aléatoirement avec nôimporte quel entier compris entre 0 et le nombre de matrices dans la plateforme.

La génération est dite pseudo-aléatoire car un traitement est fait pour donner à lôordre dôexécution une certaine sémantique. Cela consiste à présenter lôordre aléatoire sous forme dôune estimation de lôordonnancement (cycle où le sous-graphe devra être exécuté). Ceci permet de guider lôespace dôexploration de la solution sans pour autant le mener vers un optimum local. Le traitement sémantique est décrit à la figure 3.21.a. Tout au long du traitement, on garde en mémoire le cycle courant de chaque matrice, initialement le cycle courant de chaque matrice est à 0. Le traitement se fait itérativement jusqu'à ce que tous les nî uds soient ordonnancés. À chaque itération, on cherche dans la solution le temps dôexécution minimum *Min* parmi les sous-graphes qui nôont pas encore été ordonnancés (cela permet de respecter la génération aléatoire). Le graphe ainsi obtenu est ordonnancé selon quôil soit indépendant ou non.



a. Flot de traitement sémantique

b. Exemple d'ajout de sémantique a une solution

Figure 3. 21 Traitement sémantique des solutions

La figure 3.21.b illustre un exemple de traitement sémantique. La solution donnée en entrée utilise deux matrices dont les cycles ont été initialisés. Le premier minimum obtenu correspond au graphe G_0 qui est indépendant (cf. figure 3.18). On lui assigne donc la valeur du cycle courant de sa matrice qui est la matrice 1 (temps dôexécution de G0 est égal à 0). Lôalgorithme continue itérativement jusqu'à ce que tous les graphes soient ordonnancés. La solution finale obtenue est donnée en bas de la figure 3.21.b.

La solution modifiée est ensuite évaluée selon les fonctions objectives ainsi que la fonction de contrainte. En général le traitement sémantique réduit les solutions ayant des violations de contraintes, mais la reproduction peut en générer d'autres.

b. Reproduction et évaluation

À chaque itération de l'algorithme, la population est reproduite afin de diversifier les solutions. Trois opérateurs génétiques sont utilisés : la sélection, le croisement et la mutation.

- L'opérateur de sélection sert à choisir deux solutions qui vont être croisées pour créer de nouvelles solutions. La sélection choisie pour cet algorithme est celle par tournoi (Tourment selection). Elle a été décrite dans le chapitre 2. L'avantage de cette méthode de sélection est qu'elle permet de choisir n'importe quel individu sans le risque de négliger les mauvaises solutions. Ce qui est souhaitable puisque la reproduction avec de mauvaises solutions peut guider vers des solutions meilleures ;
- Le croisement binaire simulé a été choisi pour ce problème. Il a été décrit dans le chapitre 2. Ce type de croisement a été comparé plusieurs fois à d'autres opérateurs et s'est révélé très performant (Deb, et al., 2001) (Raghuwanshi, et al., 2005) ;
- L'opérateur de mutation est appliqué sur les nouvelles solutions afin de les diversifier pour éviter qu'elles soient identiques aux solutions parentes. L'opérateur utilisé est la mutation non uniforme (Non Uniform Mutation). Elle consiste à changer une variable x_i d'une solution par une valeur tirée dans une distribution non uniforme. Elle permet une bonne exploration du domaine de solution et un excellent affinement des solutions.

La reproduction commence par la sélection par tournoi de deux solutions P_1 et P_2 dans la population actuelle. Le croisement binaire simulé est appliqué à ces deux solutions pour donner deux nouvelles solutions C_1 et C_2. À chacune des nouvelles solutions, on applique la mutation non uniforme pour la diversifier de ses parents. La figure 3.22 présente un exemple de reproduction. Les nouvelles solutions obtenues seront ensuite évaluées puis rajoutées dans la nouvelle population par un algorithme de gestion de solution.

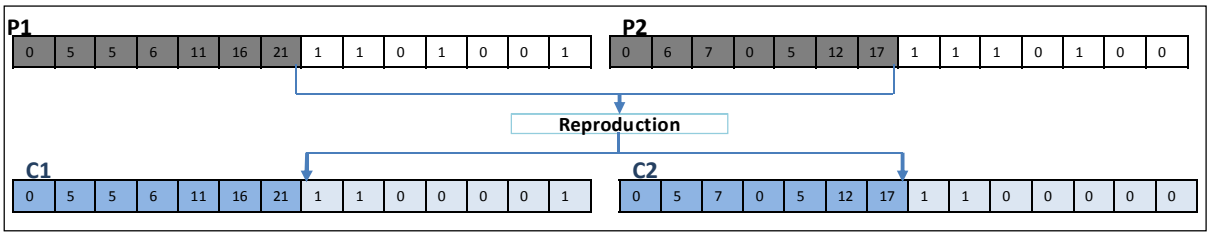


Figure 3. 22 Reproduction de solutions

c. Gestion des solutions

Dûne génération à une autre, la population est gérée par lôalgorithme NSGA-II présenté dans le chapitre 2. Les nouvelles solutions sont rajoutées à la population actuelle et lôensemble de toutes les solutions est trié et organisé en plusieurs fronts. La nouvelle population est créée à partir de cette liste triée. On rajoute les solutions du front 0 puis celle du front 1, ainsi de suite jusquôà ce que la nouvelle population soit mise en place (Nombre de solutions atteint) cf. chapitre 2. Le NSGA-II est lôun des meilleurs algorithmes pour résoudre les problèmes multi-objectifs (Durillo, et al., 2006; Deb, et al., 2001). En plus, il requiert très peu de paramètres de réglage.
Une fois que le nombre dôitérations passé en paramètre a été atteint, lôalgorithme fournit comme résultat le premier front de Pareto qui est lôensemble de solutions trouvées qui ne se dominent pas et qui ne sont pas dominées. Une solution est alors choisie parmi ces solutions et le code détaillé de lôexécution de lôapplication logique sur lôarchitecture nano-composante est donné en sortie du modèle de placement.

3.6. Résumé

Ce chapitre a présenté le modèle de placement proposé pour des architectures nano-composantes. Il prend en considération les possibilités dôexécution en pipeline et en parallèle et vise lôoptimisation de quatre métriques : le temps dôexécution, le coût de communication, le coût de reconfiguration et le nombre de cellules inactives. Le modèle de placement est réalisé en deux parties : la première partie permet de partitionner lôapplication, en entrée, en un ensemble de sous-fonctions en prenant en considération les caractéristiques des matrices et les métriques à optimiser. La seconde partie permet de déterminer le placement global des différentes sous-fonctions sur les matrices de lôarchitecture physique. Trois méthodes ont été utilisées :

- Une méthode d'assignation matricielle qui permet de placer une sous-fonction sur une matrice en utilisant des algorithmes de plus courts chemins et un parcours en largeur modifié ;
- Une méthode de partitionnement qui permet de diviser l'application en un ensemble de sous-fonctions pouvant être placées sur une matrice en utilisant un parcours en profondeur modifié et les opérations sur les graphes ;
- Enfin une méthode de placement global qui permet le placement global de l'application en utilisant les algorithmes génétiques.

Les deux premières méthodes sont utilisées dans la première partie du modèle de placement et la dernière dans la seconde partie. Le modèle fonctionne comme suit : une application est partitionnée en un ensemble de sous-fonctions, on estime le placement de chaque sous-fonction sur une matrice et on en déduit un placement global qui spécifie sur quelle matrice physique il faut placer chaque sous-fonction et à quel moment.

L'implémentation du modèle ainsi que les résultats obtenus sont présentés dans le chapitre suivant.

CHAPITRE 4. IMPLÉMENTATION, RÉSULTATS ET DISCUSSION

Ce chapitre décrit l'implémentation du modèle de placement proposé ainsi que les résultats obtenus. Il est organisé en cinq sections. Les environnements d'implémentation sont présentés dans la première section. La section 2 décrit l'implémentation du modèle de placement. Les circuits ayant servi de cas de test vont être présentés dans la troisième section. Les résultats de placement seront présentés et analysés dans la section 4 et la section 5 conclut le chapitre.

4.1. Environnements d'implémentation

Le modèle de placement a été implémenté en Java sous l'environnement Eclipse-Sun development Kit (SDK) (International Business Machines, 2006). Deux paquetages spécifiques ont été utilisés : Le JGraphT (Naveh, et al.) pour la théorie des graphes et le JMetal (Durillo, et al., 2006) pour les algorithmes génétiques.

1. **JGraphT :** Il s'agit d'une librairie gratuite qui fournit les objets et algorithmes des théories des graphes. Elle permet la représentation de grandes applications en supportant l'utilisation des millions d'arêtes et de nœuds dans un graphe. Plusieurs types de graphes sont supportés par cette libraire. Ceux qui ont été utilisés dans ce travail sont les graphes orientés et pondérés. Les algorithmes exploités sont les algorithmes de recherches de chemins, les algorithmes de recherches de voisins, l'algorithme de parcours en profondeurs. Le JGraphT est complété par une autre librairie gratuite JGraph (Alder, 2003) qui permet de visualiser les graphes traités.

2. **JMetal :** C'est un environnement de développement et d'expérimentation d'applications multi-objectives nécessitant des algorithmes métaheuristiques. Il fournit un ensemble de classes hiérarchiques qui peuvent être utilisées pour définir et implémenter n'importe quel problème multi-objectif. Toutes les applications utilisent les mêmes bases (opérateurs génétiques, solutions, problème, etc.) et spécifient leurs particularités par la suite. La figure 4.1 présente les classes de base de JMetal ainsi que les relations entre elles. Une *solution* à un problème multi- objectif (Bouchebaba, et al., 2010) est caractérisée par sa *variable de décision* qui est composée d'un ensemble de *variables* (entier binaire, réel,etc). Les solutions sont regroupées en population dans la classe *ensemble de solutions.* L'ensemble des solutions est géré par un *algorithme* de gestion

qui permet de résoudre un *problème* multi-objectif en utilisant des *opérateurs* génétiques (croisement, sélection, mutationé).

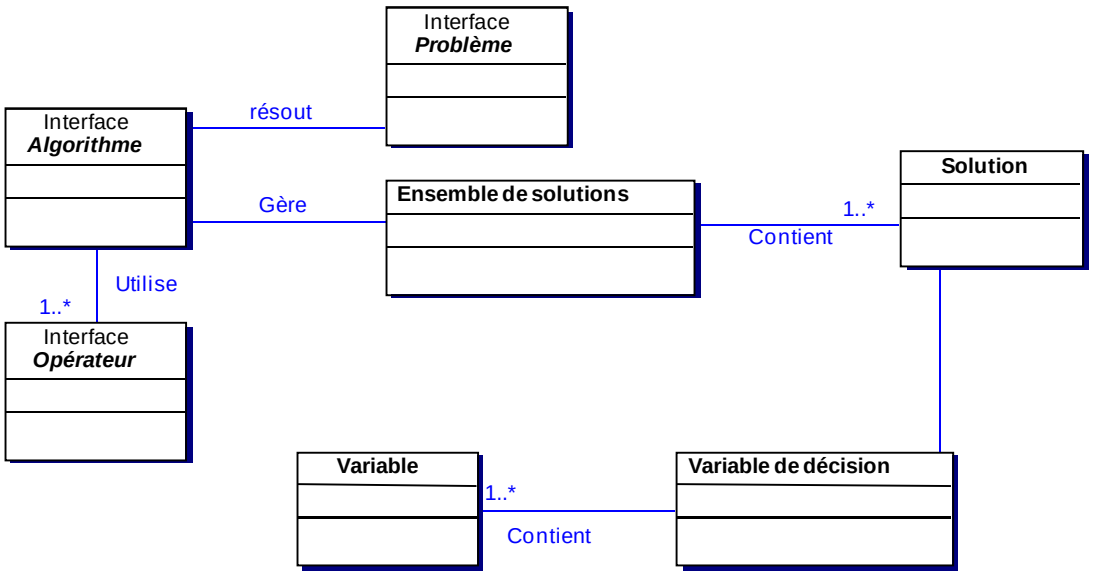


Figure 4. 1 Classes de base de JMetal

4.2. Description de lôimplémentation

La figure 4.2 présente le diagramme de classes de lôimplémentation du modèle de placement. Le paquetage *JGraphT* a été exploité pour lôutilisation des graphes et des algorithmes de recherches. Une interface *DAG* décrit le comportement global des graphes du modèle. Trois classes héritent de ce comportement :

- Le *graphe dôapplication* qui décrit les comportements des fonctions logiques en fonction du nombre dôopérations logiques et des dépendances entre elles;
- Le *graphe de matrice* qui spécifie le comportement des matrices en fonction de leur taille et des types de topologies;
- Le *graphe de dépendance* qui représente le comportement des sous-graphes et leurs de dépendances.

La classe *placement matriciel* contient toutes les méthodes nécessaires pour assigner un graphe dôapplication sur un graphe de matrice. Il sôagit de la classe qui permet de faire lôassignation matricielle. La classe *partitionnement* décrit le comportement de la méthode de

partitionnement dôun graphe dôapplication en tenant compte du type de matrice pour générer un graphe de dépendance. Cette classe est ensuite utilisée comme entrée pour le placement global

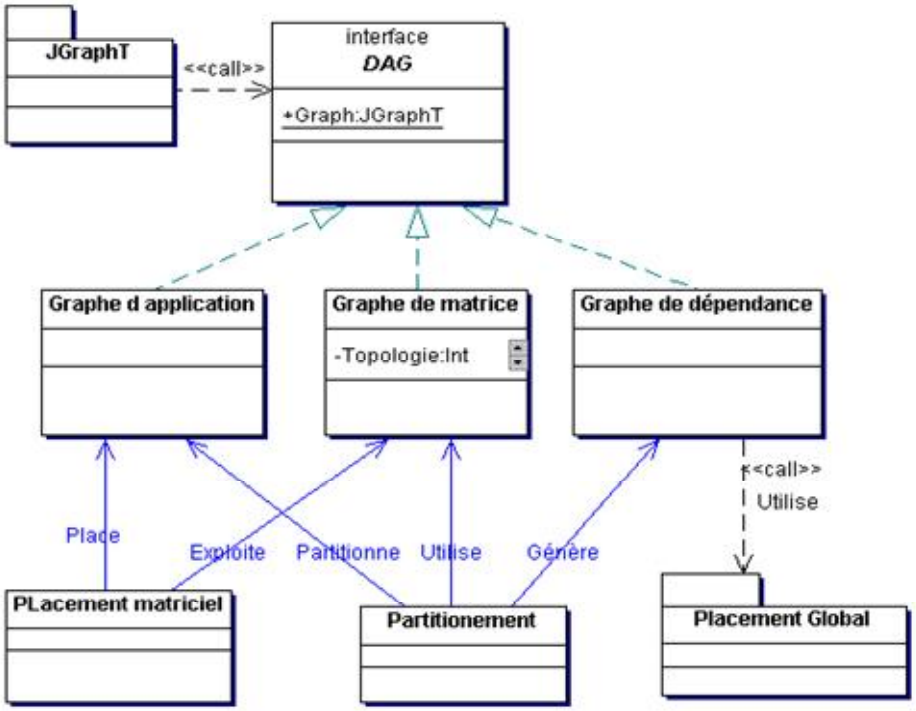


Figure 4. 2 Diagramme de classes du modèle de placement

Lôalgorithme du placement global a été défini dans lôenvironnement JMetal. Nous avons utilisé les classes de base du JMetal ainsi que des classes rajoutées pour permettre un codage en valeur entière des solutions (Bouchebaba, et al., 2010). Les classes rajoutées dans le cadre de nos travaux sont la classe de définition du *problème placement* et la classe de lôalgorithme de gestion basé sur lôimplémentation de NSGA_II de JMetal (*NSGA II pour le placement*). La figure 4.3 présente les classes utilisées lors du placement global.

Les classes *Int* pour le codage entier, *SBX_Crossover_Int* pour lôopérateur de croisement binaire Simulé des entiers et *NonUniform_MutationInt* pour lôopérateur de mutation non uniforme sont celles qui ont été rajoutées au JMetal. Les classes *NSGA II pour le placement* et *problème de placement* sont spécifiques au modèle de placement.

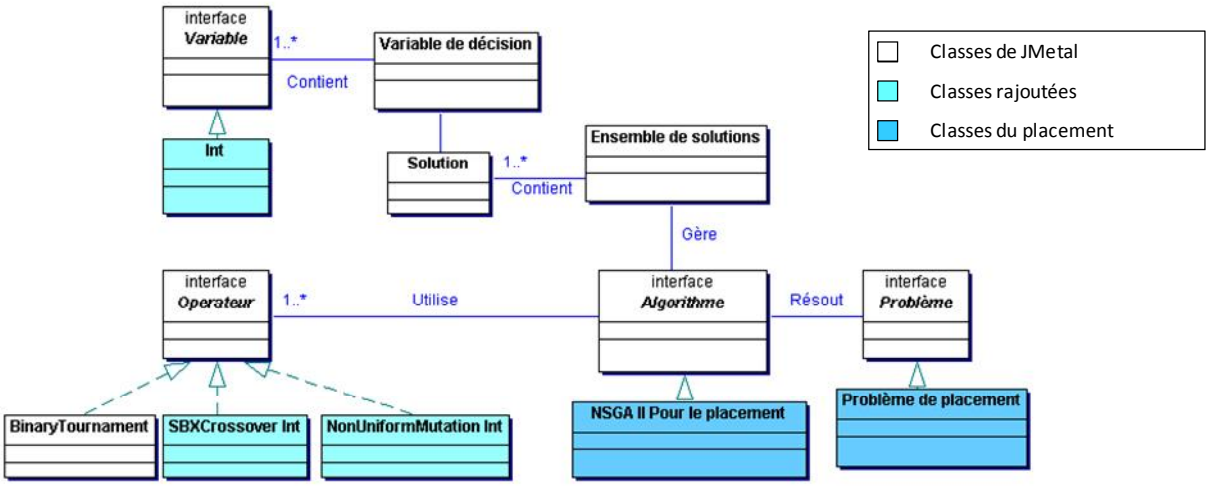


Figure 4. 3 Le paquetage du placement global

4.3. Bancs de test

Les circuits logiques utilisés pour les cas de test proviennent de XLINX-Virtex (XILINX, 2009) et du tutoriel VHDL (Zhang, 2001) pour ALU2. Ils permettent de faire des opérations standards. Le tableau 4.1 présente ces bancs de test ainsi que leurs descriptions.

Tableau 4. 1: Description des cas de test

Bancs de test	**Descriptions**
Unité Arithmétique et logique 2-bits (ALU2)	Permet de faire quatre opérations (Addition, soustraction, AND, OR) sur des entrées de 2 bits selon des bits de contrôles.
Additionneur 8 bits (ADD8)/Additionneur 16 bits (ADD16)	Additionneur par propagation de retenues. Il prend deux entrées de 8 bits/16 bits et une retenue (carry-In). Il produit la somme des deux entrées, une retenue (carry-Out) ou un dépassement (Overflow)
Additionneur&soustracteur 8 bits (ADSU8)/ Additionneur&soustracteur 16 bits (ADSU16)	Selon une entrée de contrôle ADD, le ADSU peut faire soit une addition ou une soustraction. ADSU8/ADSU16 prend deux entrées codées sur 8bits/16 bits, une retenue (carry-In) et lôentrée de contrôle. Ils produisent la somme ou la soustraction des deux entrées, une retenue (carry-Out) et/ou un dépassement (Overflow)
Comparateur 8bits (COMPM8)/ Comparateur 16bits (COMPM16)	Il permet de comparer deux nombres positifs. COMPM8/COMPM16 prend deux entrées (A, B) de 8bits/16 bits et produit deux sorties de 1 bit GT (Greater Than) et LT (Lower Than). GT est à 1 si A>B, LT est à 1 si A<B et les deux sont à 0 si A=B.

Pour placer des fonctions logiques sur les architectures nano-composantes, il faut les ramener au même niveau de granularité que ces architectures (au plus deux entrées de 1 bit). Les différentes transformations effectuées sont présentées en Annexe.

Les caractéristiques des cas de test, après transformations, sont présentées dans le tableau 4.2

Chaque fonction est caractérisée par le nombre d'opérations logiques, le nombre de dépendances entre les opérations, la largeur de la fonction ainsi que sa profondeur.

Tableau 4. 2 Caractéristiques des cas de test

Nom	Nombre d'opérations	Nombre de dépendances	Largeur	Profondeur
ALU2	45	64	9	14
CMPM8	59	75	16	11
ADD8	101	132	18	22
ADSU8	133	180	16	25
CMPM16	142	189	16	22
ADD16	97	260	33	38
ADSU16	261	356	33	42

4.4. Résultats et discussions

Les architectures nano-composantes sont paramétrables selon le nombre de cellules dans les matrices et le nombre de matrices dans le réseau. Pour explorer les possibilités de conceptions de systèmes nano-composantes en fonctions de ces paramètres, les résultats vont être analysés de deux façons :

- L'analyse des performances du partitionnement dans laquelle on discutera de l'impact de la taille des matrices sur le placement.
- L'analyse des performances de l'algorithme de placement global qui nous permettra d'analyser la pertinence des résultats obtenus et la variation du nombre de matrices dans le réseau.

4.4.1. Résultats du partitionnement et de lôassignation matricielle

Le partitionnement dôune application se fait par rapport à la taille dôune matrice donnée et aussi à la topologie de la matrice puisque chaque sous-fonction obtenue doit être placée sur une matrice. Cette sous-section présente les résultats des partitionnements et les facteurs à prendre en compte pour évaluer le partitionnement. Cinq métriques de performance ont été utilisées pour analyser les résultats :

- *Le nombre de sous-fonctions* Cette métrique nous permet dôévaluer la pertinence du partitionnement ainsi que le taux dôoccupation des matrices. Le partitionnement est fortement lié à la profondeur du circuit. Pour une matrice de taille *largeur ×profondeur*, si le plus long chemin dans un circuit est de taille m, on aura au minimum *(m/profondeur)* ou *(m/profondeur+1)* sous-fonctions, puisque le partitionnement est acyclique uniquement par rapport au plus long chemin.
- *Le nombre de dépendances* Il sôagit du nombre de dépendances entre les sous-fonctions obtenues.
- *Le nombre dôéchecs dôassignation matricielle* Lôéchec dôassignation survient lorsquôon nôarrive pas à placer une sous-fonction sur une matrice (cf. 3.3.1). La méthode dôassignation est utilisée lors du partitionnement pour simuler la transposition des sous-fonctions sur les matrices. Si une sous-fonction nôa pas pu être mappée sur une matrice, elle est réduite jusqu'à ce que lôassignation matricielle soit possible. Cette situation rend la métrique très importante dans les résultats du partitionnement.
- *Le taux de synchronisation :* Il sôagit du taux de suiveurs ajoutés lors de lôassignation matricielle. Les suiveurs permettent de faire passer les données dôune couche à une autre sans pour autant modifier le fonctionnement du circuit. Un grand nombre de suiveurs peut remplir la matrice et saturer les connexions prématurément.
- *Le taux dôoccupation des matrices* Cette métrique permet dôavoir une idée globale de nombre dôopérations placées sur la matrice sans tenir compte des éléments de synchronisation.

4.4.1.1. Partitionnement par rapport aux matrices de taille 4×4

Le tableau 4.3 présente les résultats du partitionnement par rapport aux matrices de taille 4×4. Le circuit ALU2 a été décomposé en 7 sous-fonctions soit un taux dôoccupation de matrice de 40,17 %. Malgré que le taux dôéléments de synchronisation soit faible, le retour dans lôétape de partitionnement, causé par lôéchec dôassignation matricielle a eu un impact considérable sur les résultats obtenus. En effet, lôéchec de partitionnement est en général causé par une incompatibilité entre la topologie de la matrice et les connexions des opérations logiques du circuit. Selon la connexion qui a causé lôéchec du placement, il peut entrainer de grandes réductions dans le sous-graphe généré. Ça a été également le cas pour les circuits CMPM8 et CMPM16 qui en plus des échecs matriciels, ont le plus grand taux de synchronisation dû à leurs structures (une grande partie de connexions entre les opérations traverse plusieurs niveaux). Le meilleur taux dôoccupation a été obtenu avec le circuit ADSU16, lors de son partitionnement, il nôy pas eu dôéchec dôassignation matricielle et il yô a eu moins dôéléments de synchronisation. Les circuits ADD8 et ADD16 ont également donné de bons résultats.

En résumé, sur les sept circuits expérimentés, le taux dôoccupation du partitionnement par rapport aux matrices 4×4 est en moyenne de 52,8 %, la moyenne du taux de synchronisation est de 7,2 % et il yôa eu 4.5 % de taux dôéchec.

Tableau 4. 3 Partitionnement par rapport aux matrices 4×4

Nom	Nombre de sous-fonctions	Nombre de dépendances	Nombre Échecs matriciels	Taux de synchronisations (%)	Taux dôoccupation des matrices (%)
ALU2	**7**	**9**	**1**	**7,17**	**40,17**
CMPM8	**9**	**15**	**2**	**24,44**	**40,97**
ADD8	**11**	**17**	**0**	**5,11**	**57,38**
ADSU8	**17**	**30**	**0**	**4,41**	**48,89**
CMPM16	**19**	**44**	**1**	**13,81**	**46,71**
ADD16	**21**	**36**	**0**	**5,05**	**58,60**
ADSU16	**27**	**49**	**0**	**4.39**	**60,41**

4.4.1.2. Partitionnement par rapport aux matrices de taille 4×6

Le tableau 4.4 présente les résultats du partitionnement par rapport aux matrices de taille 4×6 qui ont une plus grande profondeur que celles considérées dans la section précédente. Cela implique quêelles sont plus grandes et quêelles peuvent supporter des fonctions qui ont de plus grandes profondeurs. Comme la profondeur de la matrice est plus grande, le risque dêéchec de placement est également plus élevé, car les restrictions de connexions ont augmenté. Les Circuits CMPM8 et CMPM16 ont subi plusieurs fois lêéchec de lâassignation matricielle, ils ont également les plus grands taux de synchronisations et les plus petits taux dôoccupation. Les meilleurs résultats ont été obtenus par ADSU8 et ADSU16. Ils nôont subi aucun échec de placement ont des petits taux de synchronisations.

En résumé, sur les sept circuits expérimentés, le taux dôoccupation du partitionnement par rapport aux matrices 4×6 est en moyenne de 36,87 %, la moyenne du taux de synchronisation est de 11,8 % et il yôa eu 11,2 % de taux dêéchec.

Tableau 4. 4 Partitionnement par rapport aux matrices 4×6

Nom	Nombre de sous-fonctions	Nombre de dépendances	Nombre Échecs matriciels	Taux de synchronisations (%)	Taux dôoccupation des matrices (%)
ALU2	**6**	**8**	**1**	**9,02**	**31,25**
CMPM8	**10**	**15**	**4**	**18,33**	**24,52**
ADD8	**11**	**16**	**0**	**10,22**	**38,25**
ADSU8	**12**	**22**	**0**	**8,33**	**46,18**
CMPM16	**21**	**51**	**6**	**15,67**	**28,17**
ADD16	**21**	**38**	**1**	**11,70**	**39,08**
ADSU16	**25**	**56**	**0**	**9,16**	**43,5**

4.4.1.3. Partitionnement par rapport aux matrices de taille 6×4

Le tableau 4.5 présente les résultats du partitionnement par rapport aux matrices de taille 4×6 qui ont une plus grande largeur que celles considérées les sections précédentes. Cela implique quêelles sont plus grandes et quêelles peuvent supporter des fonctions qui ont de plus

grandes largeurs. CMPM8 et CMPM16 ont toujours le taux d'occupation le plus faible même si le nombre d'échecs d'assignation matricielle s'est réduit, ils restent également ceux qui ont le plus grand taux de synchronisation. ADSU8 et ADSU16 restent ceux qui ont le meilleur taux d'occupation. Même s'ils ont subi des échecs de partitionnement, leur structure réduit le dommage de la réduction lors des partitionnements.

En résumé, sur les sept circuits expérimentés, le taux d'occupation du partitionnement par rapport aux matrices 6×4 est en moyenne de 42,25 %, la moyenne du taux de synchronisation est de 7,6 % et il y'a eu 8,6 % de taux d'échec.

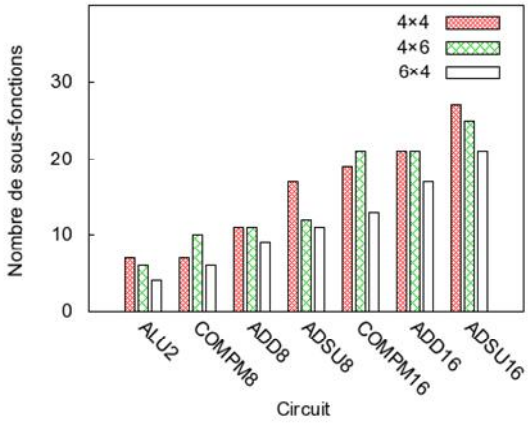
Tableau 4. 5 Partitionnement par rapport aux matrices 6×4

Nom	Nombre de sous-fonctions	Nombre de dépendances	Nombre Échecs Placement matriciel	Taux de synchronisations (%)	Taux d'occupation des matrices (%)
ALU2	**4**	**4**	**1**	**6,25**	**46,87**
CMPM8	**6**	**10**	**0**	**9,72**	**40,97**
ADD8	**9**	**14**	**1**	**3,7**	**46,75**
ADSU8	**11**	**19**	**2**	**7,19**	**50,37**
CMPM16	**13**	**37**	**1**	**13,46**	**45,51**
ADD16	**17**	**23**	**0**	**4,4**	**48,28**
ADSU16	**21**	**41**	**2**	**5,95**	**51,85**

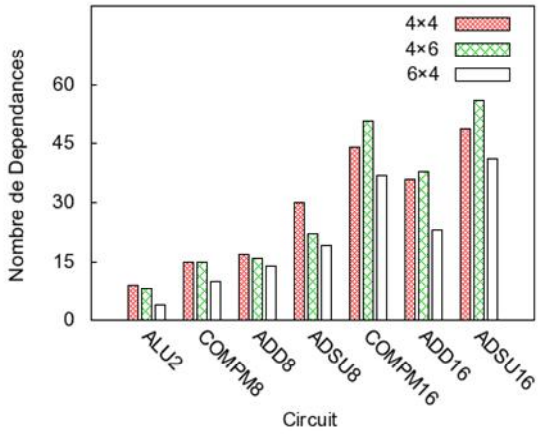
4.4.1.4. Analyse du partitionnement par rapport aux tailles des matrices

Cette section compare les résultats obtenus par chaque type de matrice afin de déterminer une taille de matrice adéquate pour les architectures nano-composantes. La figure 4.4 présente le nombre de sous-fonctions obtenues pour chaque circuit par rapport aux matrices considérées. Intuitivement, on s'attendra à ce que les matrices ayant le plus grand nombre de cellules aient le plus petit nombre de sous-fonctions même si le taux d'occupation n'est pas très grand. Ça a été le cas pour les matrices 6×4 qui, pour chaque circuit, réduit le nombre de sous-fonctions par rapport aux matrices 4×4. En effet, ces matrices ont le même nombre de couches que les matrices 4×4, donc a priori il n'y a pas de propagation de résultat en profondeur.

Les matrices 4×6 donnent des résultats différents selon les cas de test. Par rapport aux matrices 4×4, pour certains circuits, le nombre de sous fonctions a été réduit (ALU2, ADSU8, ADSU16), pour dôautres, il est stable (ADD8, ADD19) et il a augmenté pour CMPM8 et CMPM16. Cela est certainement dû à la grande profondeur des matrices qui augmentent la complexité du routage. Le même phénomène se produit pour le nombre de dépendances par rapport aux tailles des matrices.



a. Nombre de sous-fonctions par rapport aux types de matrices



b. Nombre de dépendances par rapport aux types de matrices

Figure 4. 4 Évolution de nombre de sous-fonctions par rapport aux tailles des matrices

Une comparaison globale des performances a été faite dans la figure 4.5. On y voit clairement que les taux d'occupation des matrices, de synchronisation et d'échec matriciel sont meilleurs pour les matrices 4×4. La matrice 4×6 donne le plus bas taux d'occupation et les plus grands taux de synchronisation et d'échec matriciel.

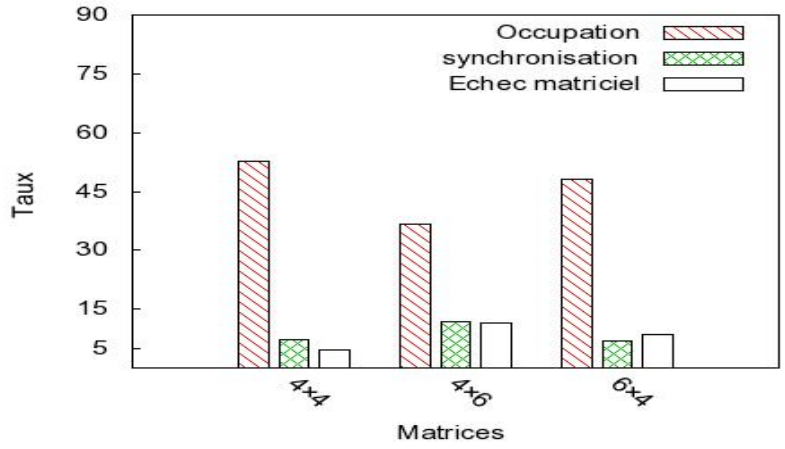


Figure 4. 5 Performances globales en fonction des matrices

Tout au long de cette analyse, nous avons remarqué que la structure des circuits a beaucoup d'impact sur les performances de partitionnement et de l'assignation matricielle. En effet, pour les trois types de matrice, les circuits CMPM ont subi des échecs matriciels et ont eu les taux d'occupation les moins élevés. Les meilleures performances ont été obtenues avec les ADSUs et les ADDs. La figure 4.6 illustre le taux d'occupation obtenue pour chaque banc de test sur les trois expérimentations.

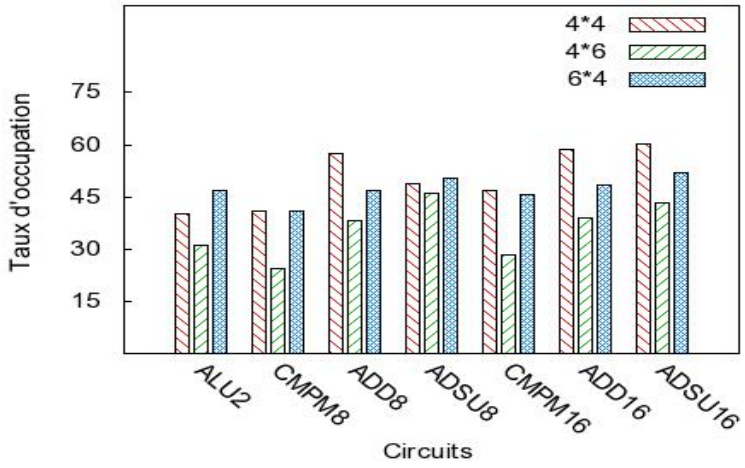


Figure 4. 6 Taux d'occupation par banc de test

4.4.1.5. Conclusion des résultats sur le partitionnement et lôassignation matricielle

La section 4.4.1 a présenté les résultats obtenus pour la partie de partitionnement et de placement matrice. On a parlé des facteurs qui rentrent en jeu dans le partitionnement et lôassignation matricielle en lôoccurrence la structure des circuits, le taux dôéchec matriciel et le taux de synchronisation. Une étude comparative des résultats par rapport aux tailles des matrices a montré que, pour le moment, avec les données dont on dispose, les matrices 4×4 semblent les plus adaptées pour les architectures nano composantes.

4.4.2. Résultats du placement global

Le placement global consiste à pouvoir définir lôordre et la matrice physique dôexécution de n sous-fonctions sur un réseau de m matrices. Ce placement doit prendre en compte les dépendances entre les sous-fonctions et la possibilité dôexécution en pipeline et en parallèle. Il doit permettre de minimiser le temps dôexécution, le coût de configuration, le coût de communication et le nombre de cellules inactives. Un algorithme génétique a été utilisé pour faire ce placement. Les résultats à ce niveau ont été analysés de deux points de vue différents :

- Les performances par rapport au nombre de matrices : Il sôagit dôutiliser les mêmes entrées avec différentes tailles de matrices et de déduire lôimpact du nombre de matrices sur les performances du placement ;

- Les performances par rapport aux graphes de dépendances donnés en entrées : Il s'agit d'analyser les performances obtenues sur différents circuits et de déduire l'impact des entrées sur les performances de placement. Chaque entrée est caractérisée par les informations sur les graphes de dépendances (Tableau 4.6).

Les métriques considérées à ce niveau sont les quatre objectifs d'optimisations, le nombre de matrices dans le réseau et les caractéristiques des circuits notamment la profondeur, le nombre de sous fonctions et le nombre de dépendances entre les sous-fonctions.

Tableau 4. 6 Les caractéristiques des graphes de dépendances

Nom	**Nombre de sous-fonctions**	**Nombre de dépendances**	**Profondeur**
ALU2	**7**	**9**	**4**
CMP8	**9**	**15**	**7**
ADD8	**11**	**17**	**7**
ADSU8	**17**	**30**	**8**
CMP16	**19**	**44**	**11**
ADD16	**21**	**36**	**11**
ADSU16	**27**	**49**	**12**

Les expérimentations ont été faites en fonction des matrices 4×4 car comme nous l'avons déduit des résultats du partitionnement, elles semblent les plus adaptées pour le placement. Les entrées du placement global sont les résultats obtenus lors du partitionnement c.-à-d. les graphes de dépendances et les informations d'assignation matricielle et de reconfigurations. Les caractéristiques de graphes de dépendances sont présentées dans le tableau 4.6. Il s'agit du nombre de sous-fonctions, le nombre de dépendances entre ces sous-fonctions et la profondeur du graphe de dépendance (le plus long chemin).

4.4.2.1. Les performances par rapport au nombre de matrices

Deux circuits ont été utilisés pour cette expérimentation : L'ADD16 et l'ADSU16. Ce choix est motivé par le fait qu'ils ont les plus grands graphes de dépendances conciliés à une grande largeur et une grande profondeur.

Lôalgorithme a été exécuté avec une population de 400 solutions en 400 itérations. Le placement a été fait sur un réseau de 2, 4 et 6 matrices.

La figure 4.7 présente des solutions de placement pour le circuit ADD16. Chaque solution y est représentée par une estimation du temps dœexécution, du coût de communication et du nombre de cellules inactives.

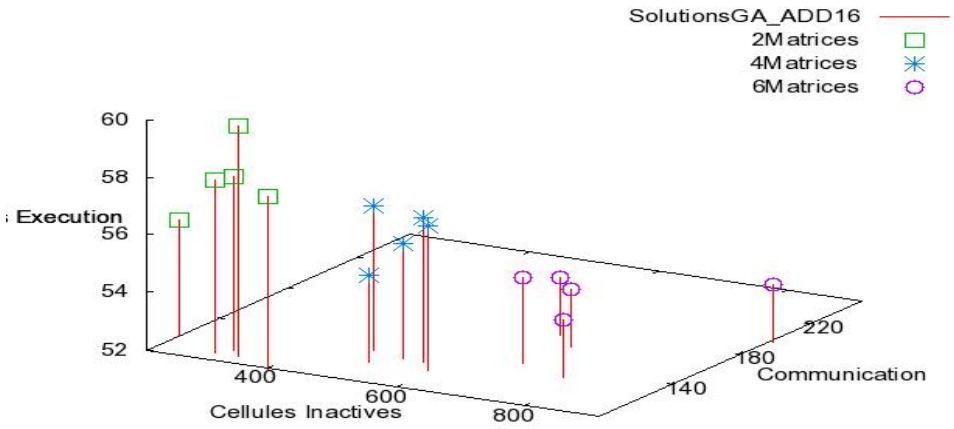


Figure 4. 7 Évolution des solutions ADD16 en fonction du nombre de matrices

En analysant lœévolution du nombre de matrices, on peut déduire trois conclusions :

a. Le temps dœexécution diminue en rajoutant des matrices Celui-ci diminue lorsque des sous-fonctions sont indépendantes, elles peuvent alors sœexécuter sur différentes matrices en parallèle. Ceci réduit le temps dœattente pour faire passer les sous-fonctions sur les mêmes matrices et rapproche de la situation idéale qui veut que seules les sous-fonctions dépendantes attendent la fin de lœexécution de leurs prédécesseurs.

b. Le coût de communication augmente en rajoutant des matrices La communication entre les matrices est évaluée par la distance de routage de données entre ces matrices. Cette distance est estimée en fonction de la localisation des matrices dans le réseau. Ainsi plus le réseau est grand, plus il yœ aura des grandes distances de routage si les matrices sont « géographiquement » éloignées. Ce qui explique la tendance vers un grand coût de communication pour les réseaux ayant de grands nombres de matrices.

c. Le nombre de cellules augmente en rajoutant des matrices Le nombre de cellules inactives est évalué à chaque cycle pour chaque matrice. Dans un cycle où une seule matrice est occupée toutes les autres seront considérées comme inactives donc plus le réseau sera grand plus il existera de matrices inactives. Ceci n'est pas vraiment un inconvénient puisqu'il y a possibilité de placer plusieurs fonctions en même temps sur le réseau.

La figure 4.8 présente des solutions de placement pour le circuit ADSU16. On y remarque les mêmes tendances que dans le cas du circuit ADD16.

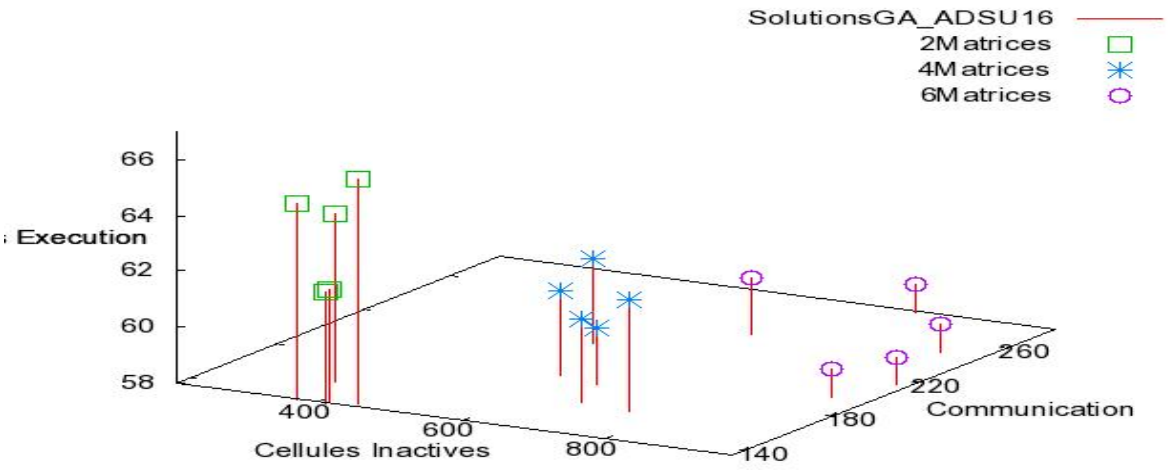
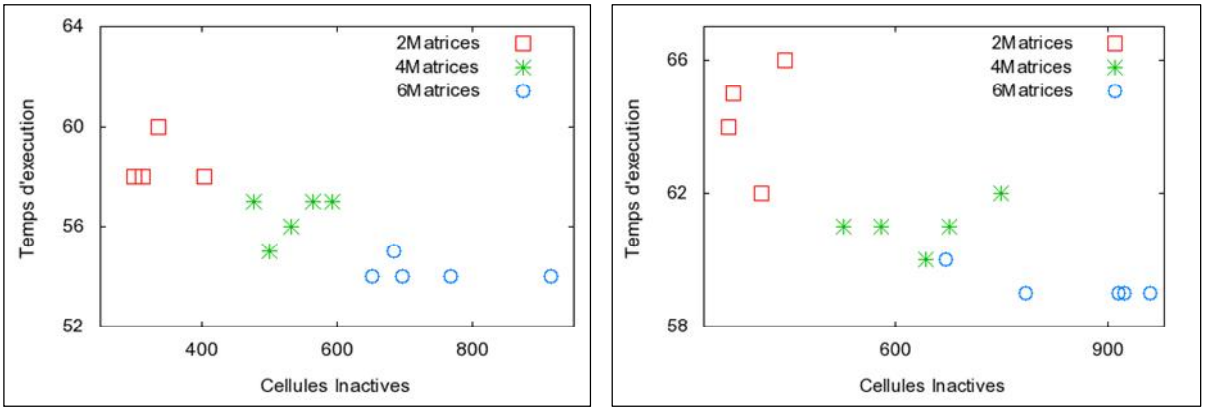


Figure 4. 8 Évolution des solutions ADSU16 en fonction du nombre de matrices

Les objectifs d'optimisations du placement global sont plus ou moins contradictoires, le choix des solutions dépendra alors du contexte d'utilisation et du compromis souhaité. L'impact du nombre de matrices a été étudié afin de déduire, s'il y a lieu, les relations entre les objectifs.

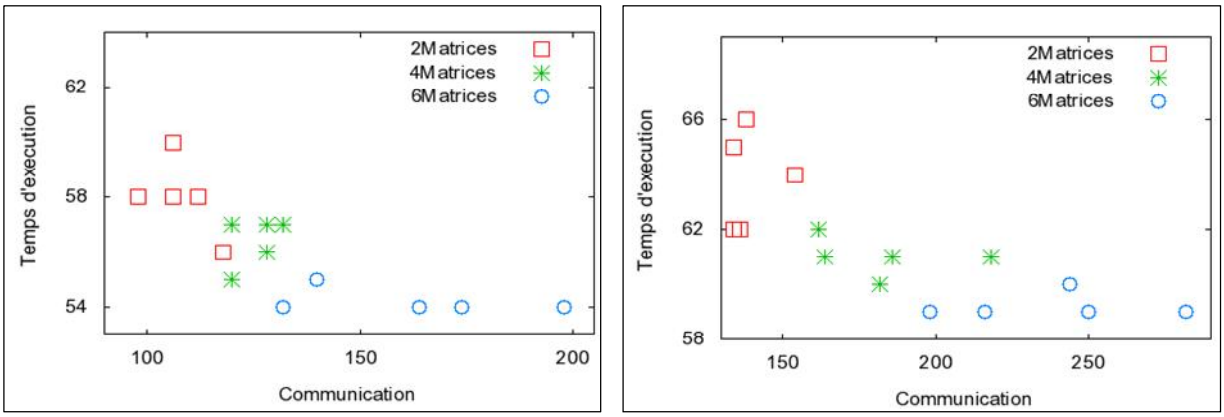
a. Temps d'exécution et Nombre de cellules inactives L'analyse précédente nous a permis de déduire que la taille de la matrice est inversement proportionnelle au temps d'exécution alors qu'elle est proportionnelle aux nombres de cellules inactives. La figure 4.9 présente plus clairement la relation entre ces deux objectifs. En effet, les solutions ayant réduit le temps d'exécution sont celles qui ont le plus grand nombre de cellules inactives.



a. Solutions ADD16 b. Solutions ADSU16

Figure 4. 9 Impacts sur le temps d'exécution et le nombre de cellules inactives

b. Temps dôexécution et Communication Le coût de communication grandit avec la taille du réseau. La figure 4.10 nous permet de voir clairement la relation entre le temps dôexécution et la communication. Les solutions ayant minimisé le plus le temps dôexécution sont celles qui ont le plus grand coût de communication.

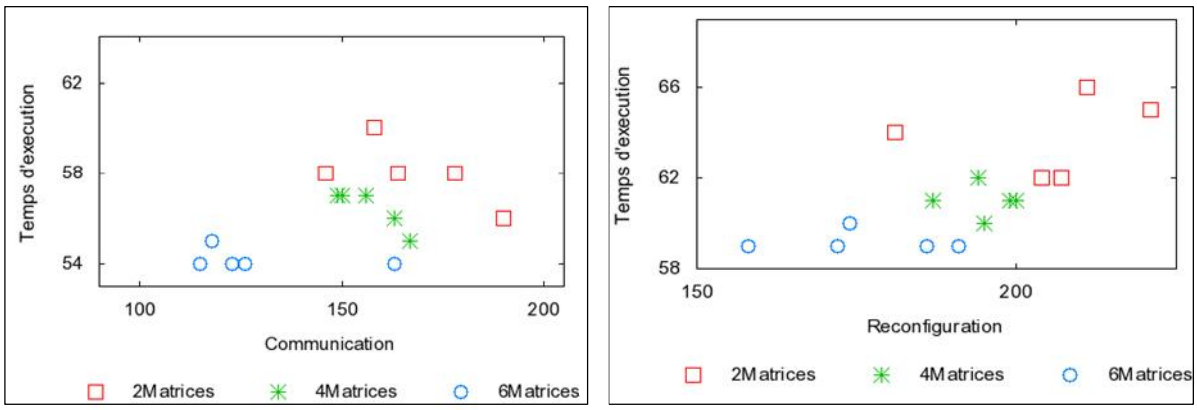


a. Solutions ADD16 b. Solutions ADSU16

Figure 4. 10 Impacts sur le temps d'exécution et la communication

c. Temps dôexécution et reconfiguration Le coût de reconfiguration est fortement lié aux opérations logiques. Plus lóassignation matricielle des sous-fonctions se ressemble structurellement moins, il yô aura de reconfiguration à faire (les cellules logiques feront les mêmes opérations dôun cycle à un autre). Comme la reconfiguration nôest pas directement liée à la structure du réseau, il est difficile de déduire une relation avec les autres objectifs. La figure 4.11 montre la relation entre le temps dôexécution et le coût de reconfiguration. On y

remarque que les solutions ayant le plus minimisé le temps d'exécution, ont moins de reconfigurations alors que les solutions prenant plus de temps d'exécution ont plus de reconfigurations. Mais il est difficile d'en déduire une relation puisqu'entre les deux on trouve des solutions ayant différents temps d'exécution avec le même coût de configuration et vice versa.



a. Solutions ADD16 b. Solutions ADSU16

Figure 4. 11 Impacts sur le temps d'exécution et la reconfiguration

4.4.2.2. Les performances par rapport aux entrées

Dans cette section nous analysons comment les caractéristiques des entrées (nombre de sous fonction, nombre de dépendancesé) peuvent avoir un impact sur les performances du placement global. Pour ce faire, nous allons analyser l'évolution des objectifs par rapport aux résultats obtenus sur chaque banc de test.

a. *Le temps d'exécution du placement* La profondeur du graphe est le plus long chemin qui existe dans le graphe. Quels que soient la parallélisation et le pipeline utilisés, le temps minimal d'exécution d'une fonction sera le temps d'exécution du plus long chemin. Si le réseau est composé de matrices de profondeur *Matprof*, pour tout graphe de profondeur *Grapro*, le temps minimal sera alors :

$$T(Matprof) = (Matprof+1)*Graprof - 1$$

La figure 4.12 présente la fonction du temps minimum en fonction de la profondeur du graphe de dépendance. Cette limite a été atteinte pour chacun des cas de test. Pour chaque cas de test, nous avons présenté la solution ayant minimisé, le plus, le temps d'exécution et on remarque

quêelles se trouvent toutes sur la droite obtenue par lãillustration de la fonction du temps minimum dêexécution (y= T (*Matprof*)).

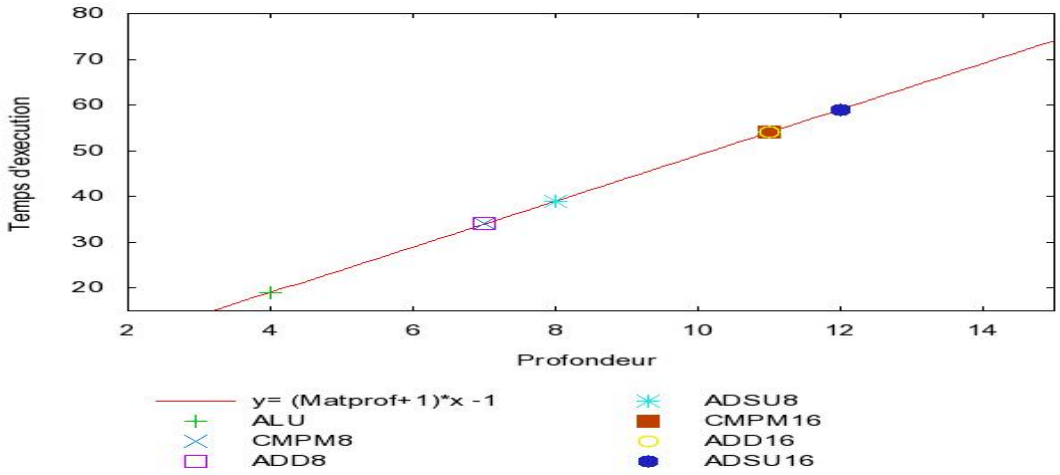


Figure 4. 12 Relation profondeur et temps d'exécution

b. *Le cout de communication du placement* Le coût de communication est fortement lié aux dépendances entre sous-fonctions. La figure 4.13 montre lêévolution du coût de communication en fonction des dépendances dans le cas de réseaux à 2 matrices et 4 matrices. Il est clair quãil existe une croissance du coût de communication lorsque le nombre de dépendances augmente. Aussi, même au sein dãun même réseau, avec le même nombre de dépendances, on peut avoir de bonne optimisation si les matrices qui communiquent sont « géographiquement » proches.

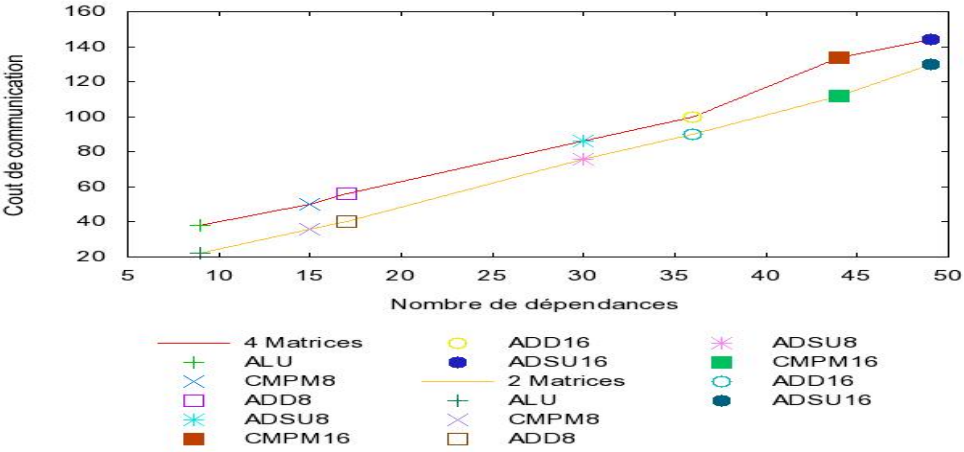


Figure 4. 13 Relation dépendances et coût de communication

c. *Le nombre de cellules du placement* La figure 4.14 illustre lô évolution du nombre de cellules inactives en fonction des dépendances. Plus il existe de dépendances moins il existe de matrices occupées à chaque cycle dô exécution. On y remarque aussi que lô écart entre le réseau de 2 matrices et le réseau de 4 matrices augmente au fur à mesure que les dépendances augmentent.

En plaçant plusieurs fonctions en même temps sur un même réseau, on va pouvoir optimiser encore plus cet objectif.

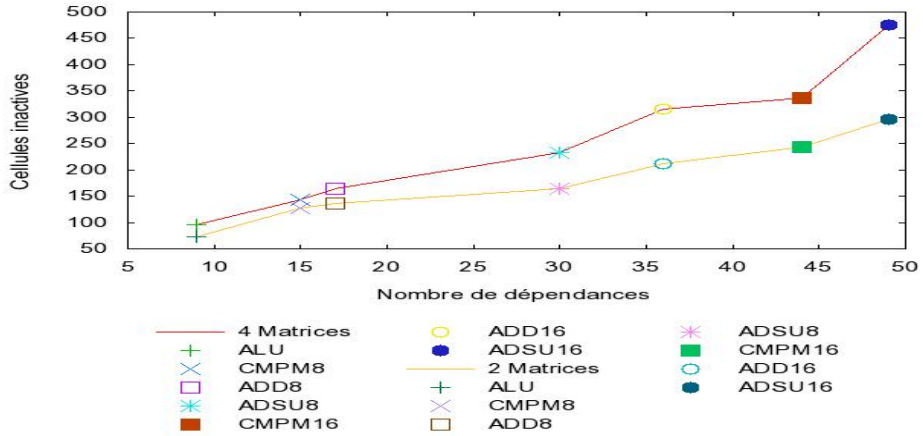


Figure 4. 14 Relation entre les dépendances et les cellules inactive**s**

4.4.2.3. Conclusion sur le placement globale

La section 4.4.2 a présenté les résultats obtenus pour le placement global. Lô analyse nous permet de déduire que plus il existe des matrices, plus on réduit le temps dô exécution. Par contre, le nombre de matrices dans le réseau est proportionnel au nombre de cellules inactives et au coût de communication. Sans définir une vraie relation, nous avons remarqué que le coût de configuration tend à diminuer pour les grands réseaux. Mais comme cet objectif est fortement lié aux sous-fonctions, il peut quand même avoir une variation. Lô analyse a également permis de déduire les relations entre les objectifs d'optimisations. Les expérimentations de cas de test donnent des solutions optimales qui peuvent atteindre le minimum de temps dô exécution. On a pu également déduire que le coût de communication et le nombre de cellules inactives augmentent avec lô accroissement des dépendances.

4.5. Résumé

Ce chapitre décrit l'implémentation du modèle de placement ainsi que les résultats obtenus et les analyses faites dessus. Le modèle a été implémenté en java en utilisant deux librairies : JGraphT et JMetal. Sept circuits ont été utilisés pour l'expérimentation. Ils ont été transformés afin d'atteindre la granularité fine des architectures nano composantes. Les résultats ont été présents sous deux angles : le partitionnement et l'assignation matricielle d'une part et le placement global de l'autre. Le premier a permis d'analyser les performances du partitionnement et de l'assignation matricielle ainsi que l'impact de la taille de la matrice sur ces performances. Le second a permis d'analyser les performances de l'algorithme génétique, l'impact de la taille du réseau de matrices ainsi que les relations entre les objectifs à optimiser.

CONCLUSION

Dans un contexte de miniaturisation accrue des transistors CMOS, lôITRS prévoit la limitation des technologies basées sur ces transistors pour 2020 (ITRS, 2007). Cette évolution a comme conséquence directe la définition de nouvelles technologies comme alternatives prometteuses de la technologie conventionnelle CMOS. Ainsi, les architectures nano-composantes ont fait leur apparition récemment. Les architectures nano-composante sont à un niveau de granularité très fin, ils permettent une reconfiguration dynamique des cellules logiques qui les composent et ont une structure hiérarchique à deux niveaux. Ces caractéristiques sont particulières aux architectures nano-composantes et pour prouver leur potentiel il nous faut évaluer leurs capacités et les possibilités quôelles offrent.

Cette situation impose dôavoir des outils de conception assistée par ordinateur (CAO) très performants qui permettront dôévaluer ces architectures et faciliterons le prototypage virtuel, la synthèse physique, la faible consommation.

Ce mémoire a proposé un modèle de placement prenant en comptes les caractéristiques des architectures nano-composantes et permettant dôoptimiser les métriques des architectures nano-composantes notamment le coût de communication, le coût de reconfiguration, le temps dôexécution des applications sur ces architectures, et de cellules logiques inactives.

Ce modèle représente la fondation dôun outil CAO qui permet dôune part, lôaide à la définition des architectures nano-composantes et de lôautre à leur exploitation efficace. Ce modèle est réalisé en suivant les étapes suivantes :

- la définition dôune méthode de partitionnement dôapplications en sous-fonctions en tenant compte des caractéristiques de lôarchitecture nano-composante ;
- le développement dôune méthode dôassignation matricielle pour associer chaque opération dôune sous fonction a cellule logique dans une matrice de lôarchitecture nano-composante, et
- la définition dôun modèle de placement global basé sur les algorithmes génétiques.

Quatre chapitres composent ce mémoire. Le premier chapitre a présenté une revue de littérature sur les méthodes de placement existantes leurs avantages et inconvénients ainsi que

lôoriginalité des méthodes définies dans ce travail. Les travaux sur les architectures nano-composantes ont aussi été présentés. Le second chapitre a présenté un état de lôart de tous les concepts utilisés dans le modèle. Il décrit les caractéristiques des architectures nano-composantes puis les concepts de bases sur la théorie des graphes et les algorithmes génétiques. Le troisième chapitre explique dôabord la formulation du problème de placement, puis il explique les différentes phases du modèle de placement ainsi que les méthodes qui y sont définies. Le dernier chapitre a décrit lôimplémentation du modèle et les résultats obtenus lors de sa validation. Les performances obtenues ont été évaluées, puis analysées en fonction des objectifs et des paramètres des architectures.

Travaux futurs

Les travaux effectués dans ce mémoire ouvrent la voie à plusieurs problématiques:

- Le partitionnement des applications en sous-fonctions est fortement lié aux réussites des assignations matricielles qui y sont effectuées. Il serait intéressant de travailler sur lôamélioration de lôassignation matricielle afin dôatteindre un taux de couverture qui tendra vers 100 %.
- Pour le placement global, nous avons utilisé un algorithme génétique basé sur une approche Pareto élitiste qui est très performante et donne dôexcellents résultats. Il pourra, quand même, être intéressant dôétudier lôimpact dôautres variantes dôalgorithmes génétiques pour voir sôils pourront donner de meilleures solutions.
- Il serait également très utile de permettre la réutilisation des placements existants. Ceci permettra dôaugmenter le taux dôoccupation des matrices.
- Nous pourrions définir une librairie de circuits de base avec des placements prédéfinis qui pourront être réutilisés lors des synthèses. Étant donné que le modèle de placement vise lôoptimisation de plusieurs ressources, cette librairie sera alors variée et diversifiée selon les ressources les plus importantes à optimiser.

BIBLIOGRAPHIE

Alder G The JGraph Tutorial [Report]. - [s.l.] : JGraph.com, 2003.

Alfaro-Cid E, McGookin E.W and Murray-Smith D.J A Novel Non-Uniform Mutation Operator and its Application to the Problem of Optimising Controller Parameters [Book]. - [s.l.] : Evolutionary Computation, 2005. The 2005 IEEE Congress on, 2005. - Vols. 1555- 1562 Vol. 2.

Alliot J and Durand N Algorithmes génétiques [Report]. - [s.l.] : http://pom.tls.cena.fr/GA/FAG/ag.pdf, 2005.

Betz V and Rose J A New Packing, Placement and Routing Tool for FPGA Research [Conference] // International Workshop on Field Programmable Logic and Applications. - 1997.

Bouchebaba Y [et al.] Mpassign: a framework for solving the many-core platform mapping problem [Book Section] / book auth. chapter Book. - [s.l.] : Springer, 2010.

Breuer M.A A class of min-cut placement algorithms [Conference]. - [s.l.] : Annual ACM IEEE Design Automation Conference, 1977.

Cong J and Xu D Exploiting Signal Flow and Logic Dependency in Standard Cell Placement [Conference] // Design Automation Conference. - 1995.

Cong J, Li Z and Bagrodia R Acyclic Multi-Way Partitioning of Boolean Networks [Conference] // Design Automation. - 1994.

Cox G. W and Carroll B. D The Standard Transistor Array (star) (Part II automatic cell placement techniquesò, [Conference] // Annual ACM IEEE Design Automation Conference. - 1986.

Deb K [et al.] A Fast Elitist Non-Dominated Sorting GeneticAlgorithm for Multi-Objective Optimization: NSGA-II [Conference] // IEEE Trans Evol Comput. - 2002. - Vols. 6(3):149-97.

Deb K and Beyer Hans-G Self-Adaptive Genetic Algorithms with Simulated Binary Crossover [Book]. - [s.l.] : Evolutionary Computation, 2001. - Vol. 9.

Durillo J. [et al.] jMetal: a Java Framework for Developing Multi-Objective Optimization Metaheuristics [Report]. - [s.l.] : TECH-REPORT, 2006.

Fiduccia . C.M. and Mattheyses R.M. A linear-time heuristic for improving network partitions [Conference] // Annual ACM IEEE Design Automation Conference. - 1982.

Gould J.M and Edge T. M The Standard Transistor Array (star) part I A two-layer metal semicustom design system [Journal]. - Minneapolis : Annual ACM IEEE Design Automation Conference, 1980 ?ɋdz

Gräbener T and Berro A Optimisation multiobjectif discrète par propagation de contraintes [Journal]. - [s.l.] : JFPC 2008- Quatrièmes Journées Francophones de Programmation par Contraintes, 2008 ?ɋdz

Hagen L and Kahng A. B. Fast spectral methods for ratio cut partitioning and clustering [Conference] // Proc.ICCAD-91. - 1991.

Hanan M, Wolff P K. and Agule B J. Some experimental results on placement techniques [Conference]. - [s.l.] : ò, Annual ACM IEEE Design Automation Conference, 1978.

Hutcheson G.D The Economic Implications of Moore's Law, [Book Section] // Into The Nano Era. - [s.l.] : Springer Series in Materials Science, 2009.

International Business Machines Corp Eclipse Platform Technical Overview [Report]. - [s.l.] : http://www.eclipse.org/documentation/, 2006.

ITRS International Technology Roadmap for Semiconductors Emerging Research Devices [Report]. - [s.l.] : ITRS, 2007.

Kang S Linear ordering and application to placement [Conference] // Annual ACM IEEE Design Automation Conference,1983. - 1983.

Kernighan B.W. and Lin S. An e f f i c i e n t heuristic procedure for partitioning graphs [Journal]. - [s.l.] : Bell SFS. Tech. J. , vol. 49, 1970 ?ɋdz - pp. Z91-308.

Kling R. and Bannerjee P ESP: A new standard cell placement package using simulated evolution [Conference] // In Proceedings of the 24th DesignAutomation Conference. - 1987.

Konak A, Coit D.W. and Smithc A E Multi-objective optimization using genetic algorithms: A tutorial [Journal]. - [s.l.] : Reliability Engineering & System Safety, 2006 ?ɋdž - Vols. 91(9) : 992-1007.

Lau H.T. Algorithms On Graphs [Book]. - [s.l.] : TAB BOOKs Inc, 1989.

Le Beux S [et al.] Optimizing Configuration and Application Mapping for MPSoC Architectures [Report]. - [s.l.] : A apparaitre dans AHS 2009 conference, 2009.

Lin Y [et al.] High-performance carbon nanotube field-effect transistor with tunable polarities [Journal]. - [s.l.] : Nanotechnology, IEEE Transactions, 2005 ?ɋdž - Vol. 4.

Liu J [et al.] Design of a Novel CNTFET-based Reconfigurable Logic Gate [Journal]. - [s.l.] : Electronics letters, 2007 ?ɋdž

Liu J These: Architectures reconfigurables à base de CNTFET double grille [Book]. - [s.l.] : Ecole Centrale de Lyon, 2008.

Moore G.E Cramming more components onto integrated circuits [Journal]. - [s.l.] : Electronics Magazine , 1965 ?ɋdž

Naveh B and Contributors JGraphT [Online] // sourceforge.net. - W3C. - 2009 ?ɋdž 02. - http://jgrapht.sourceforge.net/.

O'Connor I [et al.] Dynamically Reconfigurable Logic Gate Cells and Matrices using CNTFETs [Conference] // IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Tozeur. - 2008.

Preas B and Lorenzetti M Physical Design Automation of VLSI Systems [Book]. - [s.l.] : The Benjamin/Cummings PUblishing Compogny, Inc, 1988.

Raghuwanshi M.M [et al.] Simulated Binary Crossover with Lognormal Distribution [Book]. - [s.l.] : Complexity International, 2005. - Vol. 12.

Ramineni N. and Chrzanowska-Jeske M. A routing-driven mapping for cellular-architecture FPGAs [Conference] // Circuits and Systems, 1993., Proceedings of the 36th Midwest Symposium . - 1993.

Robert M CAO de systèmes et circuits intégrés :du transistor à la complexité des systèmes [Report]. - [s.l.] : LIRMM, UMR CNRS /Université Montpellier 2, 2004.

Saab Y A Fast Clustering-Based Min-Cut Placement Algorithm With Simulated-Annealing Performance [Conference]. - [s.l.] : VLSI DESIGN, 1996.

Schaffer D Multiple Objective Optimization with Vector Evaluated GeneticAlgorithm. [Conference] // Proceedings of the FirstInternational Conference on Genetic Algorithm. - 1985. - Vols. p.93-100..

Sechen C and Sangiovanni-Vincentelli A. TimberWolt3.2: A new standard cell placement and global routing package [Conference] // Design Automatzon Conference. - 1986.

Sechen C. The TimberWolf3.2 Standard Cell Placement and Global Routing Program. [Report]. - [s.l.] : Userôs Guide for Version 3.2, Release 2, 1986.

Shahookar K and Mazumder P A genetic approach to standard cell placement using meta-genetic parameter optimization [Conference] // IEEE Trans. Computer-Aided Design. - 1990.

Shahookar K. and Mazumder P VLSI Cell Placement Techniques [Conference] // ACM Computing Surveys. - 1991. - Vols. Vol. 23, No. 2,.

Srivinas N. and Deb K. Multiobjective Optimization using Non-dominated Sorting in Genetic Algorithms [Conference] // Evolutionary Computation . - 1994. - Vols. 2(3):257-71.

Talbi E [et al.] A Hybrid Evolutionary Approach for Multicriteria Optimization Problems: Application to the Flow Shop [Journal]. - [s.l.] : Springer Berlin / Heidelberg, 2001 ?ɋdz - Vol. Volume 1993/2001.

Venkataramana K and Irith P GAFPGA: Genetic Algorithm for FPGA Technology Mapping [Conference] // Design Automation Conference. - 1993.

Wakabayashi S., Iwauchi N. and Kubota H. A hierarchical standard cell placement method based on a new cluster placement model [Conference] // Circuits and Systems, 2002. APCCAS '02. 2002 Asia-Pacific . - 2002.

XILINX Virtex-4 Libraries Guide for Schematic Designs [Report]. - [s.l.] : XILINX, 2009.

Zhang W VHDL Tutorial: Learn by Example [Report]. - 2001.

ANNEXE 1 ï Transformation des cas de test

Pour placer des fonctions logiques sur les architectures nano-composantes, il faut les ramener au même niveau de granularité que ces architectures. Lôarchitecture nano-composante utilisée est à granularité très fine. Une cellule prend deux entrées de 1 bit et donne deux sorties de 1 bit. La cellule logique permet de faire 14 opérations logiques selon les entrées de contrôle avec lesquelles elle a été configurée.

Trois types de transformations peuvent sôimposer :

Transformation1 : Il faut sôassurer que les opérations logiques prennent en entrées au maximum deux mots de 1 bit. Si par exemple dans le circuit il yôa une opération ET de trois bits (AND3), il va falloir la décomposer en deux opérations ET de deux bits (AND2). La figure I.1 (Transformation1) présente un exemple de ce type de transformation.

Transformation2 : la sortie dôune cellule est dupliquée une fois. Lorsque la sortie dôune opération logique doit être dupliquée plus dôune fois il va falloir passer par des éléments de synchronisation pour faire passer les données. La figure I.1 (Transformation2) présente un exemple de cette transformation. Lôopération logique passe son résultat à trois autres opérations. La donnée est passée à lôune des trois opérations logiques et à un opérateur de synchronisation qui la passe ensuite aux deux autres opérations.

Transformation3 : Lôopération OU exclusif (XOR) et son complément ne font pas partie des opérations de base de la cellule CNT présentée dans le chapitre 3. Lorsquôelles se trouvent dans un circuit, il va falloir les remplacer par leurs équivalences. La figure I.3 (Transformation3) présente un exemple de ce type de transformation.

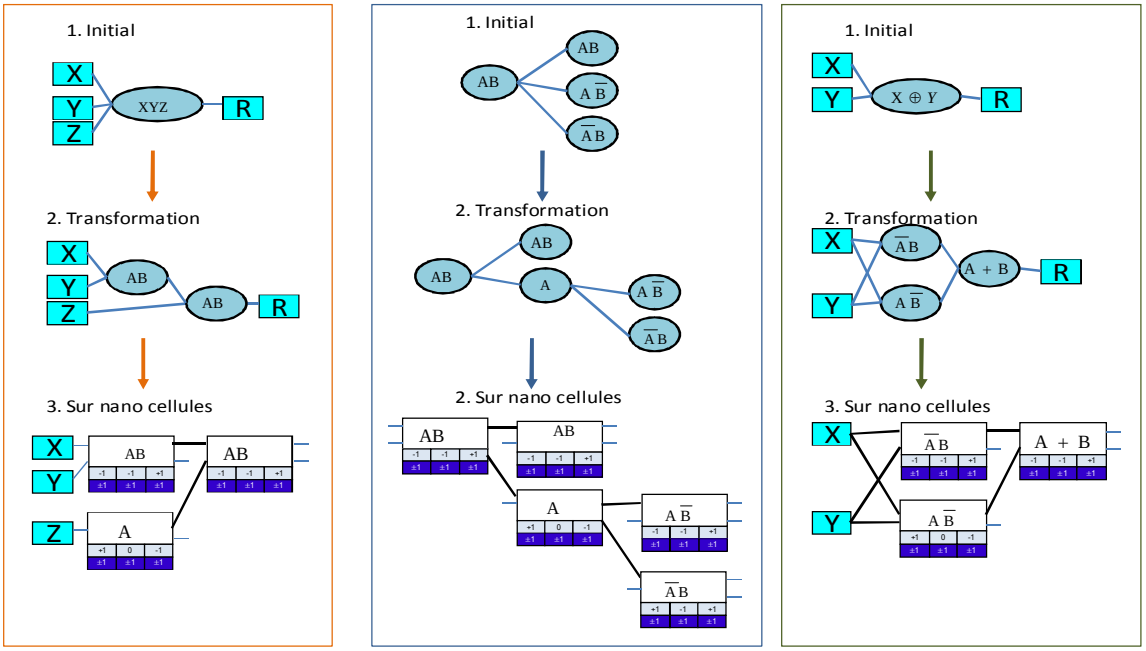


Figure I. 1 Transformations sur les fonctions logiques