MINISETÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ MOULOUD MAMMERI DE TIZI OUZOU

FACULTÉ DE GÉNIE ÉLECTRIQUE ET D'INFORMATIQUE

DÉPARTEMENT D'INFORMATIQUE

THÈSE DE DOCTORAT

SPÉCIALITÉ : INFORMATIQUE

Présentée par
Dihia BELKACEMI

Sujet

Mapping d'applications parallèles sur des architectures embarquées multiprocesseurs à base de réseaux sur puce

Devant le jury d'examen composé de :

M. Mustapha LALAM	Professeur	UMMTO	President
M. Djamel BENNOUAR	Professeur	U.BOUIRA	Examinateur
Mme. Rachida AOUDJIT	M. C. A	UMMTO	Examinatrice
Mme. Malika BELKADI	M. C. A	UMMTO	Examinatrice
M. Mehammed DAOUI	Professeur	UMMTO	Rapporteur
M. Youcef BOUCHEBABA	Docteur	ONERA	Invité

Soutenue le 06/10/2020

Abstract

To meet the growing requirements of today's applications, multiprocessor architectures (MPSoCs) interconnected with a network on chip (NoC) are considered as a major solution for future powerful embedded systems. One of the most critical challenges of a NoC-based MPSoC is how to map an application on this platform. Due to the large solutions' research space generated by both the application complexity and the platforms, this mapping phase can no longer be done manually, hence il requires powerful exploration tools called DSE (Design Space Exploration Environment). In this thesis, we propose several approaches included in an exploration tool for solving the problem of static mapping of parallel applications on NoC-based heterogeneous MPSoC. This tool has many advantages : (1) it integrates several multiobjective optimization algorithms that can be specified in order to explore different solutions' spaces, mainly : exact methods, metaheuristics (P-metaheuristics and S-metaheuristics) as well as new hybrid ones proposed in this thesis ; (2) it offers different cost functions (defined using analytical or simulation models) and constraints. The user can specify them or define others easily (easily extensible) ; (3) it provides an easy way to evaluate the performance of the Pareto front returned by different algorithms using multiple quality indicators. We also present a series of experiments by considering several scenarios,z therefore, we give designers guidelines on choosing the appropriate algorithm based on the characteristics of the mapping problem considered.

Keywords : Static Mapping, Multiobjective Optimisation, Network on Chip(NoC), Multi-processor System on Chip (MPSoC).

Résumé

Pour répondre aux exigences croissantes des applications actuelles, les architectures multiprocesseurs (MPSoCs) interconnectées avec un réseau sur puce (NoC) sont considérées comme une solution majeure des futurs systèmes embarqués de haute performance. L'un des défis les plus critiques d'un MPSoC basé NoC est de savoir comment placer (mapper) une application sur cette plateforme. En raison du grand espace de recherche de solutions engendré par la complexité conjointe des applications et de plateformes d'aujourd'hui, cette phase (mapping) ne peut plus être effectuée manuellement, d'où la nécessité d'outils d'exploration performants appelés DSE (Design Space Exploration Environment). Dans cette thèse, nous proposons plusieurs approches incluses dans un outil d'exploration pour la résolution du problème du mapping statique d'applications parallèles sur un MPSoC hétérogène basé NoC. Cet outil renferme plusieurs avantages : (1) il intègre plusieurs algorithmes d'optimisation multiobjectif pouvant être spécifiés afin d'explorer différentes solutions à savoir : des méthodes exactes, des métaheuristiques (les P-métaheuristiques et les S-métaheuristiques) ainsi que de nouvelles méthodes hybrides proposées dans le cadre de cette thèse ; (2) il offre de différentes fonctions de coûts (définies en utilisant un modèle analytique ou par simulation) et de contraintes. L'utilisateur peut les spécifier ou en définir d'autres et les intégrer facilement dans l'outil (facilement extensible) ; (3) il offre un moyen facile d'évaluer la performance du front retourné par différents algorithmes via plusieurs indicateurs de qualité. Nous présentons également une série d'expériences en considérant plusieurs cas de figures. Ainsi, nous donnons aux concepteurs des lignes directrices sur le choix de l'algorithme approprié selon les caractéristiques du problème du mapping considéré.

Mots clés : Placement statique, Optimisation multiobjectif, Réseaux sur puce, Multiprocesseurs sur puce.

Remerciements

Je tiens tout d'abord à remercier vivement M.Mustapha LALAM, professeur au Département d'Informatique de l'UMMTO et directeur du laboratoire de recherche en informatique de Tizi-ouzou (LARI) d'avoir mis à ma disposition les moyens du laboratoire ainsi que pour avoir accepté de présider le jury de soutenance.

Je remercie également M. Djamel BENNOUAR, professeur à l'université de BOUIRA d'avoir accepté de participer dans le jury d'examen.

Mes sincères remerciements vont également à Mme Rachida AOUDJIT et à Mme Malika BELKADI, M. C. A à l'université de UMMTO d'avoir accepté de participer dans le jury d'examen.

Ce travail n'aurait jamais abouti sans le soutien indéfectible de mon directeur de thèse, M. Mehammed DAOUI, professeur au Département d'Informatique de l'UMMTO. Je le remercie pour son encadrement, son aide, et ses encouragements tout au long de ces dures années de thèse.

Ce travail n'aurait également pas pu aboutir sans le soutien et la contribution de M. Youcef BOUCHEBABA, Docteur à ONERA, Palaiseau, France. Je lui exprime mes profonds remerciements de m'avoir orientée et d'avoir contribué à l'avancement de cette thèse.

Un grand merci à Madame Samia Bouzefrane, professeur au CNAM Paris, d'avoir accepté de m'accueillir dans son laboratoire pendant ma période de stage. Je la remercie pour l'aide précieuse qu'elle m'a apportée, ainsi que pour cette confiance qu'elle a pu me redonner dans les instants de doute.

Je voudrais exprimer ma gratitude à M.SOUMAYA, maître de conférences à Birla

Institute of Technology Mesra, Inde, pour son temps consacré pour évaluer mon travail. Ses conseils m'ont été d'un grand intérêt.

Je remercie également M.Ravi, qui a contribué à la lecture de mes travaux.

Enfin, mes vifs remerciements vont à ma famille et mes amies qui me soutiennent quotidiennement. Je vous remercie tous pour ce soutien affectif, sans vous, ce travail n'aurait jamais abouti.

A la mémoire de mon père

Table des matières

ABSTRACT **i**

Résumé **ii**

Remerciements **iv**

Liste des tableaux **xi**

Table des figures **xiii**

Liste des acronyms **xvi**

Introduction générale **1**

1 Concepts fondamentaux du MPSoC et du NoC **5**

1.1 Introduction 5

1.2 MPSoC (Multiprocessor System on Chip) 6

1.2.1 MPSoC homogènes 6

1.2.2 MPSoC hétérogènes 6

1.2.3 Les systèmes d'interconnexion dans un MPSoC 7

1.3 Le paradigme de réseau sur puce 9

1.3.1 Architecture du NoC 10

1.3.2 Communication dans un NoC 13

1.4 Les défis des MPSoCs : le problème de placement (Mapping) 18
1.4.1 Problème de placement (Mapping) 20
1.5 Conclusion 22

2 Optimisation multiobjectif et le problème du Mapping : Etat de l'art 23
2.1 Introduction 23
2.2 Optimisation multiobjectif 24
2.2.1 Concepts et définitions 24
2.2.2 Approches monoobjectif vs multiobjectif 28
2.2.3 Résolution d'un problème d'optimisation multiobjectif 29
2.2.4 Analyse de performance en optimisation multiobjectif 37
2.3 Approches pour la résolution du problème du mapping - Etat de l'art . . . 39
2.3.1 Discussion 42
2.4 Conclusion 43

3 Mapping des applications parallèles sur un MPSoC hétérogène basé NoC 44
3.1 Introduction 44
3.2 Description de l'outil proposé 44
3.2.1 Le modèle d'application 45
3.2.2 Le modèle d'architecture 47
3.2.3 Le placement (mapping) 47
3.3 Les métaheuristiques à base de population de solutions pour le problème du mapping 52
3.3.1 Représentation (ou codage) de la solution 52
3.3.2 Les opérateurs de mutation et de croisement 52
3.4 Les métaheuristiques à base de solution unique pour le problème du mapping 56

3.4.1 Représentation de la solution 56
3.4.2 Opérateur du voisinage 56
3.5 Evaluation expérimentale 57
3.5.1 Etude comparative des différents algorithmes 57
3.5.2 Discussion 62
3.6 Conclusion 65

4 Modèle de simulation **71**
4.1 Introduction 71
4.2 Modèle d'architecture considéré 71
4.3 Modèle de simulation proposé 73
4.3.1 Modèle de simulation à événements discrets 73
4.3.2 Les événements du modèle de simulation proposé 74
4.4 Estimation des métriques de performance avec le modèle de simulation proposé 82
4.4.1 Temps d'exécution total (Schedule Length) 82
4.4.2 Consommation d'énergie 82
4.5 Conclusion 83

5 Hybridation de métaheuristiques pour la résolution du problème de placement (Mapping) **84**
5.1 Introduction 84
5.2 Concepts sur l'hybridation 84
5.2.1 classification des méthodes hybrides 85
5.3 Approche hybride proposée 86
5.3.1 Hybridation des algorithmes évolutionnaires NSGAII [1], SPEA2 [2], PESAII [3], FastPGA [4] et IBEA [5] avec AMOSA [6] 87
5.3.2 Évaluation expérimentale de l'approche proposée 89

5.4 Discussion 96

5.5 Conclusion 97

Conclusion générale et perspectives **99**

A Le Framework jMetal **102**

B Algorithmes et suite des résultats expérimentaux des chapitres 4 et 5 **107**

Liste des tableaux

3.1 Paramétrage des algorithmes 58

3.2 Exemples de problèmes de mapping 59

3.3 Résultats des expériences du cas 1 63

3.4 Suite des résultats des expériences du cas 1 64

3.5 Test Friedman des expériences du cas 1 selon l'indicateur de qualité IGD . . 65

3.6 Résultats des expériences du cas 2 66

3.7 Suite des résultats des expériences du cas 2 67

3.8 Test Friedman des expériences du cas 2 selon l'indicateur de qualité IGD . . 68

5.1 Exemples d'instances du problème de mapping 90

5.2 Paramètres du NoC 91

5.3 Paramétrage des algorithmes 91

5.4 Indices statistiques. IGD. algorithmes hybrides vs algorithmes non hybrides (petite instance du mapping) 91

5.5 Indices statistiques. Epsilon. algorithmes hybrides vs algorithmes non hybrides (petite instance du mapping) 92

5.6 Indices statistiques. IGD. algorithmes hybrides vs algorithmes non hybrides (moyenne instance du mapping) 92

5.7 Indices statistiques. Epsilon. algorithmes hybrides vs algorithmes non hybrides (moyenne instance du mapping) 92

5.8 Paramétrage de l'algorithme HNSGAII 96

B.1 Valeurs de l'indicateur Epsilon selon la TABLE 5.5 114

B.2 Valeurs de l'indicateur Epsilon selon la TABLE 5.5 (suite) 115

B.3 Valeurs de l'indicateur Epsilon selon la TABLE 5.7 116

B.4 Valeurs de l'indicateur Epsilon selon la TABLE 5.7 (suite) 118

B.5 Indices statistiques. IGD. algorithmes hybrides vs algorithmes non hybrides (grande instance du mapping) 119

B.6 Indices statistiques. Epsilon. algorithmes hybrides vs algorithmes non hybrides (grande instance du mapping) 119

B.7 Paramétrage des algorithmes NSGAII et HNSGAII 119

B.8 Indices statistiques. epsilon. HNSGAII vs NSGAII (grande instance du mapping) 119

B.9 Indices statistiques. IGD. HNSGAII vs NSGAII (grande instance du mapping) 120

Table des figures

1.1 Exemples d'architectures multiprocesseurs (MPSoCs) 6

1.2 Systèmes d'interconnexion traditionnels dans les MPSoCs [7] 8

1.3 Architecture de système basée sur l'AMBA [8] 9

1.4 Elements de base d'un NoC [9] 10

1.5 Modèle de routeur générique. (LC = contrôleur de liaison) [10][9] 11

1.6 NoCs avec topologies régulières directes : (a) 2D-Mesh, (b) 2D-Torus, (c) hypercube, (d) octagone [11] 12

1.7 NoCs avec topologies indirectes : (a) arbre élargi , (b) multi-étages [11] . . . 12

1.8 NoCs à topologies irrégulières : (a) maille réduite, (b) cluster [11] 13

1.9 Commutation Store And Forward (SAF) [12] 15

1.10 Commutation Wormhole (WH) [12] 15

2.1 Espace de décision et espace objectif dans un PMO [13]. 25

2.2 Exemple illustrant la notion de dominance au sens de Pareto [14]. 26

2.3 Exemples de fronts de Pareto : (a) mauvaise convergence et bonne diversité, (b) bonne convergence et mauvaise diversité, (c) bonne convergence et bonne diversité [13]. 27

2.4 L'élitisme dans le multiobjectif [13] 29

2.5 Classification des méthodes de résolution multiobjectif [13][15]. 30

2.6 Exploration de X par une approche locale [16]. 31

2.7 Exploration de X par une approche évolutive [16]. 33
2.8 Principe d'un algorithme évolutionnaire (EA) [17]. 34
2.9 (a) principe de l'indicateur GD, (b) principe de l'indicateur IGD [18] 38

3.1 Entrées et sortie de l'outil proposé 46
3.2 Exemple d'un chromosome 52
3.3 Exemples de solutions invalides après avoir appliqué les opérateurs de reproduction 54
3.4 Codage intermédiaire 55
3.5 Exemple de solution invalide avec la nouvelle représentation après application de la mutation polynomiale (polynomial mutation). 55
3.6 Solution invalide corrigée avec l'algorithme 1 55
3.7 Retour au codage initial 56
3.8 Exemples de plateformes : (a) 2D-mesh, (b) 2D-torus, (c) Spidergon 59
3.9 Effet de paramétrage des algorithmes sur la qualité des solutions retournées 69
3.10 Un exemple illustrant le cas 4 70

4.1 Modèle du routeur utilisé 73
4.2 Exemple de modèle d'architecture 74
4.3 changement d'états des flits 78
4.4 Transfert flits de NI vers le routeur 79
4.5 Flit traverse la matrice de commutation (Switch) : (a) TraverserSwitch (j, Local, Est). (b) TraverserSwitch (j, West, Est). 81
4.6 Traverser le lien inter-routeurs 81

5.1 Classification hiérarchique des métaheuristiques hybrides [19][13] 86
5.2 Principe de l'approche d'hybridation proposée 88
5.3 Principe de clusterisation 89

5.4 Epsilon. algorithmes hybrides vs non hybrides : (a) petite instance du problème du mapping. (b) moyenne instance du problème du mapping . . . 94

5.5 Placement d'un graphe de 100 tâches généré aléatoirement sur une plateforme hétérogène avec six types de processeurs interconnectés en utilisant une topologie 2D-Torus (8x8). Deux fonctions de coût sont optimisées (temps d'exécution et énergie de consommation) 95

5.6 Étude de la sensibilité des paramètres d'un exemple d'algorithmes hybrides proposés (HNSGAII) : (a) variation de $MaxIter - AMOSA$ de HNSGAII, (b) variation du taux de refroidissement (α) de HNSGAII. 96

5.7 Placement d'un graphe de 100 tâches généré aléatoirement sur une plateforme hétérogène avec six types de processeurs interconnectés en utilisant une topologie Torus 8x8. Deux fonctions de coût sont optimisées (Load balancing et Communication) 97

A.1 Architecture générale de jMetal [20] 104

A.2 Architecture générale de l'outil de Mapping proposé 106

B.1 Algorithmes hybrides vs non hybrides pour une grande instances du mapping 117

Liste des acronymes

AG	Algorithme Génétique
AGA	Adaptive Genetic Algorithm
AMA	Adaptive Memetic Algorithm
ACO	Ant Colony Optimization
AbYSS	Archive-Based hYbrid Scatter Search
AMOSA	Archived Multiobjective Simulated Annealing
AMP	Asymmetric Multi-Processor
ASIP	Application Specific Instruction Set Processor
ASIC	Application Specific Integrated Circuit
ATG	Annotated Task Graph
BB	Branch & Bound (séparation et évaluation)
CWM	Communication Weighted Model
CTG	Communication Task Graph
CB	Credit Based
DSP	Digital Signal Processor
DSE	Design Space Exploration
EA	Evolutionary Algorithm
EC	Evolutionary Computation
ES	Evolution Strategy
EP	Evolution Programming
ES	Exhaustive Search (ES)
FastPGA	Fast Pareto genetic algorithm
FT	Finish Time
FO	Fonction Objective
FCFS	First Come First Serve
FIFO	First In First Out
GPP	General Purpose Processor
GP	Genetic Programming
GD	Generational Distance
GI	Greedy Incremental
GT	Graphe de tâches
HV	Hypervolume
HRH	High-level Relay Hybrid
HTH	High-level Teamwork Hybrid
HOL	Head Of Line
HPC	High Performance Computing
ITRS	International Technology Roadmap for Semiconductors
IBEA	Indicator-Based Evolutionary Algorithm
IGD	Inverse Generational Distance
IP	Intellectual Property
jMetal	java Metaheuristics algorithms
KPN	Kahn Process Networks

LCF	Largest Communication First
LRH	Low-Level Relay Hybrid
LTH	Low-Level Teamwork Hybrid
MPSoC	Multiprocessor System on Chip
MOCell	Multiobjective Cellular Genetic Algorithm
MOSA	Multiobjective Simulated Annealing
MOTS	Multiobjective TABOO SEARCH
MBB	Multiobjective Branch & Bound
MOEA	Multiobjective Evolutionary Algorithm
MOGA	Multiobjective Genetic Algorithm
MPI	Message Passing Interface
MoC	Model of Computation
MPPA	Massively Parallel Processor Array
NoC	Network on Chip
NI	Network Interface
NA	Network Adapter
NSGAII	Non-dominated Sorting Genetic Algorithm
OMOPSO	Optimized Multiobjective Particle Swarm Optimization
OEP	Optimisation par essaims particulaires
OpenMP	Open Multi-Processing
PESAII	Pareto Envelope-Based Selection Algorithm
PAES	Pareto Archived Evolution Strategy
PDA	Personal Digital Assistant
PMO	Problème d'optimisation Multiobjectif
P-metaheuristics	Population-solution based metaheuristics
PBBB	Pareto-based Branch and Bound
PBNMAP	Pareto-based NMAP
PM	Problème Mapping
PB	Priority Based
RR	Round Robin
SoC	System on Chip
SPEA2	Strength Pareto Evolutionary Algorithm 2
SMP	Symmetric Multi-Processor
SAF	Store And Forward
SMPSO	Speed-constrained Multi- objective PSO
S-metaheuristics	Single-solution based metaheuristics
SA	Simulated Annealing
SS	Scatter Search
SMPSO	Speed-constrained Multi-Objective PSO
ST	Start Time
TS	Tabu Search
TG	Task Graph
VCT	Virtual Cut Through
WH	Wormhole

Introduction générale

Contexte et problématique

Les applications actuelles (ex. le multimedia [21], le traitement de signal [22], les applications médicales [23][24], etc.) ont des exigences continuellement croissantes en termes de calcul. Afin de répondre à ces exigences, la première solution était d'augmenter la fréquence des processeurs. Or, l'augmentation de la fréquence engendre une grande consommation d'énergie, ce qui est contraignant dans ce contexte [25][26]. Grâce à la miniaturisation et le nombre accru de transistors pouvant être placés sur une puce [27][28][29], la multiplication du nombre de processeurs avec une faible fréquence sur une même puce est rendu possible. C'est ainsi que le paradigme de système multiprocesseurs sur puce MPSoC (MultiProcessor System on Chip) est né [30][31]. Les MPSoCs constituent une solution idéale à la fois en termes de puissance de calcul et de bilan énergétique [32][33][34][35]. Ces systèmes se distinguent en deux catégories : les MPSoCs homogènes et les MPSoCs hétérogènes.

Selon l'ITRS [36], les plateformes pourraient contenir plus de cinq mille PEs (ou cores) d'ici 2026. Toutefois, l'architecture d'interconnexion de ces PEs à travers des bus classiques devient vite obsolète et constitue un goulet d'étranglement à la communication et au partage de ressources.

Le Réseau sur puce (Network on chip) [37][38][39] est actuellement la solution d'interconnexion la plus prometteuse dans les systèmes actuels et ceux du futur [40], vu qu'il offre plus de flexibilité et supporte des débits élevés par rapport aux systèmes d'interconnexions traditionnels (ex. point à point, bus partagé).

Parmi les défis de la conception de ces systèmes (MPSoC basé NoC), on trouve le problème de placement (Mapping). Ce problème consiste à trouver un emplacement pour

les tâches d'une application (graphe de tâches) sur les ressources d'une plateforme donnée. Ce dernier est connu comme un problème NP-difficile [41]. Ceci est encore plus difficile lorsqu'il y a plusieurs facteurs souvent contradictoires à prendre en compte comme : la communication vs l'équilibrage de charge, la latence vs l'énergie de consommation, etc.

Les décisions du Mapping peuvent avoir une influence considérable sur la performance du système final. Selon le moment où cette décision est prise, le Mapping peut être dynamique (appelé dynamic ou on-line mapping) ou statique (appelé off-line mapping) [42]. Le mapping statique fait l'objet de nos travaux de recherches au cours de cette thèse.

Objectifs et contributions

Notre objectif est de proposer un ensemble d'approches permettant le mapping statique d'applications parallèles sur un MPSoC hétérogène basé NoC incluses dans un outil d'exploration. Comme les métaheuristiques sont de bons candidats pour résoudre cette problématique, nous avons implémenté notre outil sous le framework jMetal [43] afin de tirer profit de ses avantages tels que : (1) la multitude de métaheuristiques multiobjectif déjà implémentées que nous avons adaptées pour notre problématique. Cette variété de méthodes de résolution est un paramètre clé pour concevoir un outil d'exploration ; (2) la flexibilité offerte par ce framework nous a permis d'ajouter d'autres métaheuristiques afin d'enrichir notre outil et de faire d'éventuelles comparaisons ; (3) ce framework inclut un ensemble d'indicateurs de qualité que nous avons utilisé pour comparer entre les fronts de Pareto résultants.

Contribution

Nos principales contributions sont les suivantes :

- Nous avons adapté les différentes métaheuristques basées sur une population de solutions (P-métaheuristiques) implémentées dans le framework jMetal à savoir NSGAII [1], SPEA2 [2], PESAII [3], MOCell [44], IBEA [5], FastPGA [4], OMOPSO [45], AbYSS [46] et SMPSO [47] pour résoudre le problème du mapping d'applications sur un MPSoC hétérogène basé NoC [48].

- Nous avons implémenté de nouvelles méthodes de résolution à savoir : les S-métaheuristiques basées sur une seule solution telles que le récuit simulé multiobjectif (MOSA [49], AMOSA [6]), le MOTS [50] qui est une métaheuristique basée recherche taboue (MOTS) et une méthode exacte multiobjectif appelée Multiobjective Branch & Bound (MBB) [51]. Cette dernière est utilisée pour valider les résultats retournés par les métaheuristiques, et cela pour résoudre les problèmes de placement de petite taille.
- Nous avons proposé un modèle de simulation à événements discrets utilisé pour l'estimation de deux fonctions de coûts : (1) temps d'exécution et (2) énergie de consommation. Cela en tenant compte des effets dynamiques du système considéré.
- De nouvelles méthodes hybrides combinant entre les deux classes de métaheuristiques citées précédemment (P-métaheuristiques et S-métaheuristiques) ont été proposées dans le cadre de cette thèse, offrant ainsi à l'utilisateur un meilleur compromis répondant au mieux à ses exigences [52].

Le but est d'optimiser plusieurs objectifs (souvent contradictoires) lors du mapping des applications parallèles sur un MPSoC hétérogène basé NoC et de générer un ensemble de solutions de compromis selon les exigences et les métriques spécifiées.

Organisation de la thèse

Notre thèse est organisée en 5 chapitres et deux annexes (A et B). Les deux premiers chapitres sont consacrés au domaine d'étude.

Le chapitre 1 présente les concepts nécessaires à la compréhension générale de notre travail. Nous présentons dans un premier lieu, le concept du MPSoC et du NoC. Puis, nous explorons quelques défis de recherche des MPSoCs incluant le problème du placement (Mapping) qui fait l'objet de cette thèse.

Le chapitre 2 introduit d'abord les concepts de base de l'optimisation multiobjectif qui nous seront utiles pour modéliser et résoudre notre problème du Mapping. Puis, nous présentons un état de l'art des travaux de recherche concernant ce problème.

Les chapitres 3, 4 et 5 sont consacrés à nos contributions.

Le chapitre 3 présente l'outil proposé pour le placement statique des applications parallèles sur un MPSoC hétérogène. Après avoir donné une vue globale de l'outil, nous présentons l'adaptation de plusieurs méthodes de résolution à savoir les approches basées sur une population de solutions (appelées les P-métaheuristiques) ainsi que celles basées sur une seule solution (appelées les S-métaheuristiques). Nous concluons ce chapitre par une évaluation expérimentale pour chaque algorithme et une comparaison de leurs performances respectives.

Le chapitre 4 décrit le modèle de simulation proposé et comment ce dernier est utilisé pour l'estimation de deux fonctions de coûts à savoir : le temps d'exécution et la consommation d'énergie en présence de contentions.

Dans le chapitre 5, nous allons présenter les nouvelles approches hybrides proposées ainsi que leur évaluation.

Nous terminons cette thèse par une conclusion sur le travail et des perspectives.

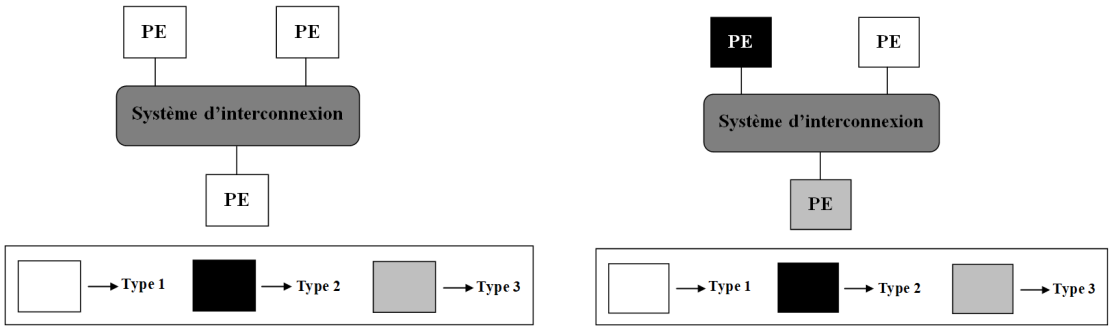
Les annexes A et B sont consacrées respectivement à la présentation du framework jMetal sur lequel nous avons implémenté l'outil du mapping proposé, ainsi que les algorithmes et la suite des résultats expérimentaux des chapitres 4 et 5.

Chapitre 1

Concepts fondamentaux du MPSoC et du NoC

1.1 Introduction

Pour pouvoir répondre aux exigences grandissantes des applications embarquées actuelles, la conception moderne des systèmes sur puce (SoC) montre une tendance claire vers l'intégration de plusieurs processeurs (PEs ou cores) sur une seule puce. Ces systèmes sont appelés Multiprocesseur sur puce (MPSoC). L'architecture d'interconnexion de ces systèmes de plus en plus complexes et intégrant un grand nombre de processeurs nécessite un nouveau système d'interconnexions plus évolué que les interconnexions traditionnels tels que : le point à point et le bus partagé. Ce nouveau système est appelé le réseau sur puce ou Network on Chip (NoC). Ce chapitre détaille ces deux concepts cités à savoir le MPSoC ainsi que le réseau sur puce (NoC). Tout d'abord, nous présentons le MPSoC dans la section 1.2, y compris sa définition, ses types et les divers systèmes d'interconnexions. Le NoC sera détaillé en section 1.3, en donnant sa définition, son architecture de base et ses protocoles de communication. Et enfin dans la dernière section 1.4, nous présentons un aperçu général sur les défis de recherche du MPSoC dont le problème du mapping qui sera l'objet de notre travail de recherche.



(a) Exemple d'un MPSoC homogène (b) Exemple d'un MPSoC hétérogène

FIGURE 1.1 – Exemples d'architectures multiprocesseurs (MPSoCs)

1.2 MPSoC (Multiprocessor System on Chip)

Les systèmes embarqués multiprocesseur (MPSoC pour Multiprocessor System on Chip) intègrent plusieurs éléments de traitement (PEs ou cores) sur une même puce de silicium [53][54]. Cette intégration a comme avantage la capacité du traitement parallèle avec une faible consommation d'énergie [32][33][34][35]. Selon le type de processeurs (PEs ou cores) utilisé, ces systèmes sont classés en deux catégories : les MPSoCs homogènes et les MPSoCs hétérogènes.

1.2.1 MPSoC homogènes

Les MPSoCs homogènes appelés aussi systèmes multiprocesseurs symétriques (SMP) sont des systèmes dans lesquels le même type d'unité de traitement (PEs ou cores) est utilisé (voir Figure 1.1 (a)). Un exemple typique de cette architecture est : le processeur multicores TILEPro64 de Tilera [55] doté de 64 cores identiques et bien d'autres [56][57]. Les MPSoCs homogènes sont flexibles, évolutifs et peuvent exécuter plusieurs types d'applications ; mais ils ne sont pas économes en énergie. Donc, ils conviennent mieux pour les consoles de jeux vidéo, les ordinateurs de bureau, les serveurs et les supercalculateurs [58].

1.2.2 MPSoC hétérogènes

Contrairement aux MPSoCs homogènes, les MPSoCs hétérogènes appelés aussi systèmes multiprocesseurs asymétriques (AMP) consistent en l'intégration des éléments de

traitement (PEs ou cores) de types différents (voir Figure 1.1 (b)) tels que des processeurs programmables (ex. General Purpose Processor, GPP), des éléments de traitement spécifiques à l'application (ex. Application Specific Instruction Set Processor, ASIP ou Application Specific Integrated Circuit, ASIC) et des éléments de traitement spécifiques au domaine (ex. Digital Signal Processor, DSP), des accélérateurs, etc. Cette hétérogénéité offre à cette classe de MPSoC une meilleure performance en termes d'énergie comparant aux MPSoC homogènes [59][35]. Tegra de NVIDIA [60], OMAP [61] de Texas Instrument et bien d'autres [56][57] sont des exemples illustrant de tels systèmes. En raison de leur bonne efficacité énergétique, les approches MPSoC hétérogènes sont utilisées pour les systèmes portables tels que les téléphone, les PDAs, etc.

En plus de cette classification décrite ci-dessus (basée sur le type de processeurs), une autre classification des MPSoCs est donnée en [62] selon la disposition de la mémoire. Deux types de systèmes multiprocesseurs sont distingués à savoir : les MPSoCs à mémoire partagée et les MPSoCs à mémoire distribuée.

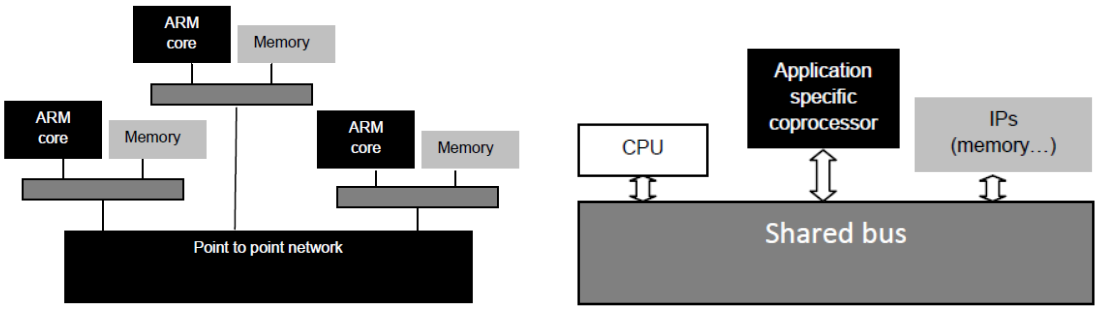
Qu'il soit homogène ou hétérogène, un MPSoC a besoin d'un système d'interconnexion pour pouvoir communiquer et véhiculer des données entre les différents PEs le composant. Dans la section suivante, nous présentons les différents systèmes d'interconnexion existants et expliquons comment ces derniers ont évolué pour répondre aux MPSoC d'aujourd'hui.

1.2.3 Les systèmes d'interconnexion dans un MPSoC

Différents systèmes d'interconnexions existent pour interconnecter les éléments composant un MPSoC dont les principaux sont : le point à point, le bus (partagé ou hiérarchique) et le réseau sur puce.

1.2.3.1 Connexion point à point

Cette solution est la première qui est apparue comme système d'interconnexion dans un SoC en général et dans un MPSoC en particulier. Elle consiste à interconnecter deux éléments communicant (PEs) via des liens dédiés [63] (Figure 1.2 (a)). L'avantage de cette solution est qu'elle est : susceptible de donner une meilleure performance en termes de latence et de débit, simple à mettre en œuvre et qu'elle ne nécessite aucun mécanisme



(a) Connexion point à point (b) Connexion par bus partagé

FIGURE 1.2 – Systèmes d'interconnexion traditionnels dans les MPSoCs [7]

d'arbitrage lors de la communication. Cependant, cette architecture est limitée en termes d'évolutivité et de flexibilité [64][7].

1.2.3.2 Connexion par bus partagé

La connexion par bus partagé (Figure 1.2 (b)) est un moyen de communication très répandu dans les SoCs pour interconnecter des unités de calcul. Venezia de Toshiba, ARM11 de ARM et bien d'autres [65] sont des exemples industriels des MPSoCs basé bus. Le bus sert à interconnecter un ensemble d'unités de traitement où une seule communication à la fois peut avoir lieu à un moment donné selon la politique d'arbitrage utilisée, ce qui limite le parallélisme et les performances de ce type d'architecture. De plus, par sa nature, le bus limite le nombre de PEs pouvant y être connectés [66]. A des fins d'amélioration du bus partagé standard en terme d'extensibilité, plusieurs variantes de bus ont été proposées dont le bus hiérarchique. Le bus hiérarchique consiste à interconnecter plusieurs bus partagés via des ponts (bridges). Cette approche est efficace en terme de puissance et peut être mise à l'échelle et permet des communications parallèles entre les PEs [67]. En revanche, l'accès d'un bus à un autre à travers un pont implique un coût en latence ; c'est pourquoi il est important de bien répartir les PEs communicants pour limiter l'utilisation du pont et ainsi favoriser le fonctionnement parallèle des bus [68]. Un exemple d'un MPSoC commercialisé basé sur ce type d'architecture est AMD de ARM [8][7] montré en Figure 1.3.

Bien que les connexions par bus (partagés ou hiérarchiques) soient plus flexibles et réutilisables par rapport à la connexion point à point, ce type d'interconnexions devient

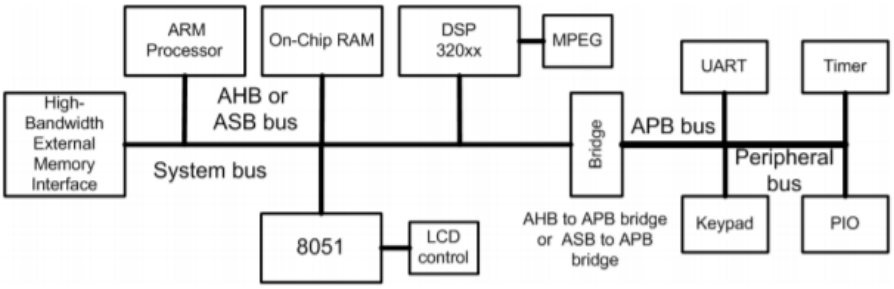


FIGURE 1.3 – Architecture de système basée sur l'AMBA [8]

vite limité et constitue un goulet d'étranglement à la communication et au partage de ressources, notamment quand le nombre de processeurs (ou cores) impliqué est grand (dépassant la dizaine [69]). Par conséquent, une nouvelle architecture d'interconnexion appelée réseaux sur puce (Network on chip) a été introduite pour pallier aux limitations des systèmes d'interconnexion citées précédemment [70].

1.3 Le paradigme de réseau sur puce

Le réseau sur puce (NoC) consiste à adapter les notions des réseaux informatiques dans le contexte des SoCs afin de réaliser des solutions d'interconnexions flexibles, modulaires et extensibles. Depuis sa proposition en Mars 2000 [71], le NoC [37][38][39][10][9] demeure la solution la plus prometteuse pour interconnecter un MPSoC avec un grand nombre de processeurs (ou cores) [72][40]. Des exemples de MPSoCs (many cores) commercialisés basés sur ce type de connexion (NoC) peuvent être cités tels que : le MPPA-256 de Kalray [73], Epiphany de Adapteva [74], TILEPro64 de Tilera [55], etc.

Dans ce qui suit, nous allons présenter en premier lieu l'architecture de base d'un NoC, incluant l'ensemble des éléments le composant ainsi que leurs dispositions (les topologies). En second lieu, nous donnons un aperçu des protocoles de communication au sein du NoC incluant : les modes de commutation, les protocoles de routage, les techniques d'arbitrage et les mécanismes de contrôle de flux.

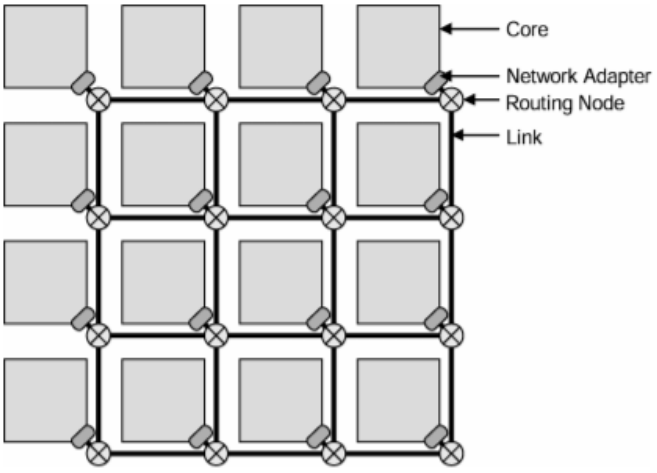


FIGURE 1.4 – Elements de base d'un NoC [9]

1.3.1 Architecture du NoC

1.3.1.1 Elements de base d'un NoC

Le réseau sur puce (NoC) est composé de trois éléments principaux : les liens, les routeurs et les adaptateurs réseaux [75][11] (voir Figure 1.4).

- Les liens : ils servent à relier physiquement les nœuds (routeurs) afin de mettre en œuvre la communication. Selon leur sens de transmission, ces liens peuvent être unidirectionnels ou bidirectionnels.
- Les routeurs : ils ont pour rôle d'acheminer les paquets entre les noeuds (source et destination) en fonction de la stratégie du routage implémentée. Un routeur est composé d'un certain nombre de ports d'entrée/sortie contenant ou pas des files d'attentes, d'une matrice de commutation servant à connecter les ports d'entrée aux ports de sortie, et un port local pour accéder aux unités de traitement (PEs) connectées à ce routeur. En plus de cette infrastructure de connexion physique, le routeur contient également une unité (logique) de routage et d'arbitrage, qui met en œuvre les algorithmes de routage (sélectionne la liaison de sortie pour un message entrant). Si plusieurs messages demandent simultanément la même liaison de sortie, ce composant doit prévoir un arbitrage entre eux. Si la liaison demandée est occupée, le message entrant reste dans la mémoire tampon d'entrée. Il sera à

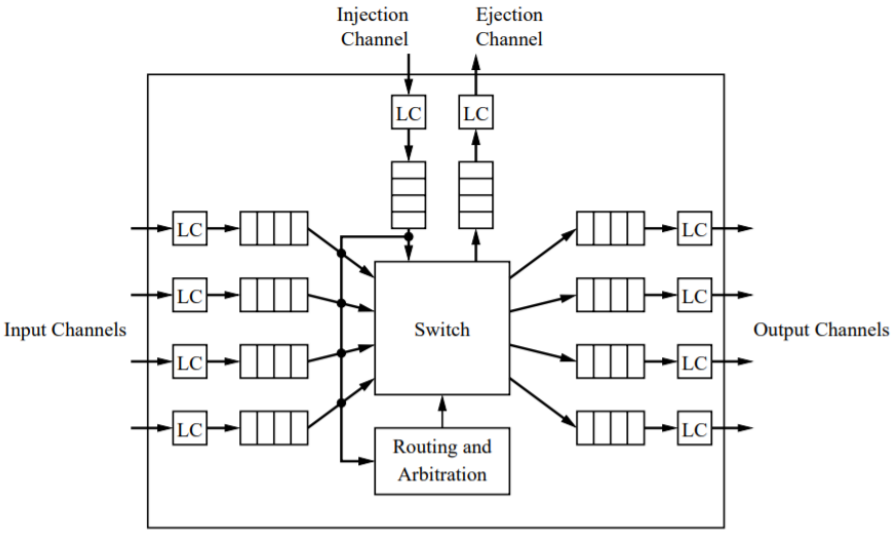


FIGURE 1.5 – Modèle de routeur générique. (LC = contrôleur de liaison) [10][9]

nouveau acheminé après que la liaison ait été libérée et s'il réussit à arbitrer pour la liaison. Le routeur implémente également des techniques de contrôle de flux. Un exemple d'un modèle de routeur générique est montré en Figure 1.5.

- Adaptateurs réseaux (NAs) : appelés aussi interfaces réseaux (NIs), ces blocs établissent la connexion logique entre les différents composants du MPSoC (PEs, IPs, DSP, blocs de mémoire intégrés, etc) et le réseau. Chaque PE peut avoir un protocole d'interface distinct par rapport au réseau. Ce bloc est important car il permet la séparation entre le calcul et la communication.

1.3.1.2 Topologie du NoC

La topologie du NoC détermine la disposition physique et les connexions entre les routeurs et les liens le composant [76]. Les routeurs peuvent être connectés en topologies directes ou indirectes [11].

- Topologies directes : dans les topologies directes, chaque routeur est associé à un processeur et cette paire peut être considérée comme un seul élément du système (appelé nœud dans le réseau). Dans cette topologie, chaque nœud est directement connecté à un nombre fixe de nœuds voisins et un message entre deux nœuds passe par un ou plusieurs nœuds intermédiaires. Seuls les routeurs sont impliqués dans

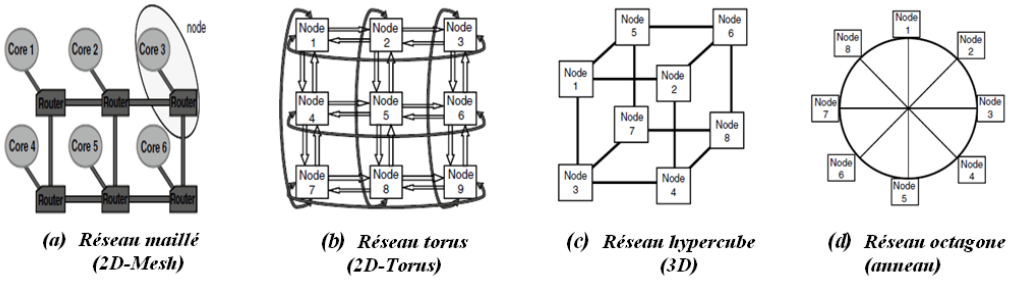


FIGURE 1.6 – NoCs avec topologies régulières directes : (a) 2D-Mesh, (b) 2D-Torus, (c) hypercube, (d) octagone [11]

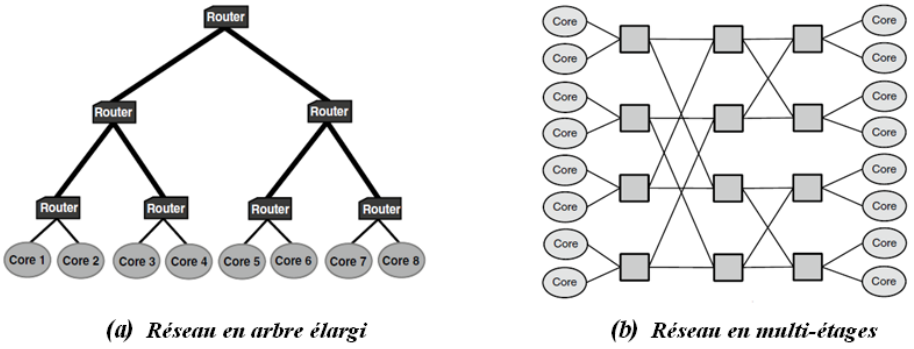


FIGURE 1.7 – NoCs avec topologies indirectes : (a) arbre élargi , (b) multi-étages [11]

la communication dans une topologie directe et cette communication est basée sur l'algorithme de routage mis en œuvre par les routeurs. Les avantages de ce type de topologies est leur évolutivité et leur structure réutilisable [77]. Les topologies directes les plus courantes sont la grille n-dimensionnelle ou maille, torus (ou k-ary n-cube), l'hypercube et octagone (anneau) comme montré en Figure 1.6.

- Topologies indirectes : dans une topologie indirecte, tous les routeurs ne sont pas connectés à des unités de traitement comme dans la topologie directe. Au lieu de cela, certains routeurs ne sont utilisés que pour propager les messages à travers le réseau. La Figure 1.7 montre deux exemples de réseaux indirects. L'avantage des topologies indirectes est qu'elles peuvent être plus facilement adaptées aux besoins spécifiques de l'application. Cela dépend de l'application cible qui peut nécessiter des contraintes de surface, de puissance ou de temps qui doivent être strictement respectés. Dans ce cas, utiliser les topologies directes peut ne pas être la bonne

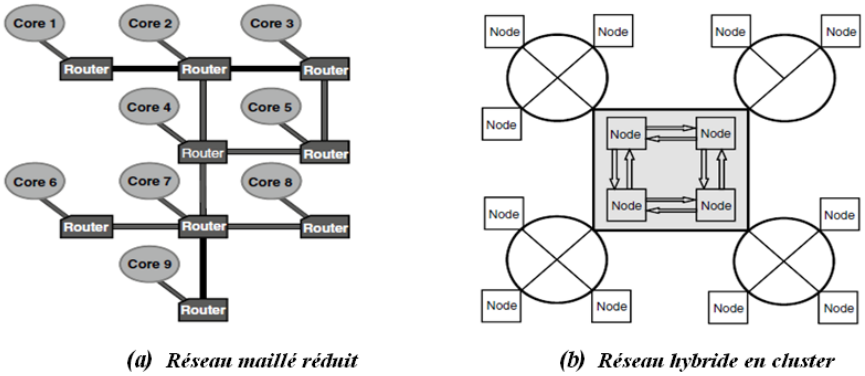


FIGURE 1.8 – NoCs à topologies irrégulières : (a) maille réduite, (b) cluster [11]

approche pour implémenter de telles applications spéciales. Par contre, les topologies indirectes sur mesure offrent une meilleure flexibilité pour répondre aux exigences souhaitées. D'autre part, l'un des principaux problèmes dont souffrent les topologies indirectes est le temps de conception nécessaire pour profiler l'application et décider de la meilleure disposition topologique qui satisfait ces exigences de conception [77]. Une autre classification possible pour les topologies de réseau (NoC) est liée à la régularité des connexions entre les routeurs. Dans les réseaux réguliers, tous les routeurs sont identiques en termes de nombre de ports. Par exemple, dans une topologie de grille régulière présentée en figure 1.6 (a), tous les routeurs ont cinq ports : un port local se connectant à l'unité fonctionnelle et quatre autres ports se connectant aux routeurs voisins. Dans les topologies irrégulières, les routeurs peuvent présenter différents schémas de connexion, généralement définis en fonction de l'application, comme le montre la Figure 1.8.

1.3.2 Communication dans un NoC

Cette section présente les techniques utilisées pour la communication au sein d'un NoC, à savoir : les modes de commutation, les techniques de routage, les politiques d'arbitrage et les mécanismes de contrôle de flux.

1.3.2.1 Modes de commutation

Les techniques de commutation déterminent quand et comment les données sont transmises du nœud source au nœud cible (destinataire). Il existe deux principaux types de commutation dans le NoC : commutation par circuit et commutation par paquet [12][11][78][77].

- Commutation par circuit (circuit switching) : dans cette commutation, le chemin physique du nœud source au nœud cible est préalablement établi et réservé avant la transmission des données. L'avantage de cette commutation est qu'elle offre certaines garanties de rendement, car la donnée est sûre d'être transférée à sa destination sans qu'il soit nécessaire de la mettre en mémoire tampon. Cependant, ce type de commutation augmente la latence et cela est dû au temps requis pour l'établissement d'un chemin, en plus de l'encombrement supplémentaire causé par les différentes données de contrôle circulant sur le réseau, qui à son tour affecte la bande passante du réseau [77].
- Commutation par paquet (packet switching) : cette technique est largement utilisée dans les NoC. Elle consiste à partitionner le message en paquets de longueur fixe et les transmettre sans réserver l'intégralité de l'espace (i.e. le chemin). La transmission d'un paquet donné ne doit pas bloquer la communication des autres paquets sur le réseau. Pour cela, une politique de commutation est nécessaire. Elle définit comment les ressources réseaux doivent être allouées et libérées une fois que le transfert est terminé. Principalement, trois types de politiques de commutation se distinguent à savoir : Store And Forward (SAF), Wormhole (WH) et Virtual Cut Through (VCT) [12][11][78][77].
  1. Store And Forward (SAF) : en commutation SAF, chaque paquet est entièrement stocké en mémoire tampon FIFO avant d'être transmis au routeur suivant (voir Figure 1.9). Le principal inconvénient de la commutation SAF est la nécessité d'utiliser des tampons de grande taille, ce qui engendre en conséquence une grande surface de silicium en plus de la latence engendrée par l'attente de la réception de la totalité du paquet.
  2. Wormhole (WH) : dans cette politique, le paquet est divisé en plusieurs flits et

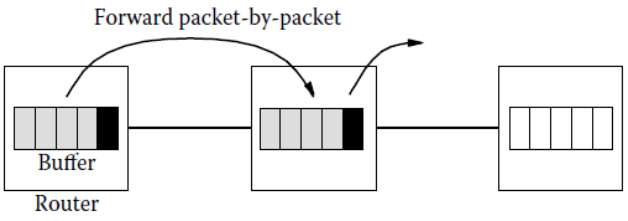


FIGURE 1.9 – Commutation Store And Forward (SAF) [12]

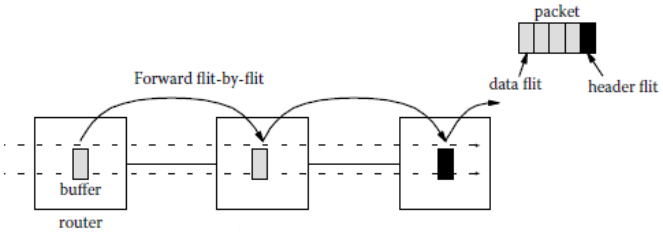


FIGURE 1.10 – Commutation Wormhole (WH) [12]

envoyé flit par flit au routeur suivant sans qu'il soit nécessaire que ce dernier puisse mémoriser sa totalité contrairement à la politique SAF (voir Figure 1.10). Par conséquent, l'espace tampon requis dans un routeur donné est réduit, ainsi que sa latence d'acheminement. En revanche, si le premier flit (appelé flit d'entête) est bloqué dans un routeur, les flits qui le suivent (appelés flits de données) appartenant au même paquet bloquent à leur tour des liens intermédiaires. Cette situation est appelée blocage en tête de ligne (HOL) qui représente un inconvénient majeur pour ce type de commutation.

3. Virtual Cut Through (VCT) : est une technique intermédiaire réduisant à la fois la latence du SAF (les flits du paquet peuvent être expédiés au prochain routeur sans que l'intégralité du paquet soit présente dans le routeur courant), ainsi que les interblocages du wormhole. Comme dans la technique SAF, la taille du tampon d'un routeur donné est supérieure ou égale à la taille du paquet. En cas de blocage, seul le nœud courant est bloqué et les liens intermédiaires ne sont pas affectés.

1.3.2.2 Fonction de routage

Le routage consiste à transférer les données de la source à la destination avec une stratégie clairement définie [79]. Dans la littérature, les chercheurs ont classé les algorithmes de routage selon différents critères en plusieurs catégories à savoir [80] : routage statique ou dynamique, routage distribué ou source et routage minimal ou non minimal.

- Routage statique ou dynamique : les décisions de routage dans un routeur NoC peuvent être statiques (également appelées déterministes) ou dynamiques (également appelées adaptatives). Dans le routage statique, des chemins fixes sont utilisés pour transférer des données entre une source et une destination particulière. Ce schéma de routage ne tient pas compte de l'état actuel du réseau (la charge sur les routeurs et les liens) au moment de prendre des décisions de routage. L'avantage du routage statique est sa simplicité de mise en œuvre ainsi que sa garantie de livraison des paquets de données dans l'ordre. XY [81] est un exemple d'algorithme de routage statique. Dans le routage dynamique, les décisions de routage sont prises en fonction de l'état actuel du réseau, en tenant compte de la disponibilité et de la charge sur les liens. Il est donc possible que le chemin entre la source et la destination change au fil du temps, à mesure que les conditions du trafic et les exigences de l'application changent. Le routage dynamique est capable de mieux répartir le trafic en utilisant des chemins alternatifs lorsque certaines directions sont encombrées. Cependant, ce comportement adaptatif s'accompagne d'un surcroit de ressources nécessaires à la surveillance de l'état du réseau et le choix des chemins de routage [80][79].
- Routage distribué ou source : les schémas de routage statique et dynamique peuvent être classés en fonction de l'endroit où les informations de routage sont stockées et de l'endroit où les décisions de routage sont prises. Dans le routage distribué, chaque paquet porte l'adresse de destination et les décisions de routage sont prises dans chaque routeur (localement) en utilisant les tables de routage. Dans le routage source, seul l'expéditeur fournit le chemin par lequel les données vont circuler. En effet, les tables de routage pré-calculées sont stockées au niveau d'un nœud NI (ou PE) et cette information est ajoutée à l'en-tête du paquet [80][79].
- Routage minimal ou non minimal : une autre classification des schémas de routage

selon le choix du chemin est donné. Un routage est dit minimal si la longueur du chemin de la source à la destination est la plus courte possible entre les deux nœuds. En routage minimal, la source ne commence pas à envoyer un paquet si un chemin minimal n'est pas disponible. Cependant, un schéma de routage non minimal n'a pas de telles contraintes et peut utiliser des chemins plus longs si un chemin minimal n'est pas disponible.

1.3.2.3 Politiques d'arbitrage

Une fois que le saut suivant est déterminé par une technique de routage, il arrive que deux paquets sollicitent le même port de sortie d'un routeur donné, et cela en même temps. Dans ce cas on parle de contention ou conflit. Afin de remédier à cette situation et de désigner qui sortira en premier, plusieurs politiques d'arbitrage ont été proposées, dont les principales sont [82] : le Time-Division Multiple Access (TDMA), la réservation de Time-slot, la priorité fixe et le tourniquet ou Round-robin. Ces deux dernières techniques seront décrites dans ce qui suit :

- Arbitrage basé priorité fixe : dans la technique par priorité fixe, chaque paquet possède une priorité fixe qui lui a été attribuée par l'émetteur avant son entrée dans le réseau. Cette priorité est utilisée tout au long du cheminement du paquet entre sa source et sa destination [82].
- Tourniquet ou Round Robin (RR) : l'arbitrage de type tourniquet ou round-robin représente la politique d'arbitrage la plus impartiale. En effet, à chaque cycle, elle donne l'accès à un paquet provenant d'une entrée différente [82].

1.3.2.4 Protocoles de contrôle de flux

Le protocole de contrôle de flux détermine quand les paquets traversent le réseau et se rendent de la source à la destination. Il prend également en charge le schéma de contrôle d'erreurs de bout en bout et au niveau du routeur en présence d'une erreur de transmission [78]. Il existe plusieurs protocoles de contrôle de flux dont les principaux sont : le protocole basé crédits d'émission (credit-based), le protocole ON/OFF, et le protocole ACK/NACK [83][77].

- Le protocole basé crédits d'émission (credit-based) : dans le contrôle de flux basé sur le crédit (CB), les nœuds émetteurs en amont (upstream) disposent d'informations sur le nombre d'emplacements vides dans les tampons des nœuds récepteurs en aval (downstream). Nous appelons cette information CN (Credit Number).Chaque fois qu'un nœud émetteur envoie un flit aux tampons récepteurs, le nombre est décrémenté de 1. Lorsque les tampons récepteurs envoient quelques flits à d'autres nœuds, ils envoient également un signal de contrôle de crédit aux routeurs émetteurs, et lorsque le routeur émetteur reçoit le signal, le CN associé au chemin est incrémenté de manière appropriée [77].
- Le protocole ON/OFF : le contrôle de flux ON/OFF est une technique qui réduit la quantité de signalisation. Dans cette technique l'état de l'émetteur en amont (upstream) a deux états : autorisé à envoyer ("ON") ou non ("OFF"). Un signal est envoyé à l'émetteur que si il est nécessaire de changer cet état. Un signal d'arrêt ("OFF") est envoyé lorsque le bit de commande est activé et le nombre de tampons libres passe en dessous du seuil F_{OFF}. Si le bit de commande est désactivé et le nombres de tampons libres dépasse le seuil F_{ON}, le signal "ON" est envoyé [83].
- Le protocole ACK/NACK : avec le contrôle de flux ACK/NACK, aucun état n'est conservé dans le nœud émetteur pour indiquer la disponibilité de la mémoire tampon du nœud récepteur. Le nœud émetteur envoie de manière optimiste des flits dès qu'ils sont disponibles. Si le nœud récepteur dispose d'une mémoire tampon, il accepte le flit et envoie un accusé de réception (ACK) au nœud émetteur. Si aucune mémoire n'est disponible à l'arrivée du flit, le nœud récepteur abandonne le flit et envoie un accusé de réception négatif (NACK). Le nœud émetteur conserve une copie de chaque flit jusqu'à ce qu'il reçoit un ACK. S'il reçoit un NACK, il retransmet le flit [83].

1.4 Les défis des MPSoCs : le problème de placement (Mapping)

Comme cité précédemment, les MPSoCs en particulier les MPSoCs hétérogènes sont apparus comme une solution incontournable pour réponde aux demandes incessantes des

applications embarquées en termes de performance. Selon l'ITRS [36], les plateformes pourraient contenir plus de cinq mille PEs (ou cores) d'ici 2026. L'une des questions clés est de savoir comment exploiter efficacement ce type de plate-forme ?. Plusieurs défis de recherche ont été présentés d'une manière détaillée dans [84][59][35] et brièvement décrits dans ce qui suit à savoir :

1. Parallélisation de l'application, appelé aussi partitionnement : il est définit comme étant le processus d'analyse d'une application et de sa fragmentation en blocs (tâches) en vue d'extraction du parallélisme. Selon la quantité de calculs à effectuer au niveau de chaque bloc, trois types de grains sont distingués : le grain fin, le grain moyen et le gros grain. Le parallélisme efficace dans les MPSoCs, en particulier le choix de granularité, reste une question ouverte.

2. Le modèle de programmation : d'une manière générale, le modèle de programmation sert à masquer les détails matériels. On distingue deux types de modèles : le modèle séquentiel et le modèle parallèle. Ce dernier présente un intérêt particulier pour la programmation des MPSoCs. Il existe de nombreux modèles de programmation parallèle, dont certains sont construits sur des langages séquentiels traditionnels tels que C ou C^{++} au moyen de directives de compilation, de bibliothèques ou d'extensions de langage. Ces modèles de programmation ont leurs racines dans la communauté du calcul haute performance (HPC), mais ils ont gagné du terrain dans le domaine embarqué. On cite à titre d'exemples : OpenMP [85], MPI [86], etc. En plus de ces modèles cités ci-dessus, un autre modèle appelé modèle flot de données semble être un choix prometteur pour décrire les applications de traitement de signal. Ce dernier est basé sur une représentation graphique des applications (eg. KPN, SDF, DDF, etc.). Malgré le nombre important de modèles de programmation parallèle qui ont été proposés depuis des décennies, des recherches sur de nouvelles façons d'exploiter les MPSoCs et de décrire efficacement les applications parallèles restent un axe de recherche ouvert, et cela est dû encore une fois, à la complexité toujours croissante du matériel, à l'hétérogénéité du système et aux nouvelles applications qui deviennent de plus en plus complexes [26].

3. Mapping : est une phase d'allocation spatiale des processeurs d'un MPSoC pour les

blocs (tâches) d'une application donnée tout en optimisant un ou plusieurs objectifs. Ce problème peut avoir certaines contraintes (applicatives ou architecturales). Dans les premiers systèmes, cette phase est effectuée manuellement, ce qui est vite rendu impossible aujourd'hui vu la complexité conjointe des applications et des plateformes. À titre d'exemple, supposons une application constituée de k tâches et une plateforme de l processeurs. Le nombre de solutions de mapping à explorer est de l^k, s'ajoute à cela la nature des objectifs à optimiser qui sont souvent conflictuels. Selon le moment ou la décision du mapping est prise, deux types de Mapping se distinguent : (1) Mapping dynamique et (2) Mapping statique. Ce dernier fait l'objet de nos travaux de recherches au cours de cette thèse et sera exposé d'une manière détaillée dans les sections qui suivent.

4. Scheduling : en plus de l'allocation spatiale, cette phase traite la problématique de l'allocation temporelle. En d'autre termes, l'ordonnancement ou scheduling consiste à trouver une séquence temporelle appropriée pour les tâches parallèles. Comme le mapping, ce problème est similaire à un problème d'assignation quadratique qui est un problème NP-difficile [41]. Il est à noter que les problèmes du mapping et du scheduling pourraient être traités comme des problèmes indépendants distincts ou comme un seul problème intégré [87][88].

5. Génération de code : après les phases du parallélisme, du mapping et du scheduling, la génération du code pour les MPSoCs en particulier le MPSoCs hétérogènes interconnectés avec un NoC est une tâche non triviale dans le flot de conception de ces derniers et constitue un autre défi à surmonter pour les chercheurs.

1.4.1 Problème de placement (Mapping)

Le problème de Mapping consiste à trouver un emplacement pour les tâches d'une application (graphe de tâches) sur les ressources d'une plateforme donnée (pour notre cas, il s'agit d'un MPSoC hétérogène interconnecté avec un réseau sur puce) afin de répondre à un ensemble d'exigences telles que : l'économie d'énergie, la réduction de congestion, etc. Les décisions du Mapping peuvent avoir une influence considérable sur la performance du système final. Selon le moment ou cette décision est prise, le Mapping peut être dynamique

ou statique [42]. Le Mapping dynamique (appelé dynamic ou on-line mapping) consiste à placer les tâches au moment de l'exécution de l'application. Par conséquent, ce type de mapping est adapté aux scénarios dynamiques [89]. En revanche, dans le Mapping statique (appelé static ou off-line mapping), les décisions de placement des tâches sont prises avant l'exécution de l'application (i.e. lors de la conception). Donc, la qualité des solutions de ce type de mapping avec une vue globale du système peut être supérieure par rapport au mapping dynamique ayant juste une vue locale [42][90]. Le Mapping statique est souvent utilisé pour les applications critiques ou la dégradation de la qualité de service (QoS) est inacceptable [91].

Dans cette thèse, nous avons porté notre intérêt sur le problème de placement statique (off-line Mapping). Les entrées de ce dernier sont :

- Le modèle d'application : l'application est spécifiée comme étant un ensemble de blocs communicants, définit généralement à l'aide des graphes (ex. TG, CTG, ATG, etc.).
- Le modèle d'architecture : est une spécification abstraite de la plateforme cible (ex. le nombre de processeurs et leurs types, les caractéristiques des éléments de stockage, les caractéristiques du système d'interconnexion, etc.)
- Un ensemble de fonctions objectives à optimiser (ex. temps d'exécution, énergie, communication, etc.). Ces fonctions sont définies en utilisant un modèle analytique ou par simulation.
  - Modèle analytique : consiste à trouver des équations mathématiques utilisées pour évaluer une solution d'un mapping donné. Ce modèle a l'avantage d'être moins couteux en termes de temps d'exécution et cela au détriment du niveau de précision. Plusieurs modèles analytiques pour définir des fonctions objectives ont été utilisés dans le cadre de cette thèse présentés en chapitre 3.
  - Modèle par simulation : à l'opposé du modèle abstrait décrit ci-dessus (modèle analytique), l'avantage du modèle par simulation est sa capacité de représenter un système donné dans les détails, et ainsi capturer ses effets dynamiques (i.e. les contentions). Un modèle de simulation à évènements discrets a été proposé dans le cadre de cette thèse présenté en chapitre 4. Ce modèle est utilisé pour évaluer

une solution de placement donnée en présence de contentions en particulier les contentions au niveau NoC.

- Un ensemble de contraintes à considérer (ex. le deadline, pré-affectation de certaines tâches, etc.)

Le problème du Mapping étant connu comme un problème NP-difficile [41]. Il est d'autant plus difficile lorsqu'il y a plusieurs facteurs à prendre en compte qui sont souvent contradictoires (conflictuels), comme : la communication vs l'équilibrage de charge, la latence vs l'énergie de consommation, etc. Des méthodes de résolution intelligentes sont requises afin de résoudre cette problématique. C'est dans ce contexte que nous nous sommes positionnés. Ainsi, nous avons proposé une multitude d'approches de résolution décrites dans les chapitres qui suivent.

1.5 Conclusion

Dans ce chapitre, nous avons présenté les deux paradigmes le NoC et le MPSoC. Nous avons exploré les défis de recherche de ce dernier dont le problème de placement en particulier le placement statique. En général, ce problème est NP-difficile et multiobjectif et nécessite d'utiliser des métaheuristiques pour sa résolution. Dans le chapitre qui suit, nous allons présenter les principes de base de l'optimisation multiobjectif suivi d'un état de l'art des différents algorithmes utilisés dans le contexte du Mapping statique.

Chapitre 2

Optimisation multiobjectif et le problème du Mapping : Etat de l'art

2.1 Introduction

De multiples objectifs, souvent contradictoires, apparaissent naturellement dans de nombreux problèmes de décision [92][13][93]. C'est notamment le cas du problème du Mapping dans un MPSoC, où plusieurs critères, souvent contradictoires, doivent être considérés simultanément lors de placement de tâches sur une plateforme cible. En effet, optimiser un tel problème relève de l'optimisation combinatoire multiobjectif, où la solution de placement cherchée n'est pas une solution unique mais un ensemble de solutions compromis communément appelé le font de Pareto.

L'optimisation multiobjectif trouve ses racines au 19 ème siècle, dans l'œuvre d'Edgeworth et de Pareto, utilisé en économie et en science de la gestion pendant plusieurs décennies, puis progressivement en sciences de l'ingénierie. De nos jours, l'optimisation multiobjectif est un domaine important de la science et de l'ingénierie [13].

Dans la première partie de ce chapitre, nous introduisons les concepts fondamentaux liés à l'optimisation multiobjectif y compris la notion de dominance, la surface de compromis, les principales méthodes de résolution ainsi que les métriques de performance des fronts de Pareto résultants. Dans la seconde partie, nous donnons un état de l'art des approches de résolution du problème du Mapping statique issues de la littérature.

2.2 Optimisation multiobjectif

Dans la vie de tous les jours, on est souvent confronté à résoudre des problèmes à plus d'un seul objectif (la plupart des cas conflictuels). L'exemple typique est lors de l'achat d'un véhicule d'occasion, on souhaite acheter un véhicule peu cher avec peu de kilomètres et bien d'autres exemples voir [13][94]. Lorsqu'on veut réaliser ces objectifs à la fois, on parle d'optimisation multiobjectif.

2.2.1 Concepts et définitions

2.2.1.1 Définition d'un problème multiobjectif (multicritères)

Un problème d'optimisation multiobjectif (appelé PMO) implique une optimisation simultanée de multiples objectifs qui sont souvent contradictoires. Lors de la résolution de tels problèmes, il n'est pas possible de trouver une solution unique qui optimise tous les objectifs simultanément. Il existe plutôt un ensemble de solutions définies comme des solutions optimales de Pareto. Cet ensemble de solutions représente les solutions de compromis entre les différents objectifs contradictoires. Le but principal de la résolution d'un problème multiobjectif est d'obtenir l'ensemble optimal de Pareto et, par conséquent, le front de Pareto. Plus formellement, un problème d'optimisation multiobjectif (cas de minimisation de tous les objectifs) peut être formulé comme suit [13][95] :

$$\text{"}min\text{"}F(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \mid x \in S \tag{2.1}$$

où n ($n \geq 2$) est le nombre d'objectifs, $x = (x_1, \ldots, x_k)$ est le vecteur représentant les variables de décisions, et S représente l'ensemble des solutions possibles associées aux contraintes d'égalité ou d'inégalité et aux limites explicites. $F(x) = (f_1(x), f_2(x), \ldots, f_n(x))$ est le vecteur des objectifs à optimiser.

L'espace de recherche S représente l'espace de décision ou l'espace de paramètres du PMO. L'espace auquel le vecteur objectif appartient s'appelle l'espace objectif. Le vecteur F peut être défini comme une fonction de coût à partir de l'espace de décision qui évalue la qualité de chaque solution $x = (x_1, \ldots, x_k)$ en assignant un vecteur objectif $y = (y_1, \ldots, y_n)$, qui représente la qualité de la solution (ou l'aptitude) (voir Figure 2.1). Dans le domaine de

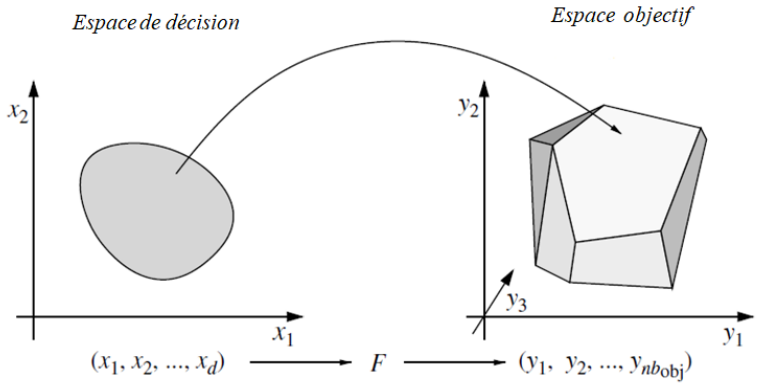


FIGURE 2.1 – Espace de décision et espace objectif dans un PMO [13].

l'optimisation multiobjectif, le décideur l'utilise pour travailler en termes d'évaluation d'une solution sur chaque critère, et se place naturellement dans l'espace objectif. L'ensemble $Y = F(S)$ représente les points réalisables dans l'espace objectif.

Les objectifs de la formule 2.1 sont souvent contradictoires. Comme précédemment cité, la résolution d'un PMO retourne une multitude de solutions. Seules les solutions optimales au sens Pareto sont intéressantes. Avant de définir l'optimalité de Pareto, nous allons d'abord introduire la notion de dominance.

2.2.1.2 La notion de dominance [14]

Définition 1

*On dit que le vecteur $\vec{x_1}$ **domine** le vecteur $\vec{x_2}$ au sens de Pareto ($\vec{x_1} \prec \vec{x_2}$) ssi :*

- *$\vec{x_1}$ est au moins aussi bon que $\vec{x_2}$ dans tous les objectifs, et,*
- *$\vec{x_1}$ est strictement meilleur que $\vec{x_2}$ dans au moins un objectif.*

Cette notion de dominance de Pareto est illustrée par la Figure 2.2 (pour un cas de minimisation à 2 objectifs), par exemple :

- Le point A domine le point B
- Le point A est dominé par la point C
- Le point A est incomparable (équivalent) aux points E et F.

Les solutions qui dominent les autres mais ne se dominent pas entre elles sont appelées solutions optimales au sens de Pareto (ou solutions non dominées) [14].

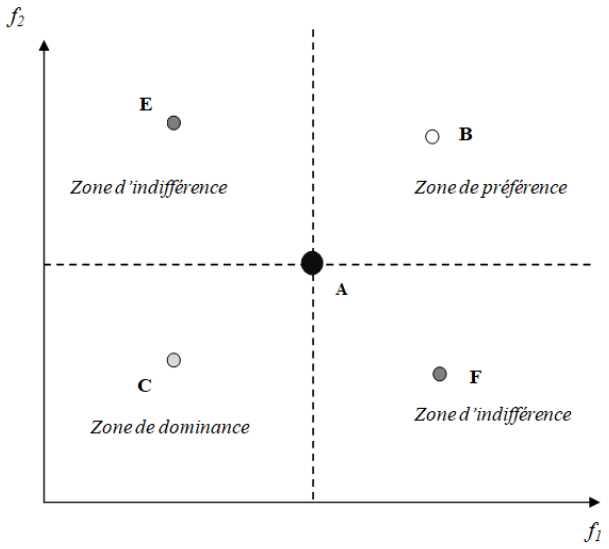


FIGURE 2.2 – Exemple illustrant la notion de dominance au sens de Pareto [14].

2.2.1.3 Ensemble optimal au sens Pareto (Pareto Optimal Set)[13][96]

Définition 2

Pour un PMO (F, S) donné, l'ensemble optimal de Pareto est défini comme suit :

$$P^* = \{x \in S \mid \nexists x' \in S, F(x') \prec F(x)\}$$

2.2.1.4 Le front de Pareto (Pareto front)[13][96]

Définition 3

Pour un PMO (F, S) donné et son ensemble optimal P^*, le front Pareto est défini commme suit :

$$PF^* = \{F(x), x \in P^*\}$$

Le front de Pareto est l'image de l'ensemble optimal de Pareto dans l'espace objectif. L'obtention du front de Pareto d'un PMO est l'objectif principal de l'optimisation multiobjectif. Cependant, étant donné qu'un front de Pareto peut contenir un grand nombre de points, une bonne approximation du front de Pareto peut contenir un nombre limité de solutions de Pareto qui doivent être aussi proches que possible du front de Pareto exact (convergence), et qui doivent être uniformément réparties sur le front de Pareto (diversification) (voir Figure 2.3). Sinon, l'approximation obtenue du front de Pareto ne

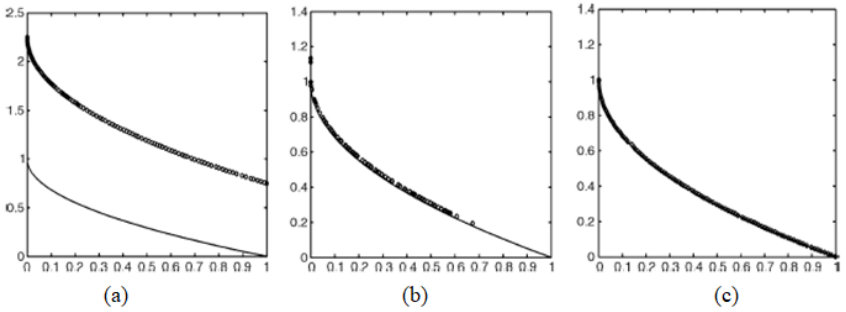


FIGURE 2.3 – Exemples de fronts de Pareto : (a) mauvaise convergence et bonne diversité, (b) bonne convergence et mauvaise diversité, (c) bonne convergence et bonne diversité [13].

serait pas très utile au décideur qui devrait avoir une information complète sur le front de Pareto [13].

La détermination de l'ensemble de solutions de Pareto n'est que la première phase dans la résolution d'un PMO, qui nécessite dans un deuxième temps le choix d'une solution à retenir à partir de cet ensemble, et cela suivant les préférences choisies par le décideur [15]. Donc, l'une des questions fondamentales dans la résolution des PMOs est liée à l'interaction entre le résolveur du problème (méthode d'optimisation) et le décideur. Le rôle du décideur est de spécifier quelques informations supplémentaires pour sélectionner sa solution préférée. Cette interaction peut prendre l'une des trois formes suivantes [14][13][15][97][98][99] :

- A priori : dans cette approche, le décideur indique ses préférences avant le processus d'optimisation ($d\acute{e}cider \Rightarrow rechercher$)[96], l'exemple de cette approche est la méthode d'agrégation pondérée [14].
- Interactive : dans ce cas, il y a une coopération progressive entre le décideur et le solveur ($d\acute{e}cider \Leftrightarrow rechercher$)[96]. A partir des connaissances acquises pendant la résolution du problème, le décideur définit ses préférences. ces dernières sont prises en compte par le solveur dans la résolution du problème. Ce processus est réitéré pendant plusieurs étapes.
- A posteriori : dans l'approche a posteriori, le processus de recherche détermine un ensemble de solutions de Pareto. Cet ensemble aide le décideur à avoir une connaissance complète du front de Pareto. Ensuite, le décideur choisit une solution parmi l'ensemble des solutions fournies par le solveur ($rechercher \Rightarrow d\acute{e}cider$)[96].

En d'autres termes, cette approche comporte deux phases :

1. Rechercher des solutions de meilleurs compromis (phase 1).
2. Choix de la solution à retenir par le décideur final (phase 2).

Dans le cadre de notre travail, nous nous sommes intéressés à cette dernière approche et plus particulièrement à la première phase (phase 1), en d'autres termes à la phase d'optimisation.

2.2.2 Approches monoobjectif vs multiobjectif

Comparé aux approches mono-objectif, trois principaux composants sont à considérer dans les méthodes multiobjectif [13] qui sont :

- Affectation de la qualité d'une solution : le rôle principal de cette procédure est de guider l'algorithme de recherche vers des solutions optimales de Pareto pour une meilleure convergence. Quatre types d'affectation de la qualité d'une solution se distinguent : les approches scalaires, les approches basées sur un critère, les approches basées sur la dominance et les approches basées sur les indicateurs de performance. Ces approches sont expliquées d'une manière détaillée dans le livre de El-Ghazali Talbi [13] et dans d'autres références telles que [15][100][17].
- Préservation de la diversité : l'accent est mis sur la génération d'un ensemble diversifié de solutions de Pareto. Plusieurs méthodes de maintien de diversité sont données dans [15].
- L'élitisme : d'une manière générale, l'élitisme consiste à archiver les "meilleures" solutions générées lors de la recherche (par exemple, les solutions optimales de Pareto). Une population secondaire, appelée archive, est utilisée pour stocker ces solutions de haute qualité. Tout d'abord, l'élitisme a été utilisé pour prévenir la perte des solutions optimales de Pareto obtenues. Dans cette stratégie d'élitisme dite passive, les archives sont considérées comme une population secondaire distincte qui n'a aucun impact sur le processus de recherche (Figure 2.4 (a)). L'élitisme garantira seulement qu'un algorithme a des performances monotones non dégradantes en termes de front de Pareto approximatif. Ensuite, l'élitisme a été utilisé dans le processus de recherche (élitisme actif), c'est-à-dire que les solutions archivées sont

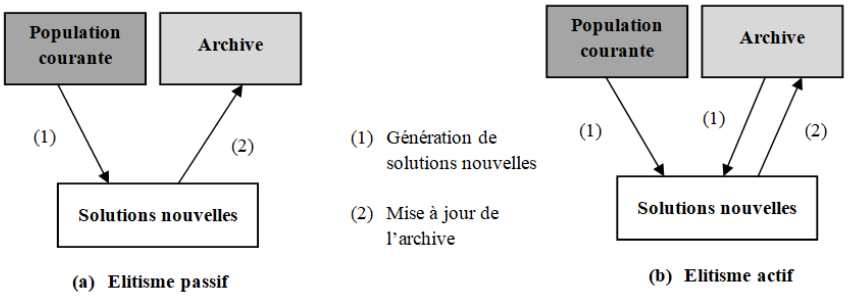


FIGURE 2.4 – L'élitisme dans le multiobjectif [13]

.

utilisées pour générer de nouvelles solutions (Figure 2.4 (b)). L'élitisme actif permet d'atteindre une convergence plus rapide et plus robuste vers le front de Pareto pour une meilleure approximation du front de Pareto.

2.2.3 Résolution d'un problème d'optimisation multiobjectif

Comme pour l'optimisation mono-objectif, les méthodes de résolution d'un problème multiobjectif se distinguent en deux classes principales : les méthodes exactes et les méthodes approchées (heuristiques) [101][13][102][103] (voir Figure 2.5).

2.2.3.1 Méthodes exactes

Les méthodes de recherche exactes garantissent de trouver d'une manière exacte l'ensemble de solutions de Pareto, Cependant, ces méthodes ne sont efficaces que pour des problèmes de petites voir de moyenne tailles et bi-objectif. Pour des problèmes de grande taille, leur durée d'exécution devient prohibitive [104]. Dans le cadre de cette thèse, nous avons implémenté le branch and bound multiobjectif appelé MBB [51] (décrit ci-dessous), et cela dans le but de valider l'efficacité des métaheuristiques (décrites juste après) pour de petites et moyennes instances de notre problème de placement (Mapping).

- MBB : Multiobjective Branch & Bound (MBB) a été proposé par Carlos et al. en 2016, est une version multiobjective du Branch & Bound basée sur la dominance de Pareto [14]. Branch & Bound (appelé méthode d'énumération implicite) est une méthode arborescente qui pratique une énumération intelligente de l'espace de solutions. Il s'agit en quelque sorte d'énumérations complètes améliorées. Elle

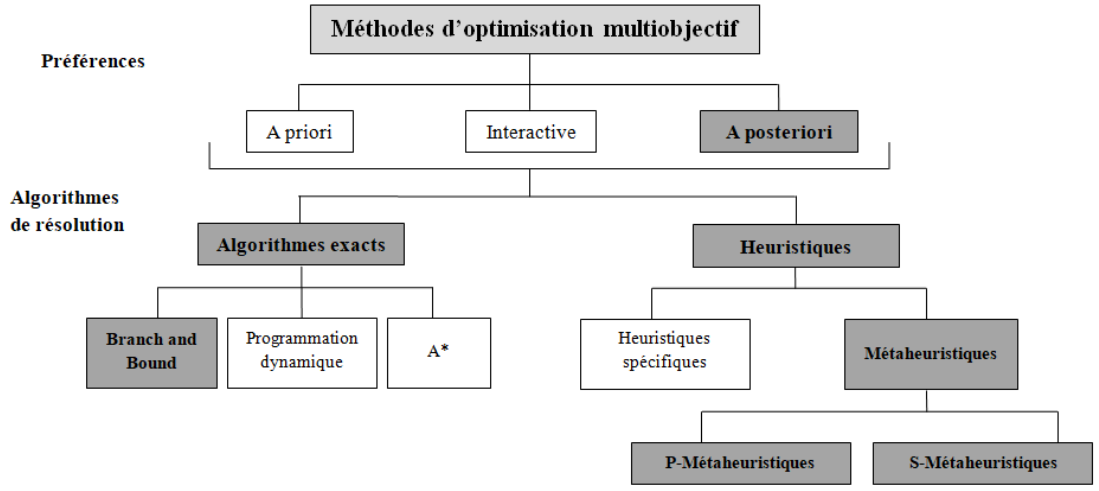


FIGURE 2.5 – Classification des méthodes de résolution multiobjectif [13][15].

partage l'espace des solutions en sous-ensembles de plus en plus petits, la plupart étant éliminés par des calculs de bornes avant d'être construits explicitement. Cette méthode de recherche comporte trois composantes principales qui sont [104] :

- Une règle de séparation (de partitionnement) des solutions (branching rule) ;
- Une fonction de bornage (bounding function) ;
- Une stratégie d'exploration (search strategy).

2.2.3.2 Méthodes heuristiques

Contrairement aux méthodes exactes, les méthodes heuristiques ne garantissent pas de trouver d'une manière exacte l'ensemble de solutions de Pareto mais une approximation (aussi bonne que possible) de ce dernier, elles ont l'avantage de pouvoir résoudre des problèmes de grande taille en un temps raisonnable. Ces méthodes peuvent être développées pour résoudre un type de problèmes particulier, dans ce cas on parle de Méthodes heuristiques spécifiques, ou bien elles peuvent s'adapter à n'importe quel type de problème, et on parle de métaheuristiques. Les métaheuristiques ont prouvé leur efficacité dans la résolution de plusieurs problèmes multiobjectif [92][102]. Vu leur succès dans la résolution des PMOs, nous avons porté notre intérêt pour cette classe de méthodes approchées (métaheuristiques) pour la résolution de notre problème de placement.

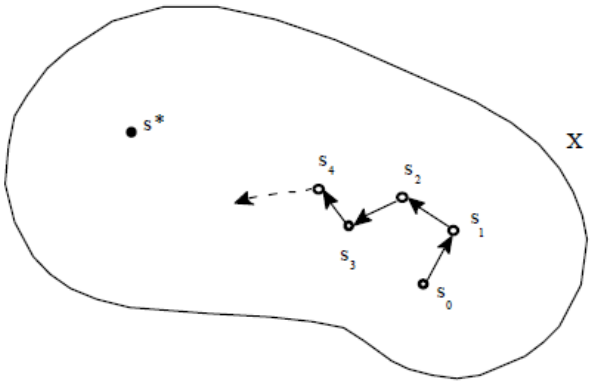


FIGURE 2.6 – Exploration de X par une approche locale [16].

Selon leur principe de recherche, les métaheuristiques se regroupent en deux classes : les métaheuristiques basées sur une solution unique (appelées S-métaheuristiques) et celles basées sur une population de solutions (appelées P-métaheuristiques) [13].

2.2.3.3 Les métaheuristiques à base de solution unique (S-metaheuristiques)

Les métaheuristiques à base de solution unique appelées aussi méthodes de recherches locales sont des méthodes itératives consistant à traiter une seule solution en essayant de l'optimiser (la remplacer par une autre issue de son voisinage [98]). Un exemple illustrant ce principe est donné en Figure 2.6, où le passage d'une solution s à une solution s' dans l'espace X se fait sur la base d'un ensemble de modifications élémentaires (en spécifiant une fonction de voisinage appelée $N(s)$). Ce processus d'exploration est réitéré jusqu'à ce qu'un ou plusieurs critères d'arrêt soient satisfaits. Les passages successifs d'une solution à une solution voisine définissent un chemin (appelé trajectoire)[16]. Les algorithmes les plus représentatifs de cette catégorie sont le récuit simulé (SA) et la recherche tabou (TS). Plusieurs variantes multiobjectives du recuit simulé [105][106] et de recherche tabou [107][108] ont été développées. Nous allons décrire les algorithmes utilisés dans le cadre de cette thèse, à savoir le recuit simulé multiobjectif (MOSA [49] et AMOSA [6]), la recherche tabou multiobjectives (MOTS [50]) et PAES [109].

- MOSA [49] : Multiobjective Simulated Annealing (MOSA) introduit par Nam et Park en 2000. MOSA reprend les mêmes principes de SA (Simulated Annealing),

mais ajoute la notion de dominance lors de la sélection des solutions. En plus de cela, MOSA construit une liste contenant les solutions non-dominées trouvées durant de la recherche (élitisme passif voir section 2.2.2).

- AMOSA [6] : Archived Multiobjective Simulated Annealing (AMOSA) proposé par bandyopadhyay et al. en 2008. Comparant à MOSA, AMOSA intègre une archive afin de stocker les solutions non-dominées trouvées lors de la recherche. Ces solutions archivées servent aussi à déterminer la probabilité d'acceptation d'une nouvelle solution (élitisme actif voir section 2.2.2). La taille de l'archive est gardée bornée car seul un nombre limité de solutions de Pareto optimales plus ou moins régulièrement distribuées est nécessaire. AMOSA utilise le concept de degré de domination (amount of domination) dans le calcul de la probabilité d'acceptation d'une solution dégradante [110]. La description détaillée de l'algorithme est donnée dans [6][110][111] .
- MOTS [50] : MultiObjective Tabu search (MOTS), est une version multiobjective de TS présentée par Jaffrès Runser et al. en 2008. Il est appelé PMOTS, ce qui signifie "Parallel-MultiObjective Tabu Search". L'algorithme exploite K trajectoires de recherche parallèles. Une liste tabou est assignée à chaque chemin de recherche. Cette liste sert à empêcher les cycles et à forcer l'acceptation des solutions dégradantes (solutions dominées), et cela dans le but de diriger la recherche vers de nouvelles régions non visitées encore. Les solutions non-dominées trouvées au cours de la recherche seront sauvegardées dans une liste. Cette dernière contiendra les solutions non dominées finales (le front Pareto optimal).
- PAES [109] : Pareto Archived Evolution Strategy (PAES), conçu et mis en œuvre par Knowles et Corne. PAES consiste en une stratégie d'évolution (1+1)(c.à.d. un parent célibataire qui génère une progéniture unique) en combinaison avec une archive historique qui enregistre les solutions non dominées précédemment trouvées, qui est également utilisée pour prendre des décisions concernant de nouvelles solutions candidates. Une grille adaptative est utilisée comme estimateur de densité dans l'archive. La caractéristique de PAES est qu'il ne fait appel à aucun opérateur de recombinaison (crossover)[112][113][111].

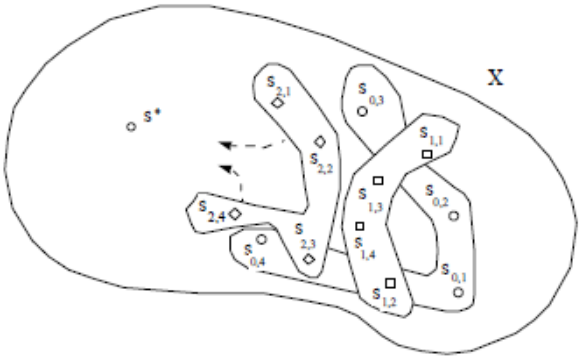


FIGURE 2.7 – Exploration de X par une approche évolutive [16].

2.2.3.4 Les métaheuristiques à base de population de solutions (P-metaheuristiques)

Contrairement aux algorithmes partant d'une solution singulière (S-métaheuristiques), les métaheuristiques à population de solutions (appelées P-métaheuristiques) améliorent, au fur et à mesure des itérations, une population de solutions [17] (voir Figure 2.7). On distingue dans cette catégorie les algorithmes évolutionnaires (EAs), la recherche par dispersion (SS), l'optimisation des essaims de particules (PSO), etc.

1. Les algorithmes évolutionnaires (EC : Evolutionary Computation) : sont une famille d'algorithmes s'inspirant de la théorie d'évolution darwinienne énoncée par Charles Darwin [114]. Le terme Evolutionary Computation englobe une classe assez large de métaheuristiques telles que : les algorithmes génétiques (GAs), les stratégies d'évolution (ES), la programmation évolutive (EP), et la programmation génétique (GP). La Figure 2.8 décrit le squelette d'un algorithme évolutionnaire type commun à la plupart des instances classiques des algorithmes évolutionnaires (EAs) [17][115]. Ces algorithmes ont plusieurs adaptations pour l'optimisation multiobjectif appelés MOEAs (MultiObjective Evolutionary Algorithms). Nous allons décrire succinctement dans ce qui suit les algorithmes évolutionnaires utilisés dans le cadre de notre étude.
   - NSGAII [1] : l'algorithme NSGAII (Non Sorting Genetic Algorithm II) est une version améliorée de NSGA [116], proposée par Deb et al. en 2000. NSGAII est l'un des algorithmes évolutionnaires populaires très utilisé aujourd'hui dans la résolution des problèmes multiobjectif. Les principales caractéristiques de

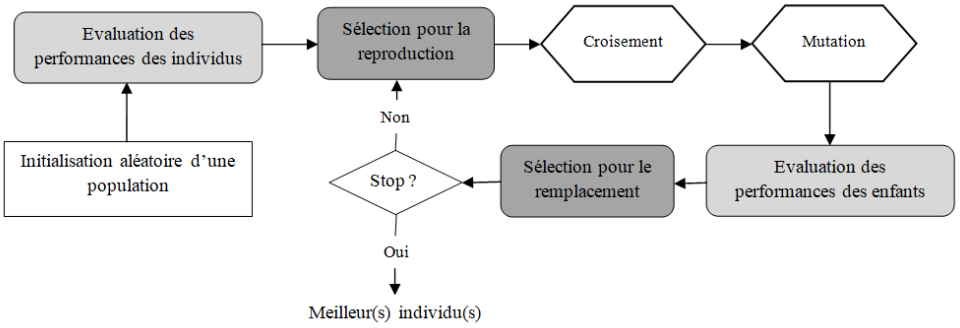


FIGURE 2.8 – Principe d'un algorithme évolutionnaire (EA) [17].

cet algorithme sont décrites comme suit [117] : (1) utilisation de l'élitisme, (2) utilisation d'un mécanisme explicite de préservation de la diversité (crowding distance) et (3) préservation de la convergence en utilisant une procédure de tri selon la notion de dominance (ranking). Pour une description plus détaillée de l'algorithme, se référer à [1][118][117].

- SPEA2 [2] : Strength Pareto Evolutionary Algorithm 2 (SPEA2) est un autre algorithme de référence dans l'optimisation multiobjectif proposé par Zitzler et al. en 2001 comme une amélioration de SPEA. Par rapport à ce dernier, SPEA2 utilise une stratégie d'affectation de fitness à grain fin qui intègre les informations de densité. La taille de l'archive est fixée, c.-à-d, à chaque fois que le nombre d'individus non-dominés est inférieur à la taille de l'archive prédéfinie, l'archive est remplie par des individus dominés. La technique de clustering, qui est invoquée lorsque le front non-dominé dépasse la limite d'archives, a été remplacée par une méthode alternative de troncature qui a des caractéristiques similaires mais ne fait pas perdre les points limites. Finalement, une autre différence à SPEA est que seulement les membres de l'archive qui participent au processus de sélection [119]. Les références [2][120][113] donnent plus de détails à propos de cet algorithme.
- PESAII [3] : Pareto Envelope-Based Selection Algorithm II (PESA-II), proposé par Corne et al. en 2001, est une version révisée de PESA. La propriété de cet algorithme est sa méthode de sélection basée- région (hypercube) au lieu d'une sélection basée-individu. Cette méthode a l'avantage d'offrir des solutions

diversifiées [121].

- IBEA [5] : Indicator-based Evolutionary Algorithm proposé par Zitzler et Künzli en 2004. Sa particularité, est son utilisation d'un indicateur de qualité dans la procédure de sélection. Par conséquent, cet algorithme n'aura pas besoin d'utiliser une technique de préservation de diversité comme un deuxième critère de comparaison [122]. Pour une description détaillée de l'algorithme, se référer à [5][123].
- MOCell [44] : Multiobjective Cellular Genetic Algorithm (MOCell) introduit par Nebro et al. en 2009. Il est basé sur un modèle cellulaire d'AG, où le concept de petit voisinage est utilisé de manière intensive, c'est à dire que les membres de la population peuvent seulement intéragir avec leurs voisins proches lors de la reproduction. Les principales caractéristiques de MOCell peuvent être résumées comme suit : (1) utilise une archive externe pour stocker les solutions non dominées trouvées, (2) sa structure de population, dont les membres sont disposés selon une grille toroïdale bidirectionnelle (d'où le nom cellulaire), et (3) un certain nombre de solutions sont déplacées à partir de l'archive afin de remplacer un certain nombre d'individus de la population sélectionnés au hasard (feedback). Cette suite de références détaille davantage l'algorithme. Le lecteur peut se référer à [44][124][113].
- FastPGA [4] : Fast Pareto Genetic Algorithm (FastPGA) introduit par Eskandari et al. en 2007. Cet algorithme combine les idées proposées dans NSGAII [1] et SPEA2 [2]. FastPGA classe les parents et les enfants en deux ensembles : solutions non-dominées et celles qui sont dominées. Pour les solutions non-dominées, la valeur de fitness est calculée en utilisant la notion de distance d'encombrement (crowding distance) introduite dans NSGAII [1]. Pour les autres solutions, la procédure de calcul de la valeur de fitness ressemble à celle utilisée dans SPEA2 [2]. En outre, FastPGA utilise un mécanisme lui permettant de réguler dynamiquement la taille de la population à chaque itération de l'algorithme génétique [122]. Le processus détaillé de l'algorithme FastPGA pourrait être trouvé dans [4][125].

2. La recherche par dispersion (SS) : la recherche dispersée (appelée en anglais Scatter

Search) [126] est une métaheuristique qui se distingue des algorithmes évolutionnaires classiques par l'utilisation d'une procédure de recherche locale et par la généralisation de l'opérateur de croisement. Cette procédure de croisement peut être appliquée à un ensemble de solutions et pas uniquement à un couple de solutions [127]. Plusieurs adaptations du SS pour le multiobjectif ont été proposées telle que Multiple Criteria Scatter Search [128], MultiObjective Scatter Search [129], AbYSS [46], etc. Nous allons décrire ce dernier (AbYSS) utilisé dans le cadre de la thèse.

- AbYSS [46] : Archive-Based hYbrid Scatter Search (AbYSS) introduit par Nebro et al. en 2008. AbYSS suit la recherche par dispersion (SS), en faisant appel à des opérateurs de croisement et de mutation tirées des algorithmes évolutionnaires. Pour résoudre les problèmes d'optimisation multiobjectif, AbYSS combine l'idée de trois algorithmes évolutifs à savoir NSGAII [1], SPEA2 [2] et PAES [109]. Le processus détaillé de AbYSS pourrait être trouvé dans [46][130][131].

3. l'optimisation des essaims de particules (PSO) : l'optimisation par essaims particulaires (OEP ou PSO en anglais) a été inventée par Kennedy et Ebarhart, en 1995 [132]. Elle est inspirée par le comportement social des animaux tel que les colonies de fourmis, les abeilles et les troupeaux d'oiseaux [133][134]. L'OEP diffère des autres méthodes de calcul évolutionnaire par le fait que les membres de la population appelés particules sont dispersés dans l'espace du problème. Le comportement de l'essaim particulaire doit donc être décrit en se plaçant du point de vue d'une particule. Chacune des particules est dotée : d'une position, d'une vitesse qui lui permet de se déplacer et d'un voisinage. A tout instant t, chaque particule connaît donc : (1) sa meilleure position visitée (xPb), (2) la position du meilleur voisin de l'essaim (xGb) ainsi que (3) la valeur qu'elle donne à la fonction objectif (fitness)[135]. Les équations de mise à jour pour la vitesse $V_i(t+1)$ et la position $X_i(t+1)$ des particules sont données dans [135][136]. Vu leur maturité et leur popularité dans la résolution des problèmes d'optimisation mono-objectif, le PSO a été vite adapté pour résoudre les PMOs en 1999 [137]. Ainsi, plusieurs variantes du MOPSO (Multiobjective Particle Swarm Optimization) ont vu le jour. Dans ce qui suit, nous allons présenter d'une manière brève les deux instances du MOPSO utilisées dans le cadre de notre travail

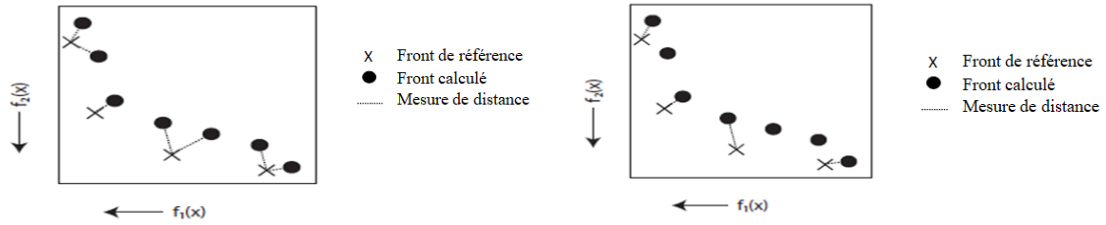
de recherche à savoir OMOPSO [45] et SMPSO [47].

- OMOPSO [45] : Optimized Multiobjective Particle Swarm Optimization (OMOPSO) proposé par Reyes et Coello Coello en 2005. OMOPSO est basé sur la notion de dominance de Pareto avec un facteur d'encombrement (crowding) pour la sélection des leaders. Cette proposition utilise deux archives externes : une pour stocker les leaders actuellement utilisés et l'autre pour stocker les solutions finales. En outre, les auteurs proposent un schéma dans lequel ils subdivisent la population en trois sous-ensembles différents. Un opérateur de mutation différent est appliqué à chaque ensemble. Une description détaillée de l'algorithme est donnée dans les références [45][138][139].
- SMPSO [47] : Speed-constrained Multiobjective PSO (SMPSO) introduit par Nebro et al. en 2009. Le SMPSO est une extension de l'algorithme OMOPSO présenté ci-dessus. Sa principale différence par rapport à OMOPSO est l'incorporation d'un mécanisme de limitation de vitesse et l'introduction d'un opérateur de mutation polynomiale [140]. Les références [47][131] donnent plus de détails sur cet algorithme.

2.2.4 Analyse de performance en optimisation multiobjectif

Contrairement à l'optimisation mono-objectif, où l'évaluation de la performance d'une métaheuristique nécessite principalement d'observer la meilleure valeur obtenue par un algorithme (c'est-à-dire, plus la valeur est faible, mieux c'est, en cas de problèmes de minimisation), l'optimisation multiobjectif repose sur un ensemble de solutions approximatif au front de Pareto du problème. Certains types d'indicateurs doivent être définis afin d'évaluer la qualité de l'ensemble des solutions obtenues. Cette qualité est mesurée en fonction de certains critères qui sont généralement liés aux propriétés de convergence et de diversité [124]. Plusieurs indicateurs de qualité (métriques de performance) ont été proposés dans la littérature [14][141][142]. Dans cette section, nous allons décrire succinctement ceux utilisés dans le cadre de notre travail à savoir : Epsilon [143] et IGD [144].

- Epsilon (EPS) [143] : donne le facteur minimum par lequel l'ensemble d'approximation doit être traduit dans l'espace objectif afin de dominer (faiblement) l'ensemble



(a) la distance générationnelle (GD) est la distance moyenne entre chaque solution du front calculé et la solution la plus proche du front de référence

(b) la distance générationnelle inversée (IGD) est la distance moyenne entre chaque solution du front de référence et la solution la plus proche du front calculé

FIGURE 2.9 – (a) principe de l'indicateur GD, (b) principe de l'indicateur IGD [18]

de référence. Deux version de cet indicateur existent : version multiplicative $EPS_{(\times)}$ et version additive $EPS_{(+)}$. Cette dernière est utilisé dans le cadre de cette thèse. L'indicateur additif epsilon $EPS_{(+)}$ est basé sur un facteur additif définit comme suit :

$$EPS_{(+)}(A) = \max_{r \in R} \min_{a \in A} \max_{i \in \{1,\ldots,d\}} (a_i - r_i) \tag{2.2}$$

Où A est le front calculé (approximation du front de Pareto), R est l'ensemble de référence (le front exact) et d est le nombre d'objectifs. L'indicateur epsilon permet de mettre l'accent sur la convergence des solutions trouvées. Plus cette distance est petite, meilleur est l'algorithme.

- IGD [144] : c'est une métrique qui prend en compte à la fois la diversité ainsi que la convergence appelée mixed metric [145], variante de la Distance générationnelle (GD). Elle mesure les distances entre chaque solution composant le front de Pareto optimal (front de référence) et l'approximation calculée (front calculé) comme montré en Figure 2.9. Elle peut être définie comme suit [146] :

$$IGD = \frac{\sqrt{\sum_{i=0}^{n} d_i^2}}{n} \tag{2.3}$$

Où n est le nombre de solutions dans le front de Pareto optimal (front de référence), et d_i est la distance euclidienne (mesurée en espace objectif) entre chaque point de ce front et le membre d'approximation le plus proche.

Il est à noter que ces deux métriques présentées $EPS_{(+)}$ et IGD exigent une connaissance de l'ensemble de solutions de références.

2.3 Approches pour la résolution du problème du mapping - Etat de l'art

Comme cité précédemment, la phase du mapping consiste en une phase cruciale et déterminante de la qualité du système final. Ce problème de placement (mapping) a été identifié comme l'un des problèmes les plus importants à résoudre pour la mise en œuvre des systèmes embarqués à haute performance [147][148]. Pour cela, plusieurs approches traitant cette problématique ont été proposées [149][147][148][150][151]. Dans cette section, nous allons présenter un état de l'art des travaux de recherche effectués particulièrement dans le contexte du mapping statique.

Hu et Marculescu [152] ont proposé une approche basée sur l'algorithme Branch and Bound pour mapper les IPs sur un NoC (avec une topologie maillée), et cela dans le but de minimiser l'énergie totale de communication avec réservation de la bande passante. Ces mêmes auteurs ont introduit un nouvel algorithme appelé EAS (Energy Aware Scheduling) [153]. Cet algorithme ordonnance statiquement les transactions de communication et les tâches de calcul sur des architectures hétérogènes de réseau sur puce (NoC) sous des contraintes temps réel.

Lie et Kumar [154] ont proposé une approche basée sur un algorithme génétique à deux étape (Two-step Genetic Algorithm) pour placer (mapper) un graphe de tâches (TG) sur une architecture NoC (avec une topologie maillée). Dans la première étape, les tâches sont assignées aux IPs en supposant que les retards de communication sont égaux et constants. Ainsi, les IPs sélectionnées sont basées sur leurs temps d'exécution uniquement. Dans la seconde étape, ils ont utilisé les temps de communications réels pour trouver un placement optimal. Cette approche a pour objectif la minimisation du temps d'exécution total.

Murali et De Micheli [155] ont abordé le problème du mapping en proposant une approche basée sur une heuristique (appelée NMAP). Leur but est de minimiser les délais de communication en exploitant la possibilité de répartir le trafic entre différents chemins sous la contrainte de la bande passante.

Marcon et al. [156] proposent et comparent des algorithmes pour obtenir des placements à faible consommation d'énergie sur des NoCs en se basant sur un modèle pondéré en

communication (CWM). Les modèles de recherche utilisés sont : les méthodes de recherche exhaustives (ES), les méthodes stochastiques (SA et TS), les approches heuristiques (LCF et GI) et des combinaisons pertinentes entre ces approches.

Wang et al. [157] utilisent un algorithm ACO (Ant Colony Optimization) pour placer (mapper) les tâches sur un NoC de sorte que le besoin en bande passante soit minimisé.

Wang et Zhang [158] proposent une nouvelle approche du mapping dans le NoC en utilisant un algorithme génétique adaptatif (AGA). L'approche proposée varie de manière adaptative les probabilités des opérateurs de croisement et de mutation dans l'algorithme génétique, dans le but de réduire le coût global des communications du NoC.

Wang et al. [159] ont proposé un algorithme adaptatif mimétique (AMA) qui combine un algorithme génétique adaptatif (AGA) et un algorithme de recherche locale pour résoudre le problème du mapping. Leur objectif est d'optimiser le coût des communications.

Tous les travaux cités ci-dessus prennent en compte une seule métrique, en d'autres termes, optimisent une seule fonction de coût telle que l'énergie [152][153][156], le temps d'exécution [154], la bande passante [157], la communication [155][158][159]. L'optimisation d'un seul objectif peut entrainer la détérioration d'autres objectifs (par exemple : communication vs équilibrage de charge, énergie de consommation vs temps d'exécution, etc.). Le problème de placement (mapping) doit être résolu dans un environnement multiobjectif. Plusieurs auteurs ont proposé d'utiliser les algorithmes d'optimisation multiobjectif pour résoudre le problème du mapping dans un NoC ou MPSoC, dont les principaux sont décrits dans ce qui suit.

Ascia et al. [160] ont proposé l'utilisation de SPEA2 [2] ainsi que deux autres approches qu'ils ont adaptées pour traiter le multicritères à savoir PBBB (Pareto-based Branch-and-bound) et PBNMAP (Pareto-based NMAP) pour placer les IPs sur les tiles d'un NoC. Leur but est d'optimiser les performances et la consommation énergétique par le bias d'une simulation événementielle (modèle dynamique).

Zhou et al. [161] ont suggéré une approche d'exploration multiobjectif, traitant le problème du mapping comme un problème d'optimisation à deux objectifs contradictoires tentant de minimiser le nombre moyen de sauts et atteindre un équilibre thermique. Ils ont

utilisé l'algorithme évolutif multiobjectif NSGA [116].

Erbas et al. [162] ont comparé entre des métaheuristiques : NSGAII [1] et SPEA2 [2]. L'objectif de leur travail était d'optimiser le temps de traitement, la consommation d'énergie et le coût de l'architecture lors de mapping d'applications sur un MPSoC hétérogène à base de bus partagé.

Jena et Sharma [163] ont abordé le problème du mapping dans un NoC (topologie maillée) en deux étapes systématiques utilisant NSGAII [1]. Leur approche vise à optimiser la consommation d'énergie (énergie de calcul et de communication) ainsi que les besoins en bande passante.

Tornero et al. [164] présentent une approche basée sur un algorithme génétique multiobjectif (MOGA), Leur but est d'optimiser le délai moyen ainsi que la robustesse du routage.

Nedjah et al. [165] proposent une solution qui s'appuie sur des algorithmes évolutifs multiobjectif NSGAII [1] et MicroGA [166]. Les auteurs déterminent d'abord les tâches exécutées par chaque IP, puis leur placement sur les tiles d'un NoC (avec une topologie maillée). Leur optimisation est guidée par la minimisation de la surface matérielle requise, le temps d'exécution et la consommation d'énergie.

Wu et al. [167] proposent un nouvel algorithme de placement appelé GA-MMAS basé sur l'algorithme génétique (GA) et le MAX-MIN Ant System Algorithm (MMAS) pour optimiser la consommation d'énergie ainsi que la latence du NoC.

He et Guo [168] utilisent l'optimisation par colonies de fourmis (ACO) pour résoudre le problème de placement tout en optimisant la consommation d'énergie de communication et le délai.

Chatterjee et al. [169] proposent une méthode heuristique constructive pour résoudre le problème du mapping d'applications sur un NoC (avec une topologie maillée). Leur but est d'optimiser le coût des communications de réseau ainsi que la fiabilité du système.

Bruch et al. [170] présentent un flux d'optimisation permettant le placement des applications sur un NoC, dans le but de répondre aux exigences temporelles et de minimiser les coûts liés à l'utilisation des canaux virtuels. L'approche utilisée est basée sur l'algorithme

génétique NSGAII [1].

2.3.1 Discussion

D'après les références citées ci-dessus, nous constatons que :

- Certaines références [152][153][154][155][156][157][158][159] traitent la problématique du mapping comme un problème d'optimisation mono-objectif. L'optimisation d'une seule métrique entraine la détérioration de d'autres métriques.
- Dans d'autres travaux tels que [167][168][169], les auteurs ont utilisé l'approche d'agrégation (en utilisant une fonction de coût unifiée) afin de prendre en compte plusieurs objectifs lors du mapping. L'inconvénient de leur approche vient de la difficulté d'ajuster les poids, ainsi une connaissance a priori du problème s'avère nécessaire.
- Certains de ces travaux [161][163][165] n'ont pas tenu compte des effets dynamiques du NoC tels que les contentions lors du mapping.
- Une limite importante de ces approches est que seules quelques métaheuristiques sont explorées principalement : NSGA ([161]), NSGAII ([162][163][165][170]), SPEA2 ([160][162]), MOGA ([164]), etc.
- Un autre constat est le nombre d'objectifs pris en compte lors de l'optimisation ; la plupart des travaux ne prend en compte que l'espace d'optimisation bidimensionnel (i.e. optimisent seulement deux fonctions de coût), par exemple, performances et consommation énergétique dans [160], nombre de sauts et l'équilibre thermique [161], l'énergie et bande passante dans [163], etc.

Dans le cadre de cette thèse, nous allons :

- Considérer la résolution du problème du mapping dans un environnement multiobjectif, où plusieurs objectifs peuvent être spécifiés tels que : le temps d'exécution, le taux de communication, l'équilibrage de charge et l'énergie de consommation, etc.
- Utiliser plusieurs métaheuristiques à savoir les P-métaheuristiques (NSGAII [1], SPEA2 [2], PESAII [3], MOCell [171], IBEA [5], FastPGA [4], OMOPSO [45], AbYSS [46] et SMPSO [47]), et les S-métaheuristiques (MOSA [49], AMOSA [6], MOTS [50] et PAES [109]), ainsi que des approches hybridées combinant ces deux

classes de métaheuristiques proposées dans le cadre de cette thèse.

- Prendre en compte les effets dynamiques du NoC (i.e. les contentions) lors de l'exploration du problème du mapping.

2.4 Conclusion

Dans ce chapitre, nous avons présenté les concepts liés à l'optimisation multiobjectif. Ces concepts sont nécessaires pour la modélisation de notre problème du mapping, suivi d'un état de l'art sur les différentes approches du mapping dans le NoC trouvées dans la littérature. Nous avons également discuté les manques et les défis de ces dernières afin de mettre en avant nos contributions dans ce domaine.

Les trois chapitres (3, 4 et 5) qui suivent décrivent nos contributions pour résoudre la problématique du mapping statique des applications parallèles sur un MPSoC hétérogène basé NoC.

Chapitre 3

Mapping des applications parallèles sur un MPSoC hétérogène basé NoC

3.1 Introduction

Comme nous l'avons vu précédemment aux chapitres 1 et 2, le problème de placement (Mapping) des applications sur un MPSoC basé NoC est un problème NP-difficile [41] et multiobjectif. De plus, la résolution de ce problème dépend fortement de sa taille. Vu la complexité grandissante des systèmes actuels, la phase de placement dans la conception de ces derniers ne peut plus être effectuée manuellement (l'espace du problème est trop grand pour tester toutes les alternatives manuellement dans un temps et pour un budget donné). D'où l'intérêt des outils d'exploration (DSE) pour l'aide à la décision.

Ce chapitre présente la première contribution de cette thèse. Il s'agit de proposer un outil d'exploration pour le mapping des applications parallèles sur un MPSoC hétérogène basé NoC.

3.2 Description de l'outil proposé

Notre outil est une extension de l'outil du mapping MPAssign proposé par Bouchababa et al. en 2012 [172]. Dans cette version :

1. Nous avons considéré les architectures MPSoCs hétérogènes en définissant une nouvelle

représentation de la solution et en adaptant les différents opérateurs de reproduction (crossover et mutation). Certaines fonctions de coût sont aussi redéfinies.

2. Nous avons introduit un ensemble d'indicateurs de performance (de qualités) afin d'effectuer facilement la comparaison des fronts de Pareto résultants.
3. Nous avons implémenté de nouvelles métaheuristiques telles que MOSA [49], AMOSA [6], MOTS [50] et MBB [51] dans le but d'enrichir notre outil de mapping et de faire d'éventuelles comparaisons.
4. D'autres métaheurisques ont été proposées dans cette version. Elles sont appelées métaheuristiques hybrides présentées en chapitre 5.
5. En plus des modèles analytiques utilisés pour définir les fonctions de coût, nous avons proposé un modèle de simulation afin de prendre en compte l'aspect dynamique (les contentions) du système lors du placement. Ce modèle est utilisé dans notre cas pour l'estimation de deux métriques de performance : temps d'exécution et énergie de consommation. Il sera détaillé en chapitre 4.

   La figure 3.1 donne une vue globale de l'outil du mapping proposé. Cet outil a comme entrées : (1) le modèle d'application représenté sous forme d'un graphe de tâches, (2) le modèle d'architecture, (3) des fonctions objectives pouvant être spécifiées en utilisant des modèles analytiques ou par simulation, (4) des contraintes d'applications ou d'architectures. Un ensemble d'algorithmes d'optimisation multiobjectif est utilisé pour explorer l'espace du mapping et trouver l'ensemble de solutions de compromis (sortie de l'outil).

3.2.1 Le modèle d'application

Le modèle d'application est représenté sous forme d'un graphe de tâches noté $A(T, E)$ où T est un ensemble (non vide) de sommets représentant les tâches de l'application, et E un ensemble d'arcs e_i reliant ces dernières. Chaque tâche t_i est annotée avec deux vecteurs V_i et C_i, représentant respectivement la consommation d'énergie et le temps d'exécution de la tâche t_i sur chaque type d'éléments de traitement (PE ou core). $Load(t_i)$ indique le nombre d'instructions d'une tâche t_i. Une tâche t_i peut avoir une échéance temporelle (date limite ou deadline) noté $d(t_i)$. Chaque arc $e_i \in E$ correspond à une relation de précédence

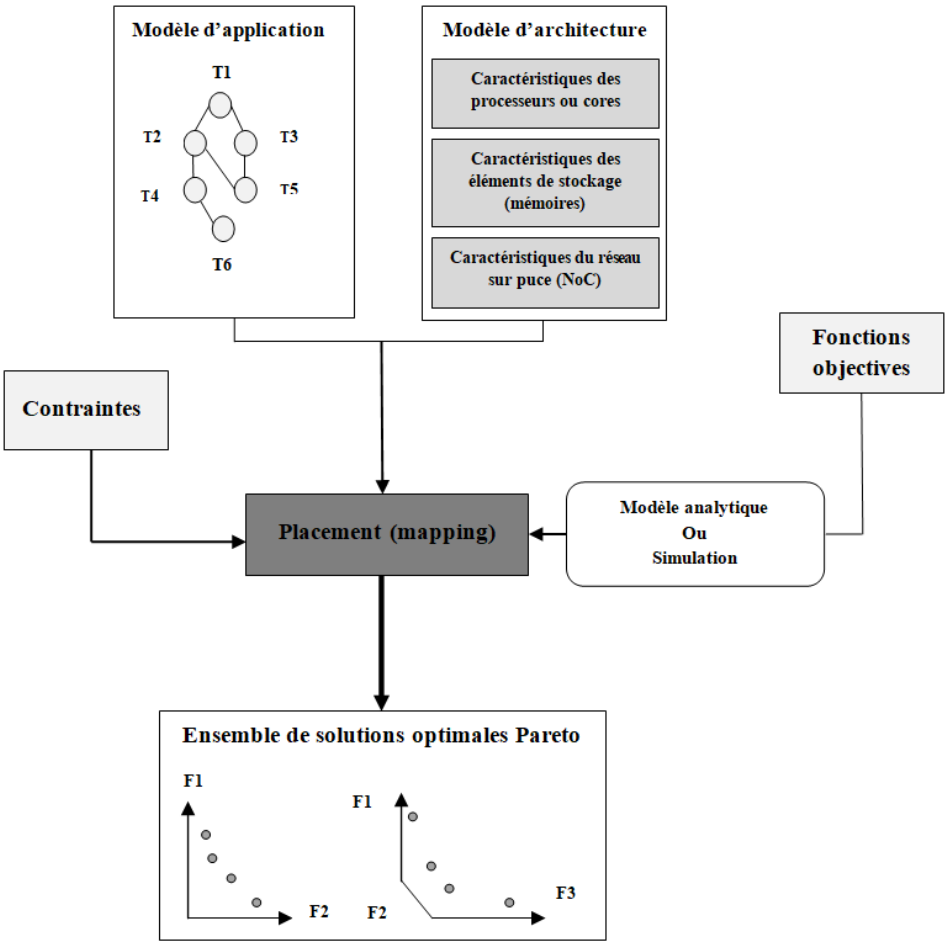


FIGURE 3.1 – Entrées et sortie de l'outil proposé

entre deux tâches annoté par $volume(e_i)$ représentant la quantité de données échangées entre ces tâches.

3.2.2 Le modèle d'architecture

Le modèle d'architecture comprend les principales caractéristiques de la plateforme cible qui exécutera l'application. Les informations considérées dans notre cas sont :

- Les caractéristiques des processeurs (ex. nombre, type, fréquence, etc.)
- Les caractéristiques des éléments de stockage (ex. capacité maximale de la mémoire)
- Les caractéristiques du NoC incluant : les caractéristiques des routeurs (ex. algorithme de routage, modes de commutation, techniques de contrôle de flux, arbitrage, etc.) ; les caractéristiques des liens (ex. sens, débit, etc.) et la topologie du NoC.

3.2.3 Le placement (mapping)

Étant donné : (1) le modèle d'application composé de n tâches, (2) le modèle d'architecture composé de m processeurs interconnectés avec un réseau sur puce (NoC), le problème du placement (mapping) consiste à trouver un placement de ces n tâches sur les m processeurs composant la plateforme cible, tout en optimisant un ensemble de métriques (fonctions de coût) telles que : la consommation d'énergie, le temps d'exécution, etc. Ce problème peut avoir des contraintes applicatives ou architecturales à considérer telles que la bande passante (contrainte architecturale), le deadline des tâches ou d'application (contrainte applicative), etc.

Dans ce qui suit, nous allons décrire les fonctions objectives ainsi que les contraintes considérées dans le cadre de notre travail.

3.2.3.1 Les fonctions objectives

Les fonctions objectives peuvent être définies en utilisant un modèle analytique ou par simulation. Les fonctions objectives (métriques ou fonctions de coût) considérées dans le cadre de notre étude sont : l'équilibrage de charge (load balancing), la communication, l'énergie de consommation et le temps d'exécution. Il est à noter que le concepteur peut spécifier ces métriques lors du placement ou en définir d'autres et les ajouter facilement à

notre outil d'exploration (notre outil est facilement extensible).

Dans la suite de cette thèse, nous utilisons une variable de décision x_{ij} qui est définie comme suit :

$$x_{ij} = \begin{cases} 1 & \text{si la tâche } t_i \text{ est affectée au processeur } PE_j \\ 0 & \text{sinon} \end{cases}$$

1. **Équilibrage de charge (Load balancing)** : cette fonction de coût donne l'écart de charge entre les différents PEs (ou cores) pour un placement donné, de sorte que tous les éléments de traitement du système soient chargés de manière égale (évitant la concentration des tâches dans seulement quelques processeurs ou cores). Cette fonction de coût est définie comme suit :

$$\sum_{j=0}^{P-1} abs\left(\frac{load(PE_j)}{f(PE_j)} - M\right) \tag{3.1}$$

Où $load(PE_j)$ représente la charge du processeur PE_j, exprimée par la somme des instructions des tâches affectées à ce processeur (PE_j).

$$load(PE_j) = \sum_{ti \in T} x_{ij} \times load(t_i) \tag{3.2}$$

M représente la charge moyenne donnée par l'équation 3.3.

$$M = \frac{\sum_{j=0}^{P-1} load(PE_j)}{\sum_{j=0}^{P-1} f(PE_j)} \tag{3.3}$$

$f(PE_j)$ représente la fréquence du processeur PE_j, qui diffère d'un processeur à un autre car la plateforme cible est hétérogène.

2. **Communication** : cette fonction de coût donne la quantité totale de communication entre tous les PEs (ou cores).

$$Commcost = \sum_{e_i \in E} volume(e_i) \times Distance[PE(Src(e_i)), PE(Snk(e_i))] \tag{3.4}$$

Où E est l'ensemble d'arcs du graphe de tâches de l'application, $volume(e_i)$ est la quantité de données échangées par les tâches reliées par l'arc e_i, $Src(e_i)$ et $Snk(e_i)$ représentent respectivement les tâches source et destination reliées par l'arc e_i, $PE(t_i)$ donne le PE sur lequel la tâche t_i est placée (mappée) et $Distance(PE_1, PE_2)$ donne la distance (le nombre de sauts) entre PE_1 et PE_2.

3. **Énergie de consommation** : cette fonction estime l'énergie de consommation totale du système considéré donnée comme suit :

$$E_{total} = E_p + E_{comm} \tag{3.5}$$

Où E_p est l'énergie de traitement et E_{comm} est l'énergie de communication. Soit E_{ij} l'énergie nécessaire pour exécuter la tâche t_i sur le processeur PE_j, l'énergie de traitement peut être calculée comme suit :

$$Ep = \sum_{ti \in T} \sum_{PEj \in P} x_{ij} \times E_{ij} \tag{3.6}$$

L'énergie de communication (énergie de consommation du NoC) est directement proportionnelle au nombre de transitions de bits au sein du réseau. Pour estimer la consommation d'énergie du NoC analytiquement, nous avons implémenté le modèle énergétique bien connu donné par Hu and Marculescu [152], où l'énergie d'un bit de données transféré par le routeur est estimée comme suit :

$$E_{bit} = E_{S_{bit}} + E_{L_{bit}} \tag{3.7}$$

Où $E_{S_{bit}}$ et $E_{L_{bit}}$ représentent respectivement l'énergie consommée sur le commutateur et sur le lien de sortie du routeur. En utilisant l'équation précédente, la consommation d'énergie moyenne pour envoyer un bit de données de PE_i à PE_j peut être calculée comme suit :

$$E_{bit}^{i,j} = (nhops + 1) \times E_{S_{bit}} + nhops \times E_{L_{bit}} + 2 \times E_{PE-R} \tag{3.8}$$

Où E_{PE-R} est l'énergie consommée sur le lien entre le processeur et le routeur. $nhops$ est le nombre de sauts entre le routeur R_i attaché au processeur source PE_i et le routeur R_j attaché au processeur destination PE_j. Si $volume(e_i)$ est la quantité de données échangées par les tâches reliées par l'arc e_i , la consommation d'énergie totale du NoC est déterminée comme suit :

$$E_{NOC} = \sum_{ei \in E} volume(e_i) \times E_{bit}^{PE(src(e_i)),PE(snk(e_i))} \tag{3.9}$$

Où $src(ei)$ et $snk(ei)$ représentent respectivement les tâches source et destination reliées par l'arc e_i. $PE(t_i)$ donne le PE sur lequel la tâche t_i est affectée.

L'inconvénient de ce modèle analytique est qu'il ne considère pas l'énergie consommée par les buffers en cas de contentions. A cet effet, le concepteur peut spécifier le modèle de simulation présenté en chapitre 4 afin d'estimer la consommation d'énergie de communication en considérant les effets dynamiques du NoC (contentions).

4. **Temps d'exécution (Shedule length)** : deux modèles de calcul (analytique et par simulation) sont implémentés dans notre outil afin d'estimer le temps total d'exécution d'une application donnée. Dans cette section, nous allons décrire le modèle analytique implémenté. Le modèle par simulation sera décrit dans le chapitre suivant. Ce dernier est plus réaliste et prend en compte le temps d'attente engendré lors des contentions. Pour définir le temps d'exécution total d'une application, nous allons définir deux attributs $ST(t_i, PE_j)$ et $FT(t_i, PE_j)$ qui représentent respectivement le temps début (Start Time) et le temps fin (Finish Time) d'exécution de la tâche t_i sur le processeur PE_j. Les valeurs de ST et FT sont calculées comme suit :

$$ST(t_i, PE_j) = \begin{cases} max(0, FT(t_j, PE_j)), \text{si } pred(t_i) = 0 \\ \\ max(FT(t_j, PE_j), \max_{t_k \in pred(t_i)}(FT(t_k, PE_k) + Tcomm_{ki})), \text{sinon} \end{cases}$$

$$FT(t_i, PE_j) = ST(t_i, PE_j) + C_{ij} \tag{3.10}$$

Où $pred(t_i)$ est l'ensemble des tâches prédécesseurs de la tâche t_i, $FT(t_j, PE_j)$ représente le temps de fin de l'exécution de la dernière tâche exécutée sur le même processeur PE_j où t_i est placée (mappée). $FT(t_k, PE_k)$ est le temps de fin de la tâche t_k, ou $t_k \in pred(t_i)$. $Tcomm_{ki}$ est le temps nécessaire à la communication entre deux tâches t_k et t_i.

L'équation (3.10) donne le temps de fin de l'exécution de la tâche t_i. C_{ij} représente le temps d'exécution de la tâche t_i sur le processeur PE_j. Une fois que toutes les tâches sont placées, le temps d'exécution total (T_{total}) de l'application est donné par l'équation (3.11) comme suit :

$$T_{total} = \max_{t_i \in T} FT(t_i) \tag{3.11}$$

3.2.3.2 Les contraintes d'application et d'architecture

Dans cette section, nous allons présenter les contraintes d'application et d'architecture considérées dans cette thèse. Comme pour les fonctions objectives, le concepteur peut spécifier les contraintes implémentées dans l'outil ou en définir d'autre aisément.

1. Affectation de la tâche : chaque tâche est affectée à un seul processeur, c'est-à-dire :

$$\sum_{j=0}^{P-1} x_{ij} = 1, \forall i \in [0, NBT - 1] \tag{3.12}$$

   où P est le nombre de processeurs (PEs) et NBT est le nombre de tâches.

2. Échéance temporelle (ou deadline) : nous pouvons fixer pour chaque tâche ou application un deadline. Dans ces deux cas, leur temps de fin doit être inférieur à leurs deadlines respectifs.

3. Pré-placement : dans certains cas, on peut prédéfinir un placement de tâches sur des processeurs spécifiques (comme les accélérateurs dédiés) pour des raisons spécifiques à l'application.

Lors de la résolution du problème de placement (mapping), le concepteur est souvent appelé à spécifier plusieurs objectifs à optimiser simultanément. Ces objectifs sont souvent conflictuels (en contradiction), à titre d'exemples : le temps d'exécution vs énergie, l'équilibrage de charge vs communication, etc. Donc trouver une solution unique du mapping qui optimise tous ces objectifs simultanément n'est pas possible. Notre but est de trouver l'ensemble de solutions du mapping compromis entre les différents objectifs contradictoires, communément appelé solutions optimales de Pareto. Pour cela, nous avons utilisé les métaheuristiques comme approches de résolution de notre problème. Comme présenté précédemment au chapitre 2, les métaheuristiques ont prouvé leur succès dans la résolution de plusieurs problèmes PMOs. Par conséquent, nous les avons jugées adéquates pour résoudre notre problème de placement qui est une instance de cette classe de problèmes (PMOs). Deux classes de métaheuristiques ont été utilisées : les P-métaheuristiques et les S-métaheuristiques.

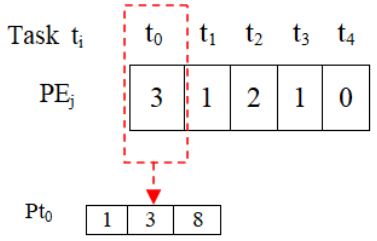


FIGURE 3.2 – Exemple d'un chromosome

3.3 Les métaheuristiques à base de population de solutions pour le problème du mapping

Dans cette section, nous présentons comment les métaheuristiques à base de population de solutions (les P-métaheuristiques) ont été adaptées pour résoudre notre problématique. Pour cela, nous avons : (1) défini un nouveau codage de la solution, et (2) adapté les différents opérateurs de reproduction (crossover et mutation).

3.3.1 Représentation (ou codage) de la solution

Pour notre problème de placement, nous avons utilisé un codage entier : chaque solution de placement est représentée par un tuple $(x_1, x_2, ..., x_n)$, où x_i donne le PE sur lequel la tâche t_i doit être placée (mappée). La valeur de chaque variable x_i est choisie dans l'ensemble Pt_i. Cet ensemble Pt_i comporte la liste de tous les processeurs permissibles (autorisés) pour la tâche t_i. La Figure 3.2 donne un exemple d'un chromosome (ou particule) où 5 tâches sont placées (mappées) sur une plateforme hétérogène donnée. Supposons que Pt_0={1, 3, 8} représente la liste des processeurs permissibles pour la tâche t_0. Dans cet exemple, la valeur de t_0 est 3, choisie parmi sa liste des processeurs permissibles Pt_0.

3.3.2 Les opérateurs de mutation et de croisement

Après avoir appliqué les opérateurs de reproduction (crossover et mutation), il est possible que certaines solutions (chromosomes) soient invalides en raison de l'affectation d'une tâche donnée à un processeur qui n'appartient pas à sa liste des processeurs permissibles. Les figures 3.3 (a) et 3.3 (b) représentent respectivement des exemples de chromosomes

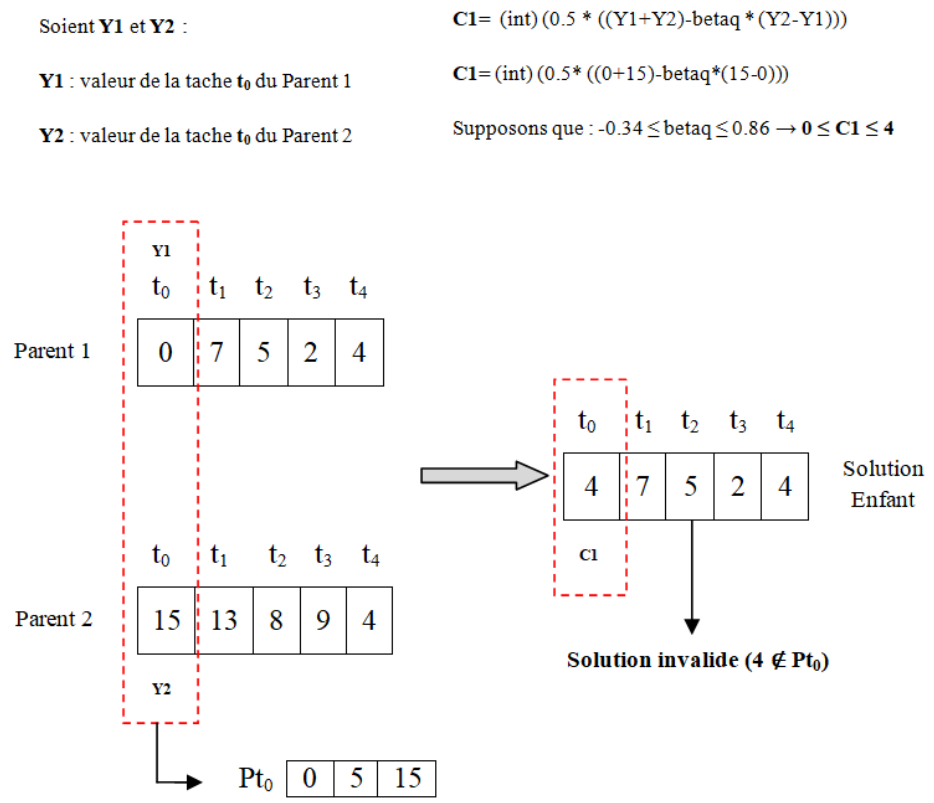
invalides après avoir appliqué des exemples d'opérateurs de croisement (SBX crossover) et de mutation (Polynomial mutation).

Pour remédier à ce problème rencontré, nous appliquons les étapes suivantes :

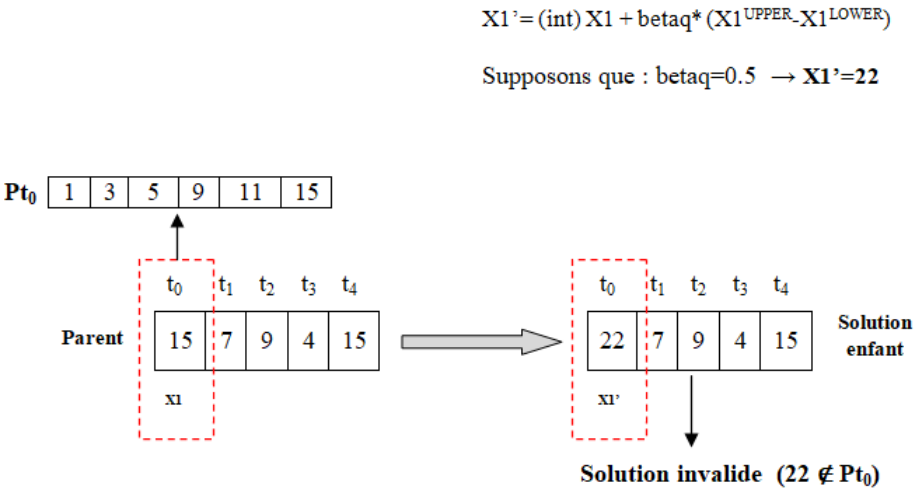
- Étape 1 : nous avons transformé le codage initial en un codage intermédiaire, en utilisant les indices des éléments de Pt_i comme montré par la Figure 3.4. Le chromosome du parent (15, 7, 9, 4, 15) donné en 3.3 (b) sera transformé en (5, 1, 3, 0, 2). 5 est l'indice de l'élément 15, 1 est l'indice de l'élément 7, 3 est l'indice de l'élément 9, etc.. L'ensemble Pt_i sera transformé en $P't_i$. L'objectif derrière cette nouvelle représentation est de minimiser le nombre de solutions invalides générées. Malgré l'avantage de cette nouvelle représentation, les solutions invalides ne sont pas toutes éliminées. Un exemple de solution invalide générée par cette nouvelle représentation est donné en Figure 3.5, où la tâche t_0 est affectée au processeur avec l'indice 7 qui n'est pas dans l'ensemble des indices de ses processeurs permissibles P't$_0$=$\{0, 1, 2, 3, 4, 5\}$.
- Étape 2 : comme seconde étape, nous avons proposé l'algorithme 1 pour corriger ces solutions invalides. La solution résultante après l'application de l'algorithme 1 est donnée par la Figure 3.6.
- Étape 3 : comme dernière étape, le codage intermédiaire obtenu (corrigé) est retransformé au codage initial (voir Figure 3.7). Le chromosome (1, 1, 1, 3, 0, 2) est transformé en (3, 7, 9, 4, 15). 3 est l'élément avec l'indice 1 dans l'ensemble Pt_0, 7 est l'élément avec l'indice 1 dans l'ensemble Pt_1, etc.

Algorithme 1 Correction des solutions invalides

pour toutes les solutions invalides **faire**
  pour toutes les tâches **faire**
    si val(t$_i$) $\notin P'(t_i)$ **alors**
      $v = random(P'(t_i))$
      $val(t_i) = v$
    fin si
  fin pour
fin pour



(a) Solution invalide après avoir appliqué l'opérateur de croisement SBX (SBX crossover)



(b) Solution invalide après avoir appliqué l'opérateur de mutation polynomiale (Polynomial mutation).

FIGURE 3.3 – Exemples de solutions invalides après avoir appliqué les opérateurs de reproduction

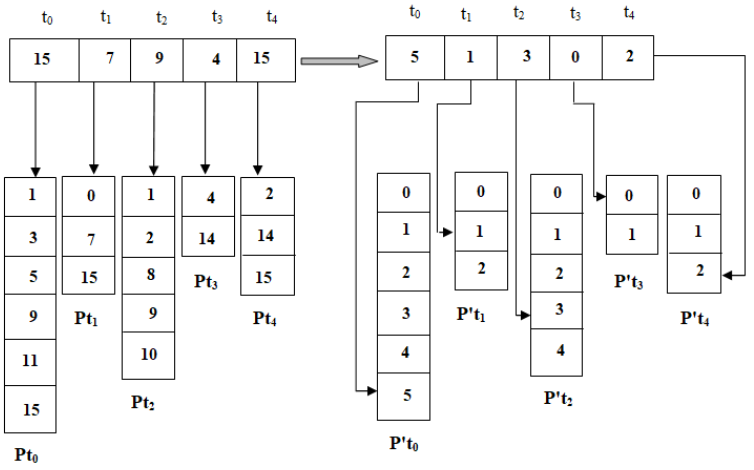


FIGURE 3.4 – Codage intermédiaire

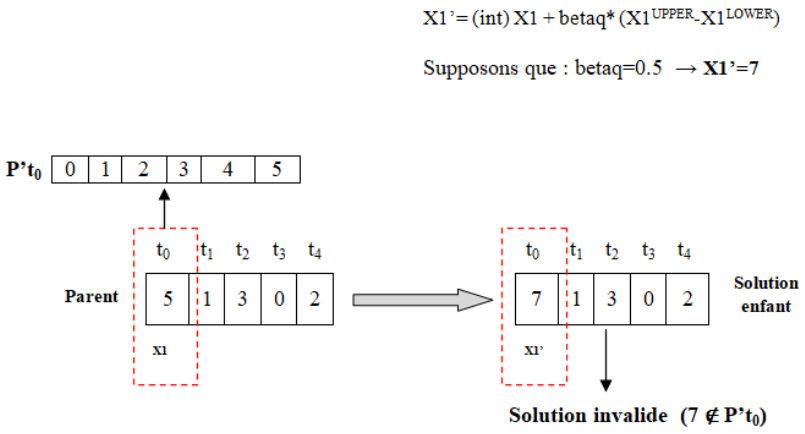


FIGURE 3.5 – Exemple de solution invalide avec la nouvelle représentation après application de la mutation polynomiale (polynomial mutation).

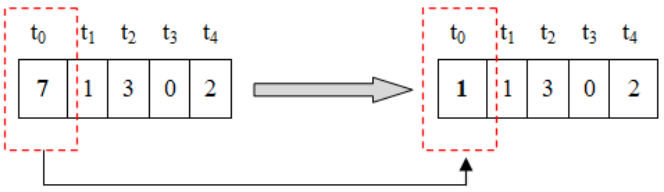


FIGURE 3.6 – Solution invalide corrigée avec l'algorithme 1

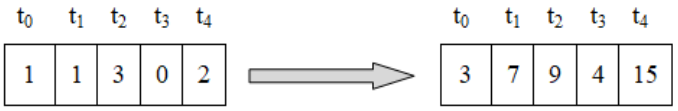


FIGURE 3.7 – Retour au codage initial

3.4 Les métaheuristiques à base de solution unique pour le problème du mapping

En plus des métaheuristiques basées population de solutions (P-metaheuristiques), nous avons aussi utilisé une autre classe de métaheuristiques basées sur une seule solution appelée S-metaheuristiques telles que le récuit simulé multiobjectif (MOSA [49], AMOSA [6]), la recherche tabou multiobjectif (MOTS) [50] et PAES [109] pour résoudre notre problématique. Pour cela, nous avons défini : (1) la structure de la solution à manipuler ainsi que (2) la fonction de voisinage.

3.4.1 Représentation de la solution

La structure de la solution potentielle utilisée dans les S-métaheuristiques est la même que celle utilisée par les P-métaheuristiques représentée en Figure 3.2. Comme présenté précédemment dans le chapitre 2, cette classe de métaheuristiques travaille sur un seul point de l'espace de recherche à un instant donné, en essayant de l'améliorer (c.à.d trouver une nouvelle solution dans son voisinage), et cela en appliquant un opérateur dit opérateur de voisinage décrit ci-dessous.

3.4.2 Opérateur du voisinage

Dans notre cas, l'opérateur de voisinage utilisé est le même que l'opérateur de mutation utilisé par les P-métaheuristiques que nous avons adapté pour notre problématique en section 3.3.2. Par conséquent, tous les opérateurs utilisés par les P-metaheuristiques tels que : Flip mutation, Polynomial mutation, Uniform mutation, Non Uniform mutation peuvent être spécifiés comme opérateur de voisinage pour les S-métaheuristiques.

3.5 Evaluation expérimentale

Dans cette section, nous présentons un ensemble d'expériences illustrant l'utilisation de notre outil du mapping proposé pour résoudre plusieurs instances de problèmes de placement. Pour cela, TGFF [173] est utilisé pour générer un ensemble de graphes de tâches, en faisant varier le nombre de tâches. D'autre part, le modèle d'architecture utilisé est composé de k types de processeurs interconnectés avec un réseau sur puce (NoC). Plusieurs topologies ont été explorées à savoir : 2D-mesh, 2D-torus et Spidergon (voir Figure 3.8).

Pour évaluer et comparer les performances des algorithmes d'optimisation multiobjectif, deux propriétés sont généralement à considérer à savoir : (1) la qualité des solutions retournées ainsi que (2) le temps de calcul requis (runtime). La qualité des solutions est mesurée en utilisant la métrique de performance IGD définie précédemment en section 2.2.4. Pour chaque algorithme, nous avons effectué 30 exécutions (runs), et les tableaux résultants représentent la moyenne (MOY) ainsi que l'écart-type (ET) de l'indicateur de qualité considéré (IGD). Les deux meilleurs résultats pour chaque problème sont représentés dans les tables (3.3, 3.4, 3.6 et 3.7) par deux niveaux de gris : le gris foncé indique l'algorithme qui obtient la meilleure valeur de l'indicateur, et le gris clair souligne l'algorithme qui obtient la deuxième meilleure valeur de l'indicateur. Tous les calculs ont été effectués sur un PC Intel (R) Core (TM) i7 CPU, 2.7GHz, avec 8 Go de RAM.

La TABLE 3.1 donne le paramétrage des algorithmes utilisé dans la suite de nos expériences. Les valeurs de ces paramètres ont été choisies arbitrairement au lieu de trouver les paramètres les plus appropriés pour obtenir le meilleur résultat pour chaque algorithme car l'objectif de cette étude est de présenter l'utilisation de l'outil proposé. L'exemple d'évaluation dans lequel le choix des paramètres est pris en compte est donné dans le chapitre 5. Cependant, nous avons réalisé une série d'expériences pour montrer l'effet du paramétrage de certains algorithmes sur la qualité des solutions du mapping retournées.

3.5.1 Etude comparative des différents algorithmes

Pour effectuer une comparaison entre les algorithmes offerts par notre outil de mapping incluant les P-métaheuristiques (NSGAII [1], SPEA2 [2], PESAII [3], FastPGA [4], IBEA

TABLE 3.1 – Paramétrage des algorithmes

Les P-métaheuristiques	(NSGAII [1], FastPGA [4], SPEA2 [2], PESAII [3], IBEA [5], MOCell [171], AbYSS [46], OMOPSO [45], SMPSO [47])
NSGAII/ FastPGA	
Taille de la population	100
Max Evaluation	25000
Probabilité de mutation (pm)	1.0/L (L : longeur de la solution)
Probabilité de croisement (pc)	0.8
SPEA2/ PESAII/ IBEA/ MOCell/ AbYSS	
Taille de la population	100
Taille de l'archive	100
Max Evaluation	25000
Probabilité de mutation (pm)	1.0/L (L : longeur de la solution)
Probabilité de croisement (pc)	0.8
SMPSO/ OMOPSO	
Taille de l'essaim	100
Max Iterations	500
Taille de l'archive	100
Probabilité de mutation (pm)	1.0/L (L : longeur de la solution)
Les S-métaheuristiques	(MOSA [49], AMOSA [6], MOTS [50], PAES [109])
MOSA	
Température initiale (T_0)	900
Température finale (T_1)	10^{-3}
Taux de refroidissement (α)	0.8
Max Iterations	5000
AMOSA	
Température initiale (T_0)	700
Température finale (T_1)	10^{-3}
Taux de refroidissement (α)	0.8
Max Iterations	1000
HL	100
SL	110
Gamma	1.7
MOTS	
K	10
Max Iterations	5000
Max Pareto Rank (R_{max})	4
Max Neighbours	50
Size min of Tabu List (Tab_{min})	10
Size max of Tabu List (Tab_{max})	15
PAES	
Taille de l'archive	100
Probabilité de mutation (pm)	1.0/L (L : longueur de la solution)

TABLE 3.2 – Exemples de problèmes de mapping

problème du mapping	Graphes de tâches GTs	Plateformes
P1	5 tâches	3x3 2D-mesh (3 types de processeurs)
P2	10 tâches	4x4 2D-mesh (voir Figure 3.8 (a))
P3	20 tâches	4x4 2D-torus (voir Figure 3.8 (b))
P4	100 tâches	Spidergon (voir Figure 3.8 (c))
P5	100 tâches	4x4 2D-mesh (voir Figure 3.8 (a))
P6	100 tâches	8x8 2D-torus (6 types de processeurs)

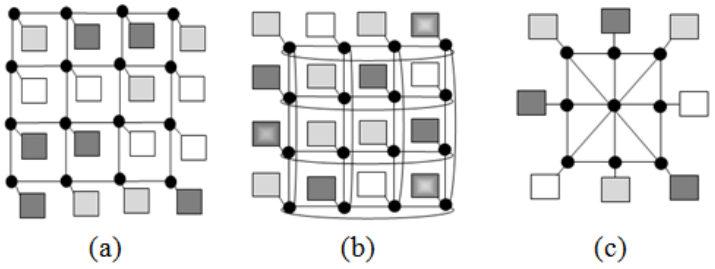


FIGURE 3.8 – Exemples de plateformes : (a) 2D-mesh, (b) 2D-torus, (c) Spidergon

[5], MOCell [171], AbYSS [46], OMOPSO [45], SMPSO [47]) et les les S-métaheuristiques (MOSA [49], AMOSA [6], MOTS [50] et PAES [109]), nous avons considéré plusieurs cas de figures :

- Cas 1. Variation des graphes de tâches et des plateformes : dans cette première expérience, plusieurs instances du problème du mapping (PMs) ont été considérées. Ces dernières diffèrent selon le graphe de tâches et la plateforme cible utilisée (voir TABLE 3.2). Deux fonctions de coût conflictuelles sont spécifiées pour être optimisées, à savoir : l'équilibrage de charge (load varaince) et la communication. Elles sont définies par des modèles analytiques. Les résultats de ces expériences sont donnés par les TABLES 3.3 et 3.4.

  Nous pouvons observer d'après les résultats donnés en TABLE 3.3 et 3.4 que l'ordre de performance des métaheuristiques change selon le problème du mapping considéré (graphe de tâches et plateformes utilisés). Par exemple, pour le problème P1, les deux meilleurs algorithmes sont respectivement AMOSA et MOTS, tandis que pour P4, les meilleurs résultats sont donnés par SMPSO et OMOPSO respectivement. La TABLE 3.5 donne d'une manière plus explicite le classement de la performance des algorithmes comparés pour chaque instance du problème du mapping (PM), et cela selon le test de Friedman. Le test de Friedman suppose que plus la valeur de

l'indicateur (IGD) est faible mieux c'est. Ceci confirme qu'un outil de mapping doit fournir plusieurs métaheuristiques afin d'explorer différents espaces de solutions.

- Cas 2. Variation du nombre d'objectifs à optimiser (NObj) : les expériences présentées dans ce second cas donnent une comparaison entre les algorithmes offerts par notre outil du mapping, et cela en fonction du nombre d'objectifs spécifié, pour un problème du mapping donné. Soit P5 un exemple du problème du mapping utilisé dans ces expériences et présenté en TABLE 3.2. Les TABLES 3.6 et 3.7 donnent respectivement les résultats de comparaison des algorithmes en spécifiant deux objectifs (load balancing, communication), trois objectifs (load balancing, communication et énergie de consommation) et quatre objectifs (load balancing, communication, énergie de consommation et temps d'exécution). Ces objectifs sont définis par des modèles analytiques.

  D'après les TABLES 3.6 et 3.7, nous constatons : (1) pour un même problème du mapping (pour notre cas nous avons considéré le P5), les performances des métaheuristiques changent en fonction du nombre d'objectifs spécifié. A titre d'exemple, les résultats (les solutions compromis) retournés par OMOPSO sont mauvais en considérant deux objectifs, mais ils sont bons en cas de trois et quatre objectifs. Ceci est indiqué d'une manière plus explicite dans la TABLE 3.8, où la valeur de IGD pour OMOPSO est 12.0 en considérant deux objectifs, et 2.0 pour trois et quatre objectifs. (2) on peut aussi voir que le temps d'exécution des métaheuristiques augmente avec l'augmentation du nombre d'objectifs considéré, par exemple, le temps d'exécution de NSGAII pour 2, 3 et 4 objectifs est respectivement 0.560s, 0.950s et 6.791s.

- Cas 3. Effet du choix des paramètres des métaheuristiques sur la qualité des solutions du mapping retournées : dans ce troisième cas, nous présentons un ensemble d'autres expériences montrant comment les métaheuristiques sont sensibles à leurs paramètres. La Figure 3.9 présente plusieurs expériences sur l'effet de certains paramètres comme : (1) nombre d'évaluations ($MaxEval$) (Figures 3.9 (a), (b) et (c)), (2) Température initiale (T_0)(Figure 3.9 (d)), (3) type d'opérateur de mutation (Figure 3.9 (e)), et (4) probabilité de croisement (P_c) (Figure 3.9 (f)) sur la performance de quelques

métaheuristiques proposées par notre outil de mapping telles que : NSGAII [1], SMPSO [47], AMOSA [6], SPEA2 [2] et PESAII [3]. Dans toutes ces expériences, nous avons fait varier un paramètre pour chaque algorithme et d'autres paramètres ont été fixés. L'étude a été faite sur le problème P4 présenté en TABLE 3.2. Load balancing et communication ont été spécifiées comme fonctions objectives à optimiser dans ces expériences.

Les Figures 3.9 (a), (b) et (c), montrent clairement que plus le nombre d'évaluations est grand, mieux est la qualité des solutions du mapping trouvée, et cela au détriment du temps d'exécution (voir les Tables 1,2 et 3 de la Figure 3.9). Un autre exemple de paramètre affectant l'algorithme AMOSA est donné en Figure 3.9 (d). Il s'agit du paramètre α dont la valeur varie entre $]0, 1[$. Plus la valeur de ce paramètre se rapproche de 1 mieux est la performance de AMOSA, et cela aussi au détriment du temps d'exécution de ce dernier (voir Table 4 de la Figure 3.9). D'autres paramètres importants affectant la performance des métaheuristiques sont les opérateurs de reproduction : (1) mutation (type et probabilité) et (2) croisement (type et probabilité). Ceci est prouvé clairement dans les deux expériences montrées en Figures 3.9 (e) et (f). La figure 3.9 (e) indique que l'opérateur de mutation de type FlipMutation utilisé par SPEA2 pour résoudre le problème du mapping considéré (P4) donne de meilleurs résultats comparés aux autres types d'opérateurs de mutation. D'autre part, la valeur affectée à la probabilité de croisement ($P_c = 0.6$) pour l'algorithme PESAII montre de bons résultats pour ce cas de figure comparée aux autres valeurs de P_c (0.7, 0.8, 0.9, 1.0).

- Cas 4. Relation entre le choix des paramètres d'un algorithme donné et le problème supposé : comme dernière expérience, nous tentons de trouver les bons paramètres d'un algorithme, soit l'exploration du paramètre probabilité de croisement (P_c) pour l'algorithme PESAII, et cela pour résoudre deux problèmes différents P4 et P6 présentés en TABLE 3.2.

  Comme montré en Figure 3.10, la valeur de probabilité de croisement (P_c) de PESAII donnant de meilleurs résultats diffère selon le problème du mapping considéré. Par exemple, pour ce cas de figure, les meilleures probabilités de croisement (c.à.d. la

meilleure configuration) donnant les meilleurs résultats pour les problèmes P4 et P6 sont respectivement $P_c = 0.6$ (voir Table 5) et $P_c = 0.9$ (voir Table 6).

3.5.2 Discussion

L'outil de mapping proposé offre plusieurs métaheuristiques pouvant être spécifiées lors de la résolution du problème du mapping. La première question est : quelle est la métaheuristique la plus performante (qualité des solutions et temps) pour résoudre un problème du mapping donné ? Dans cette section, nous tentons de résumer notre étude expérimentale, et donner ainsi une suite de directives sur l'utilisation de l'outil proposé. Selon les résultats expérimentaux présentés ci-dessus, le choix de la métaheuristique pour résoudre un problème du mapping donné est fortement lié à plusieurs facteurs :

1. Les entrées de l'outil telles que :
   - Le graphe de tâches et la plateforme utilisés (premier cas d'étude) : la performance d'une métaheuristique varie selon les caractéristiques du problème du mapping considéré. Cela est démontré par les résultats donnés par les TABLES 3.3, 3.4 et 3.5. Selon ces résultats, on peut constater que par exemple, l'algorithme MOTS donne de bons résultats pour les problèmes P1, P2 et P4, et de mauvais résultats pour P3, P5 et P6.
   - Le nombre d'objectifs à spécifier (deuxième cas d'étude) : en plus du graphe de tâches et de la plateforme utilisés, la performance d'une métaheuristique change selon le nombre d'objectifs à optimiser, comme le démontrent les résultats donnés en TABLES 3.6, 3.7 et 3.8. A titre d'exemple, l'algorithme OMOPSO donne de bons résultats lors de l'optimisation de trois et de quatre objectifs et de mauvais résultats lors de l'optimisation de deux objectifs. On constate aussi, que l'augmentation du nombre d'objectifs à optimiser engendre une augmentation du temps d'exécution et cela pour toutes les métaheuristiques données par notre outil de mapping.
2. Les paramètres des métaheuristiques.

   Un autre paramètre important à prendre en considération est le choix de paramètres

1. MOY : la moyenne
2. ET : l'écart-type

PM			NSGAII	FastPGA	SPEA2	PESAII	IBEA	MOCell	AbYSS	SMPSO
P1	IGD	MOY [1]	$2.87e-03$	$4.04e-03$	$3.24e-03$	$4.40e-03$	$3.26e-03$	$4.05e-03$	$3.50e-03$	$2.73e-03$
		ET [2]	$1.2e-03$	$9.0e-04$	$1.1e-03$	$1.3e-03$	$1.7e-03$	$1.3e-03$	$7.9e-04$	$5.4e-04$
		Temps	0.145s	6,937s	2,304s	0,189s	1,913s	0,088s	0,044s	0,096s
P2	IGD	MOY	$4.90e-03$	$7.69e-03$	$5.13e-03$	$6.86e-03$	$6.79e-03$	$6.95e-03$	$6.56e-03$	$2.48e-03$
		ET	$1.3e-03$	$2.5e-03$	$1.3e-03$	$2.1e-03$	$2.0e-03$	$2.1e-03$	$1.9e-03$	$6.4e-04$
		Temps	0,170s	6,299s	1,413s	0,236s	1,952s	0,122s	0,063s	0,168s
P3	IGD	MOY	$3.60e-02$	$5.63e-02$	$4.10e-02$	$4.93e-02$	$4.73e-02$	$4.82e-02$	$5.07e-02$	$1.94e-02$
		ET	$1.0e-02$	$1.2e-02$	$1.2e-02$	$1.2e-02$	$1.4e-02$	$1.1e-02$	$9.4e-03$	$1.9e-03$
		Temps	0,211s	4,721s	1,090s	0,347s	2,014s	0,169s	0,095s	0,318s
P4	IGD	MOY	$1.95e-01$	$2.49e-01$	$2.23e-01$	$2.20e-01$	$2.22e-01$	$1.99e-01$	$2.04e-01$	$4.23e-02$
		ET	$3.3e-02$	$1.9e-02$	$3.5e-02$	$1.7e-02$	$1.9e-02$	$2.1e-02$	$3.8e-02$	$2.8e-03$
		Temps	0,534s	3,125s	1,422s	0,790s	2,424s	0,624s	0,364s	0,925s
P5	IGD	MOY	$8.18e-02$	$9.10e-02$	$8.86e-02$	$8.36e-02$	$8.82e-02$	$8.13e-02$	$8.30e-02$	$2.78e-02$
		ET	$8.5e-03$	$6.2e-03$	$8.5e-03$	$6.3e-03$	$5.4e-03$	$6.8e-03$	$6.9e-03$	$1.4e-03$
		Temps	0,560s	0,950s	1,144s	0,734s	2,421s	0,568s	0,368s	1,086s
P6	IGD	MOY	$5.94e-02$	$6.25e-02$	$6.14e-02$	$5.99e-02$	$6.21e-02$	$6.01e-02$	$5.94e-02$	$3.62e-02$
		ET	$4.1e-03$	$4.1e-03$	$4.0e-03$	$3.6e-03$	$2.6e-03$	$3.6e-03$	$4.9e-03$	$1.3e-03$
		Temps	0,718s	1,006s	1,306s	0,928s	2,608s	0,744s	0,480s	1,519s

TABLE 3.3 – Résultats des expériences du cas 1

PM			OMOPSO	MOSA	AMOSA	MOTS	PAES
P1	IGD	MOY	$2.95e-03$	$1.30e-02$	$0.00e+00$	$0.00e+00$	$1.25e-02$
		ET	$6.5e-04$	$3.7e-03$	$0.00e+00$	$0.00e+00$	$5.8e-03$
		Temps	$0,072$s	$0,0001$s	$0,140$s	$0,263$s	$0,048$s
P2	IGD	MOY	$1.05e-02$	$3.49e-02$	$6.45e-03$	$4.81e-03$	$7.92e-03$
		ET	$3.6e-03$	$6.8e-03$	$1.7e-03$	$1.2e-03$	$1.7e-03$
		Temps	$0,123$s	$0,0002$s	$0,105$s	$0,423$s	$0,068$s
P3	IGD	MOY	$2.36e-02$	$1.52e-01$	$4.67e-02$	$5.34e-02$	$4.08e-02$
		ET	$3.1e-03$	$1.4e-02$	$1.6e-02$	$9.1e-03$	$1.2e-02$
		Temps	$0,197$s	$0,0002$s	$0,127$s	$0,706$s	$0,087$s
P4	IGD	MOY	$6.29e-02$	$4.49e-01$	$1.38e-01$	$1.64e-01$	$1.97e-01$
		ET	$2.7e-03$	$2.1e-02$	$1.0e-02$	$6.5e-03$	$2.1e-02$
		Temps	$0,656$s	$0,0005$s	$0,303$s	$2,273$s	$0,243$s
P5	IGD	MOY	$1.16e-01$	$1.82e-01$	$7.88e-02$	$8.99e-02$	$6.44e-02$
		ET	$1.5e-02$	$9.1e-03$	$5.9e-03$	$3.2e-03$	$5.0e-03$
		Temps	$0,777$s	$0,0006$s	$0,349$s	$2,522$s	$0,265$s
P6	IGD	MOY	$4.22e-02$	$1.13e-01$	$5.16e-02$	$6.09e-02$	$5.28e-02$
		ET	$8.1e-04$	$6.1e-03$	$2.9e-03$	$1.6e-03$	$4.8e-03$
		Temps	$1,418$ss	$0,0009$s	$0,572$s	$3,817$s	$0,395$s

TABLE 3.4 – Suite des résultats des expériences du cas 1

TABLE 3.5 – Test Friedman des expériences du cas 1 selon l'indicateur de qualité IGD

PMs / Algorithmes	P1	P2	P3	P4	P5	P6
NSGAII	4.0	3.0	3.0	5.0	5.0	5.0
FastPGA	9.0	10.0	12.0	12.0	11.0	12.0
SPEA2	6.0	4.0	5.0	11.0	9.0	10.0
PESAII	11.0	8.0	9.0	9.0	7.0	7.0
IBEA	7.0	7.0	7.0	10.0	8.0	11.0
MOCell	10.0	9.0	8.0	7.0	4.0	8.0
AbYSS	8.0	6.0	10.0	8.0	6.0	6.0
SMPSO	3.0	1.0	1.0	1.0	1.0	1.0
OMOPSO	5.0	12.0	2.0	2.0	12.0	2.0
MOSA	13.0	13.0	13.0	13.0	13.0	13.0
AMOSA	1.5	5.0	6.0	3.0	3.0	3.0
MOTS	1.5	2.0	11.0	4.0	10.0	9.0
PAES	12.0	11.0	4.0	6.0	2.0	4.0

lors de la résolution d'un problème du mapping donné, et cela pour chaque métaheuristique, vu que ces dernières sont par leurs nature très sensibles à leurs paramètres. Cela est démontré par les Figures 3.9 (a, b, c, d, e, f). Il est à noter aussi, que le choix des paramètres est dépendant du problème du mapping considéré (voir les Figures 3.10 (a) et (b)). Donc, une analyse de sensibilité doit être faite pour donner de bons paramètres à chaque algorithme avant toute étude comparative.

Comme conclusion, il n'existe pas de paramétrage optimal standard pour une métaheuristique donnée, puisque cela est lié au problème du mapping considéré.

3.6 Conclusion

Dans ce chapitre, nous avons en premier lieu présenté l'outil de placement proposé. Ensuite, nous avons exposé une suite de résultats expérimentaux afin de donner des lignes directrices sur l'utilisation de l'outil proposé.

Une suite de fonctions objectives a été présentée et utilisée dans l'évaluation expérimentale. Ces fonctions sont décrites par des modèles analytiques (voir section 3.2.3.1). Dans le chapitre qui suit, nous présentons en plus de ces modèles analytiques, un modèle de simulation que le concepteur peut spécifier comme fonction objective lors de l'exploration des solutions de placement.

NObj			NSGAII	FastPGA	SPEA2	PESAII	IBEA	MOCell	AbYSS	SMPSO
2	IGD	MOY	$8.18e-02$	$9.10e-02$	$8.86e-02$	$8.36e-02$	$8.82e-02$	$8.13e-02$	$8.30e-02$	$2.78e-02$
		ET	$8.5e-03$	$6.2e-03$	$8.5e-03$	$6.3e-03$	$5.4e-03$	$6.8e-03$	$6.9e-03$	$1.4e-03$
		Temps	$0,560$s	$0,950$s	$1,144$s	$0,734$s	$2,421$s	$0,568$s	$0,368$s	$1,086$s
3	IGD	MOY	$6.59e-02$	$7.25e-02$	$6.91e-02$	$6.55e-02$	$6.63e-02$	$6.56e-02$	$5.33e-02$	$2.72e-03$
		ET	$5.6e-03$	$5.0e-03$	$4.8e-03$	$4.1e-03$	$3.2e-03$	$3.7e-03$	$8.3e-03$	$1.5e-04$
		Temps	$0,950$s	$1,540$s	$1,580$s	$1,345$s	$3,002$s	$0,928$s	$0,727$s	$1,527$s
4	IGD	MOY	$2.17e-02$	$2.54e-02$	$2.41e-02$	$2.33e-02$	$2.20e-02$	$2.20e-02$	$1.61e-02$	$6.50e-03$
		ET	$1.5e-03$	$8.7e-04$	$1.1e-03$	$1.4e-03$	$1.1e-03$	$1.1e-03$	$2.7e-03$	$3.9e-04$
		Temps	$6,791$s	$13,063$s	$7,162$s	$11,450$s	$7,222$s	$6,687$s	$8,910$s	$265,742$s

TABLE 3.6 – Résultats des expériences du cas 2

PM			OMOPSO	MOSA	AMOSA	MOTS	PAES
2	IGD	MOY	$1.16e-01$	$1.82e-01$	$7.88e-02$	$8.99e-02$	$6.44e-02$
		ET	$1.5e-02$	$9.1e-03$	$5.9e-03$	$3.2e-03$	$5.0e-03$
		Temps	$0,777$s	$0,0006$s	$0,349$s	$2,522$s	$0,265$s
3	IGD	MOY	$1.82e-02$	$1.30e-01$	$5.72e-02$	$6.50e-02$	$5.09e-02$
		ET	$6.0e-04$	$5.0e-03$	$2.1e-03$	$2.0e-03$	$3.7e-03$
		Temps	$1,325$s	$0,001$s	$0,930$s	$5,139$s	$0,642$s
4	IGD	MOY	$1.21e-02$	$3.90e-02$	$2.53e-02$	$2.44e-02$	$1.90e-02$
		ET	$4.4e-04$	$1.4e-03$	$5.0e-04$	$4.6e-04$	$1.0e-03$
		Temps	$377,501$s	$0,016$s	$8,953$s	$35,616s$	$11,522$s

TABLE 3.7 – Suite des résultats des expériences du cas 2

TABLE 3.8 – Test Friedman des expériences du cas 2 selon l'indicateur de qualité IGD

NObj / Algorithmes	2	3	4
NSGAII	5.0	9.0	5.0
FastPGA	11.0	12.0	12.0
SPEA2	9.0	11.0	9.0
PESAII	7.0	7.0	8.0
IBEA	8.0	10.0	7.0
MOCell	4.0	8.0	6.0
AbYSS	6.0	4.0	3.0
SMPSO	1.0	1.0	1.0
OMOPSO	12.0	2.0	2.0
MOSA	13.0	13.0	13.0
AMOSA	3.0	5.0	11.0
MOTS	10.0	6.0	10.0
PAES	2.0	3.0	4.0

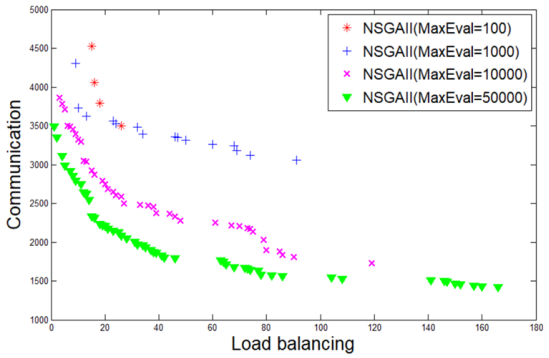


Table1 : temps d'exécution de NSGAII

MaxEval=100	MaxEval=1000	MaxEval=10000	MaxEval=50000
0,005s	0,027s	0,213s	0,968s

(a) Variation du nombre d'évaluations ($MaxEval$) de NSGAII [1]

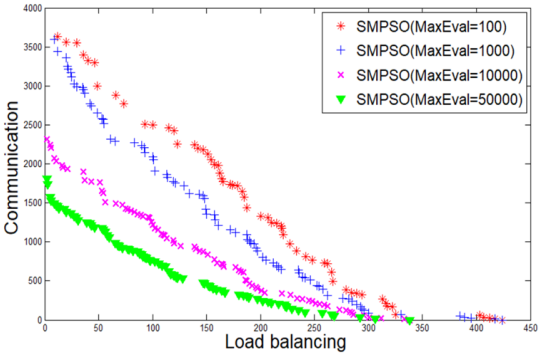


Table2 : temps d'exécution de SMPSO

MaxEval=100	MaxEval=1000	MaxEval=10000	MaxEval=50000
0,208s	1,839s	18,049s	90,118s

(b) Variation du nombre d'évaluations ($MaxEval$) de SMPSO [47]

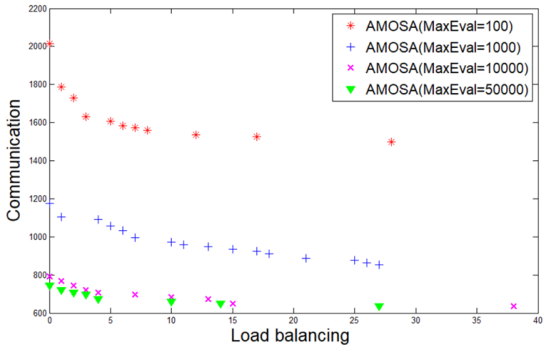


Table3 : temps d'exécution d'AMOSA

MaxEval=100	MaxEval=1000	MaxEval=10000	MaxEval=50000
0,040s	0,301s	2,877s	15,774s

(c) Variation du nombre d'évaluations ($MaxEval$) de AMOSA [6]

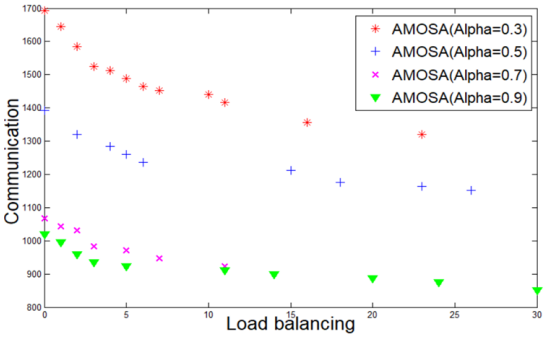
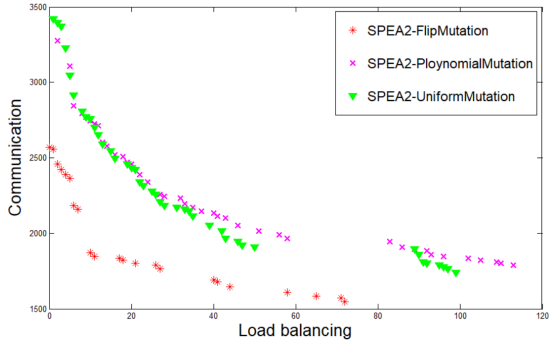


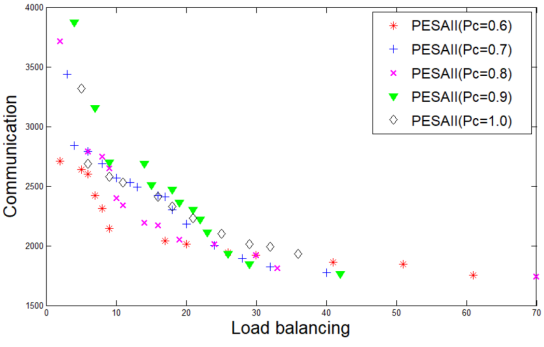
Table4 : temps d'exécution d'AMOSA

Alpha=0.3	Alpha=0.5	Alpha=0.7	Alpha=0.9
0,071s	0,103s	0,191s	0,611s

(d) Variation de taux de refroidissement (α) de AMOSA [6]



(e) Variation des types d'opérateurs de mutation de SPEA2 [2]



(f) Variation des probabilités de croisement (P_c) de PESAII [3]

FIGURE 3.9 – Effet de paramétrage des algorithmes sur la qualité des solutions retournées

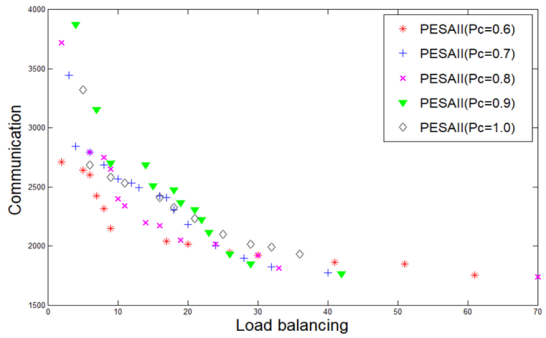


Table5 : IGD. Moyenne et écart type

Pc=0.6	Pc=0.7	Pc=0.8	Pc=0.9	Pc=1.0
8.31e-02$_{2.4e-02}$	9.30e-02$_{2.9e-02}$	8.73e-02$_{2.2e-02}$	8.60e-02$_{2.4e-02}$	9.00e-02$_{2.2e-02}$

(a) Variation des probabilités de croisement (Pc) de PESAII [3] pour résoudre le problème du mapping P4 donné en TABLE 3.2

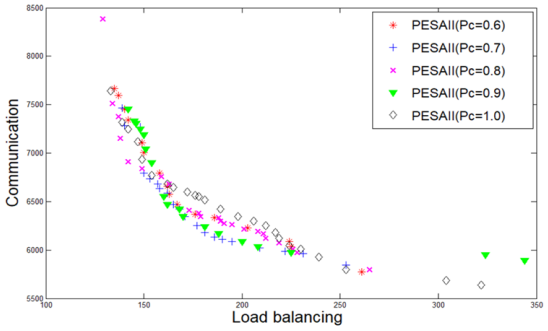


Table6 : IGD. Moyenne et écart type

Pc=0.6	Pc=0.7	Pc=0.8	Pc=0.9	Pc=1.0
4.82e-02$_{1.1e-02}$	4.57e-02$_{7.8e-03}$	4.68e-02$_{1.1e-02}$	**4.52e-02**$_{1.1e-02}$	4.70e-02$_{9.4e-03}$

(b) Variation des probabilités de croisement (Pc) de PESAII [3] pour résoudre le problème du mapping P6 donné en TABLE 3.2

FIGURE 3.10 – Un exemple illustrant le cas 4

Chapitre 4

Modèle de simulation

4.1 Introduction

Pour évaluer une solution de placement (mapping) donnée, deux modèles peuvent être utilisés : (1) le modèle analytique et (2) le modèle par simulation. Ce dernier a la capacité de représenter et d'analyser le comportement d'un système dans les détails, et ainsi capturer ses effets dynamiques tel que les contentions. A cet effet, nous avons proposé en plus des modèles analytiques présentés en chapitre 3, un modèle de simulation permettant d'évaluer le coût d'un mapping.

Dans ce chapitre, nous allons d'abord présenter le modèle d'architecture considéré pour notre cas d'étude. Ensuite, décrire le modèle de simulation proposé et comment ce dernier est utilisé pour l'estimation de deux exemples de métriques de performances à savoir : le temps d'exécution et la consommation d'énergie.

4.2 Modèle d'architecture considéré

Le modèle d'architecture comprend les principales caractéristiques de la plateforme cible qui exécutera une application donnée. Le modèle considéré dans notre cas d'étude consiste en un ensemble de processeurs (ou cores) hétérogènes caractérisés par : un identificateur (Id), un type, une fréquence et une position (x, y). Chaque processeur est relié à un routeur via l'interface réseau (NI). Soit **nœud** un élément composé d'un processeur (PE), d'une mémoire (M) et d'un routeur (R) comme montré en Figure 4.2. Chaque nœud est

interconnecté au réseau sur puce (NoC) dont les caractéristiques sont décrites comme suit :

- La topologie : décrit la disposition des nœuds constituant un réseau sur puce. Nous avons considéré les topologies 2D-Mesh et 2D-Torus (avec N×N nœuds).
- Le mode de commutation : la commutation par paquets en particulier Wormhole (WH) est utilisée pour notre cas d'étude.
- La technique de routage : l'algorithme de routage décrit comment les flits constituant un paquet sont routés vers leur nœud de destination. Nous avons utilisé le protocole de routage XY à cette fin.
- La technique de contrôle de flux : nous avons considéré le protocole basé crédits d'émission comme technique de contrôle de flux. Le principe de ce protocole est de permettre la transmission de données uniquement lorsqu'il y a un espace de données libre dans le tampon de destination. En d'autres termes, la valeur du crédit du tampon source est supérieur à 0. Dans notre cas, ce protocole est utilisé entre l'interface réseau (NI) et le routeur, à l'intérieur du routeur (entre buffer d'entrée et sortie) et inter-routeurs.
- La technique d'arbitrage : il arrive qu'à un instant donné, plusieurs flits provenant de différents ports d'entrées sollicitent le même port de sortie ; on parle dans ce cas de contention. Afin d' y remédier, plusieurs techniques d'arbitrage ont été proposées dont les principales sont décrites en section 1.3.2.3 (chapitre 1). Pour notre cas, nous avons considéré l'arbitrage à priorité tournante (Tourniquet ou Round Robin).
- Architecture du routeur considéré : définit l'organisation interne du routeur. L'architecture du routeur supposée est similaire à celle utilisée dans [10] donnée en Figure 4.1, où chaque routeur comporte :
  - Cinq ports d'entrée/sortie : Nord (N), Sud (S), Est (E), Ouest (W), et Local (L). Ce dernier établit une communication entre le routeur et son processeur (ou core) local, les autres ports sont connectés aux routeurs voisins. Chaque port contient un tampon (buffer) de type FIFO servant à stocker les messages (les flits) qui ne pourront pas être transmis tout de suite.
  - Une matrice de commutation : connectant les ports d'entrée aux ports de sortie.
  - Un bloc logique (bloc Arbitrage/routage) : servant à mettre en œuvre les poli-

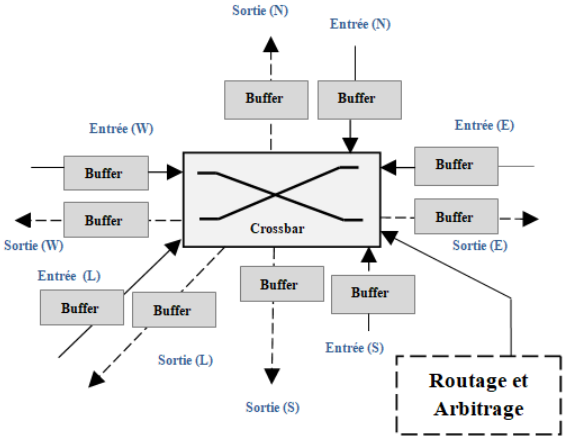


FIGURE 4.1 – Modèle du routeur utilisé

tiques d'arbitrage et de routage.

On suppose que tous les liens (i.e. entre PEs et routeurs et les liens inter-routeurs) sont bidirectionnels. La figure 4.2 montre un exemple d'un modèle d'architecture constitué de 3 types de processeurs interconnectés avec un réseau sur puce 3x3 2D-Torus.

4.3 Modèle de simulation proposé

Suivant le modèle d'architecture présenté ci-dessus, un modèle de simulation est proposé. Ce dernier est basé sur des évènements discrets, dont le principe général est décrit dans ce qui suit.

4.3.1 Modèle de simulation à événements discrets

Comme dans tous les modèles de simulation à évènements discrets, une liste d'évènements est utilisée pour stocker les évènements du système dans un ordre chronologique. Chaque évènement se produit à un instant donné. La simulation consiste à extraire et traiter les évènements un par un jusqu'à ce que la liste des évènements devient vide (i.e. les tâches d'une application soient terminées) (voir Algorithme 2). Chaque évènement traité génère d'autres évènements qui seront insérés dans la liste des évènements. Il est à noter que l'horloge de simulation ($currentTime$) est avancée au temps du prochain évènement.

Algorithme 2 Principe du modèle de simulation proposé

Soit $List_{event}$: liste des évènements du système considéré
Soit NB_{TT} : nombre de tâches terminées initialisé à 0
Soit NB_{TTotal} : nombre total de tâches d'une application donnée
Soit $currentTime$: le temps courant initialisé à 0

```
Déterminer les R_list (voir Algorithme 4)
pour chaque processeur PE_j faire
  si R_list(PE_j) n'est pas vide alors
    T_i = R_list(PE_j).peek()[1]
    Schedule(ExécuterTâche(T_i, PE_j), currentTime)
  fin si
fin pour
tant que (NB_TT < NB_TTotal) faire
  tant que (List_event n'est pas vide) faire
    event = List_event.peek()[1]
    si (event.getWhen()[2] > currentTime) alors
      next_time = event.getWhen()
      break
    fin si
    event.run()
    List_event.remove(event)[3]
  fin tant que
  currentTime = next_time
fin tant que
T_total = currentTime
```

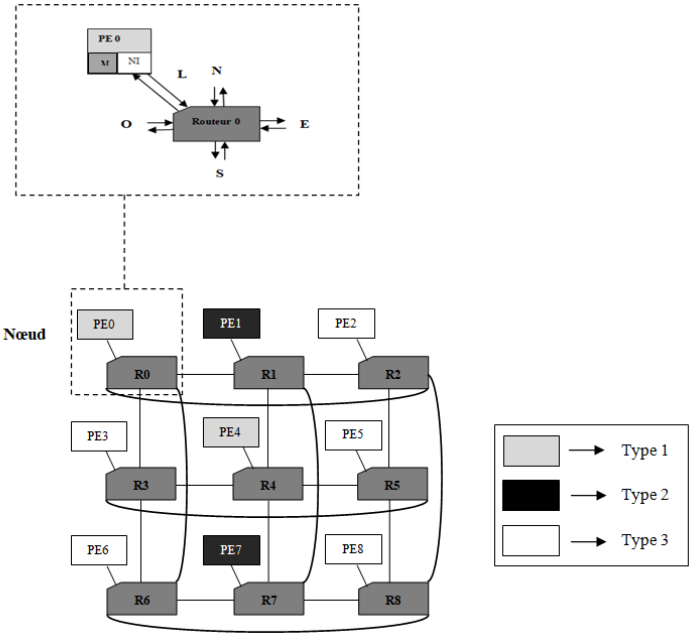


FIGURE 4.2 – Exemple de modèle d'architecture

Après avoir présenté le principe du modèle de simulation proposé, nous allons dans ce qui suit décrire les principaux événements le constituant.

4.3.2 Les événements du modèle de simulation proposé

Les principaux événements constituant le modèle de simulation proposé sont décrits comme suit :

- Évènement 1. $Exécuter Tâche(T_i, PE_j)$: la simulation commence par l'exécution des tâches prêtes de chaque processeur (ready tasks), comme décrit dans les algorithmes 3 et 4. Où $ST(T_i, PE_j)$ et $FT(T_i, PE_j)$ représentent respectivement le temps de début et de fin d'exécution de la tâche T_i sur un processeur PE_j, $FT(T_k, PE_j)$ est le temps de fin de la dernière tâche exécutée sur le processeur PE_j, C_{ij} est le temps d'exécution de la tâche T_i sur un processeur PE_j, *delta* est le temps d'attente pour lancer la tâche T_i sur le processeur PE_j, x_{ij} est une variable de décision définie

1. *peek*() : extraire le premier élément de la liste, l'élément récupéré n'est pas supprimé
2. event.getWhen() : renvoie le moment où l'événement (event) s'est produit
3. *remove*(l) : supprimer l'élément l de la liste

Algorithme 3 $Exécuter Tâche(T_i, PE_j)$

si $(PE_j.state = FREE)$ **alors**
  $ST(T_i, PE_j) = currentTime$
  $T_i.state = ASSIGNED$
  $PE_j.state = BUSY$
  $FT(T_i, PE_j) = ST(T_i, PE_j) + C_{ij}$
  $Schedule(GénérerPaquets(T_i, PE_j), FT(T_i, PE_j))$
  $R_{list}(PE_j).remove(T_i)$[3]
sinon
  delta= $FT(T_k, PE_j) - currentTime$
  $Schedule(ExécuterTâche(T_i, PE_j), currentTime$+delta)
fin si

comme suit :

$$x_{ij} = \begin{cases} 1 & \text{si la tâche } t_i \text{ est affectée au processeur } PE_j \\ 0 & \text{sinon} \end{cases}$$

$R_{list}(PE_j)$ la liste des tâches prêtes à être exécutées sur le processeur PE_j calculée suivant l'Algorithme 4, $nb(T_i.pred())$ est le nombre de communications entrantes de la tâche T_i.

Algorithme 4 Déterminer les R_{list}

Soit T_{list} : une liste contenant les tâches de l'application
pour chaque tâche $T_i \in T_{list}$ **faire**
  $T_i.state = INIT$
  si $(nb(T_i.pred()) = 0$ and $x_{i,j} = 1)$ **alors**
    $R_{list}(PE_j).add(T_i)$
    $T_i.state = READY$
  fin si
fin pour

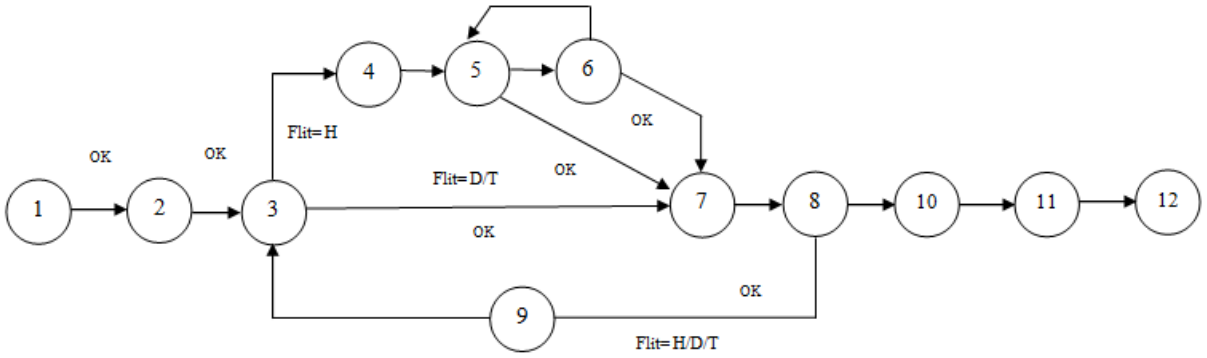
- Évènement 2. $GénérerPaquets(T_i, PE_j)$: une fois que la tâche T_i(ayant des successeurs qui ne sont pas affectés sur le processeur PE_j), la génération des paquets se fait au niveau de l'interface réseau (NI) du processeur PE_j. Chaque paquet est divisé en un ensemble de flits, principalement flit en-tête contenant les informations de routage, flits data contenant la charge utile, et flit queue pour indiquer la fin du paquet (voir Algorithme 5). $T_i.succ()$ et $T_k.pred()$ renvoient la liste des tâches successeurs et prédécesseurs de T_i et T_k respectivement, et $T_{GénérerPaquets}$ est le temps de génération de paquets (flits). La Figure 4.3 montre les changements d'états

de ces derniers durant leur transit via le NoC.

Algorithme 5 $GénérerPaquets(T_i, PE_j)$

Soit j : l'indice du processeur PE_j
  $T_i.state = FINISHED$
  $PE_j.state = FREE$
  si $(nb(T_i.succ()) > 0)$ **alors**
    pour chaque e_i $|Src(e_i) = T_i$, $Snk(e_i) = T_k$ **faire**
      si (T_i et T_k ne sont pas affectées au même processeur) **alors**
        Effectuer la génération de paquets (flits)
        pour tous les flits **faire**
          $ChangerEtat(flit, 1)$
        fin pour
        Schedule $(TransfertFlits_{NI\_routeur}(j),\ FT(T_i, PE_j) + T_{GénérerPaquets})$
      sinon
        $nb(T_k.pred())^{--}$
        si $(nb(T_k.pred()) == 0)$ **alors**
          $R_{list}(PE_j).add(T_k)$
        fin si
      fin si
    fin pour
  fin si
  si ($R_{list}(PE_j)$ n'est pas vide) **alors**
    $Schedule(ExécuterTâche(T_i, PE_j), currentTime)$
  fin si

- Évènement 3. $TransfertFlits_{NI\_routeur}(j)$: les flits composant un paquet seront envoyés flit par flit de l'interface réseau (NI) du processeur PE_j vers le routeur j via le lien les reliant (voir la Figure 4.4), et cela si le buffer d'entrée de ce dernier n'est pas plein (la valeur du crédit au niveau du $BENI$ est supérieure à 0). D'où l'état du flit passant de 1 à 2 (voir la Figure 4.3). $T_{NI-Router}$ est le temps de traversée du lien de l'interface réseau (NI) vers le routeur et δ est le temps qui sépare la transmission de deux flits successifs (voir Algorithme 8 en Annexe B). Il est à noter que la taille des buffers (entrée et sortie) au niveau de l'interface réseau (NI) est considérée illimitée.
- Évènement 4. $FlitArriveBufferEntree(j, direction_{Entree}, flit)$: après la traversée du lien reliant l'interface réseau du processeur source et le routeur, le flit arrive au buffer d'entrée de ce dernier. Par conséquent, son état passe de 2 à 3 (voir la Figure 4.3 et l'Algorithme 9 en Annexe B).



1 : flit au niveau de l'interface réseau (NI) du processeur source

2 : flit traverse lien (NI-routeur)

3 : flit au niveau du buffer d'entrée du routeur

4 : calculer route (flit entête)

5 : flit après routage

6 : Appliquer arbitrage (flit entête)

7 : flit traverse la matrice de commutation (Switch)

8 : flit au niveau du buffer de sortie du routeur

9 : flit traverse lien (inter-routeur)

10 : flit traverse lien (routeur-NI)

11 : flit au niveau de l'interface réseau (NI) du processeur destination

12 : fin

Ok : crédit > 0

NOK : crédit =0

H : flit en-tête, **D** : flits données, **T :** flit queue

FIGURE 4.3 – changement d'états des flits

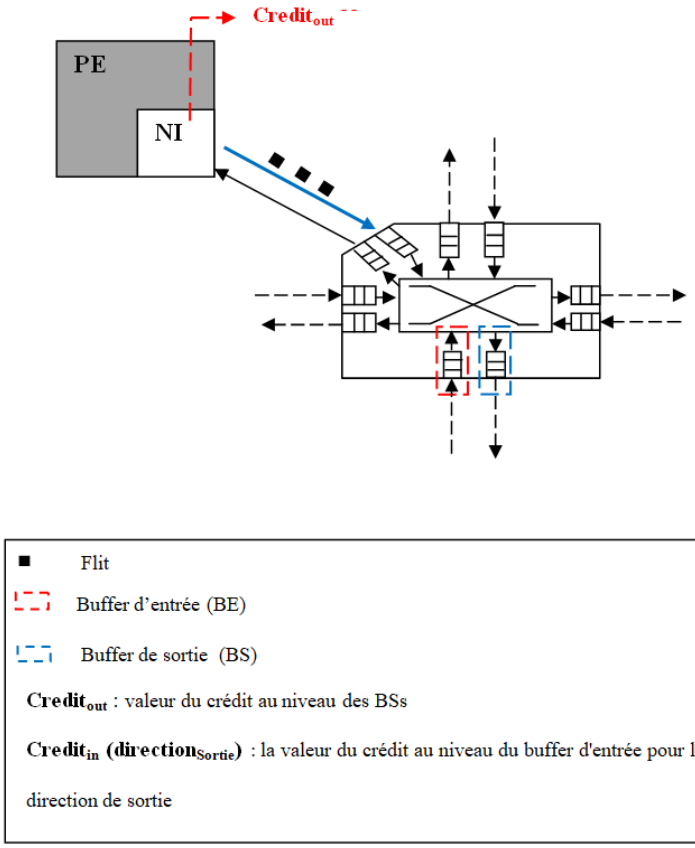
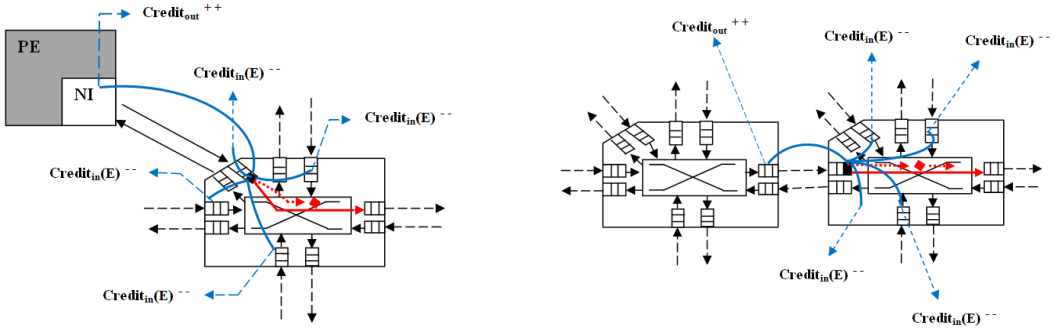


FIGURE 4.4 – Transfert flits de NI vers le routeur



(a) (b)

FIGURE 4.5 – Flit traverse la matrice de commutation (Switch) : (a) TraverserSwitch (j, Local, Est). (b) TraverserSwitch (j, West, Est).

- Évènement 5. $CalculerRoute(j, direction_{Entree})$: une fois que le flit d'entête arrive au tampon d'entrée du routeur, le saut suivant est calculé selon le protocole de routage supposé. $T_{calculRoute}$ est le temps requis pour le calcul de route pour un flit donné (voir Algorithme 10 en Annexe B).
- Évènement 6. $AppliquerArbitrage(j, direction_{sortie})$: si plusieurs flits d'en-tête sollicitent le même port de sortie, la politique d'arbitrage est appliquée pour sélectionner l'un d'eux (voir Algorithme 11 en Annexe B). Il est à noter que dans le cas ou il n'y a pas de concurrence d'accès au buffer de sortie (pas de contention), $PS.direction_{Gagnante}$ = la direction du seul buffer d'entrée sollicitant le port de sortie PS. $T_{Arbitrage}$ est le temps requis pour l'arbitrage pour un flit donné.
- Évènement 7. $TraverserSwitch(j, direction_{Entree}, direction_{Sortie})$: la totalité du paquet dont le flit d'en-tête est gagnant sera envoyée flit par flit à travers la matrice de commutation (Switch), et cela suivant la politique de contrôle de flux établie (voir la Figure 4.5). Pour notre cas, la politique de contrôle de flux basée crédit est utilisée. Par conséquent, la traversée aura lieu si la valeur du crédit au niveau du buffer d'entrée pour une direction de sortie donnée est supérieure à 0 (voir Algorithme 12 en Annexe B). $T_{TraverseSwitch}$ est le temps de traversée de la matrice de commutation (Switch), et ω représente le temps qui sépare la transmission de deux flits successifs.
- Évènement 8. $FlitArriveBufferSortie(j, direction_{Sortie}, flit)$: une fois qu'un flit

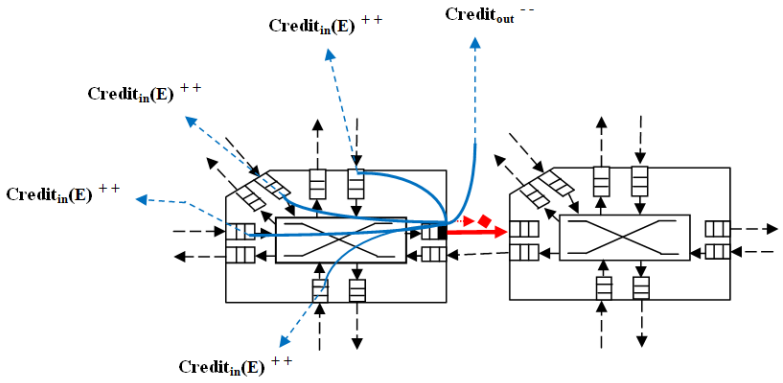


FIGURE 4.6 – Traverser le lien inter-routeurs

donné arrive au buffer de sortie, son état passe de l'état 7 à l'état 8 (voir la Figure 4.3 et l'Algorithme 13 en Annexe B).

- Évènement 9. $TraverserLien_{InterRouteurs}(j, direction_{Sortie})$: les flits sont transmis entre des routeurs voisins (source et destination) si la valeur du crédit au niveau du tampon de sortie d'un routeur source est supérieur à 0 (voir la Figure 4.6 et l'algorithme 14 en Annexe B). $T_{Traversee_{InterRouteurs}}$ est le temps de traversée du lien reliant deux routeurs (source et destination), et Φ est le temps qui sépare la transmission de deux flits successifs.
- Évènement 10. $TransfertFlits_{(Routeur-NI)}(j)$: le flit se trouvant dans le buffer local d'un routeur donné sera envoyé vers le processeur relié à ce dernier (voir Algorithme 15 en Annexe B). $T_{Traversee_{Routeur-NI}}$ est le temps de traversée du lien reliant un routeur à l'interface réseau (NI), et Ω est le temps qui sépare la transmission de deux flits successifs.

  Évènement 11. $FlitArriveBufferEntree_{NI}(j, flit)$: au final, le flit arrive au niveau de l'interface réseau du processeur destination, où une phase de contrôle de l'arrivée des paquets (flits) et la constitution du message initial auront lieu. L'état du flit passe de 10 à 11 (voir la Figure 4.3 et l'Algorithme 16 en Annexe B).

La fonction $ChangerEtat(flit, NouvelEtat)$ utilisée dans la totalité des événements sert à changer l'état du flit vers un nouvel état, ainsi qu'à sauvegarder les informations de l'état précédent. Pour notre cas, les informations sauvegardées sont le délai et l'énergie du

flit. Le détail de cette fonction est donné dans l'annexe B (Algorithme 17).

Dans ce qui suit, nous allons expliquer comment ce modèle est utilisé pour l'estimation de deux métriques de performance à savoir : le temps d'exécution total et la consommation d'énergie.

4.4 Estimation des métriques de performance avec le modèle de simulation proposé

4.4.1 Temps d'exécution total (Schedule Length)

Le temps d'exécution total d'une application donné correspond au temps écoulé entre l'exécution du premier évènement et l'exécution du dernier donné par T_{total} (voir l'Algorithme 2). L'avantage de ce modèle est qu'il prend en considération le temps d'attente engendré par les accès simultanés aux ressources partagés (ex. les ports de sortie des routeurs).

4.4.2 Consommation d'énergie

En plus du temps d'exécution, pour la même simulation, ce modèle renvoie la consommation d'énergie totale. Cette dernière est calculée comme la somme de la consommation d'énergie des composants du système, qui est l'énergie de traitement (c.à.d. l'énergie des processeurs) et l'énergie de communication (c.à.d. l'énergie des composants du NoC). L'avantage d'utiliser ce modèle vient de sa capacité de prendre en compte l'énergie de consommation des tampons (buffers) des routeurs en présence de contentions. Le détail du calcul de l'energie est donné par l'Algorithme 6.

4.5 Conclusion

Dans ce chapitre, nous avons présenté dans un premier temps le modèle de simulation proposé suivant les caractéristiques de l'architecture considérée décrite en section 4.2, par la suite, nous avons présenté comment ce dernier est utilisé pour l'estimation de deux fonctions objectives à savoir le temps d'exécution et l'énergie de consommation. Ce modèle proposé est facilement extensible : nous offrons au concepteur un moyen facile d'ajouter de nouvelles

Algorithme 6 Calcul de l'énergie

$E_{total} = 0$
pour chaque *communication* e_i **faire**
  pour chaque *paquet* p_i de la communication e_i **faire**
    pour chaque *flit* du paquet p_i **faire**
      pour chaque *etat* e_j **faire**
        $flitEnergie_{total} = flitEnergie_{total} + flit.getFlitEnergieParEtat(e_j)$ {voir Algorithme 17 en Annexe B}
      fin pour
      $E_{comm} = E_{comm} + flitEnergie_{total}$
    fin pour
  fin pour
fin pour
$E_{total} = E_p + E_{comm}$ {Pour le calcul de E_p se référer à la section 3.2.3.1 (chapitre 3)}

topologies, politiques de routage, techniques de commutation, mécanismes d'arbitrage, techniques de contrôle de flux et/ou de définir une autre architecture du routeur.

Dans le chapitre qui suit, ce modèle sera utilisé lors de l'exploration des solutions du mapping. En plus de cela, nous allons présenter une nouvelle série de métaheuristiques que nous avons proposées afin d'enrichir notre outil d'exploration présenté au chapitre 3. Ces dernières consistent en une combinaison d'une suite de métaheuristiques déjà présentes dans notre outil, donnant naissance à de nouvelles métaheuristiques communément appelées métaheuristiques hybrides.

Chapitre 5

Hybridation de métaheuristiques pour la résolution du problème de placement (Mapping)

5.1 Introduction

Ce présent chapitre est consacré à la description d'une nouvelle classe de méthodes de résolution proposée dans le cadre de cette thèse pour la résolution du problème de placement (Mapping). Il s'agit de combiner les deux classes de métaheuristiques à savoir : les P-métaheuristiques et les S-métaheuristiques présentées précédemment en chapitre 3. La suite de méthodes résultantes est appelée métaheuristiques hybrides.

Ce chapitre est organisé comme suit : dans la section 5.2, nous présentons le concept de l'hybridation ainsi que les diverses classifications des méthodes hybrides trouvées dans la littérature en particulier celle proposée par E-G.Talbi [19][13]. Dans la section 5.3, nous détaillons l'approche d'hybridation proposée ainsi que son évaluation expérimentale. Nous terminons ce chapitre avec une discussion des résultats en section 5.4, et une conclusion en section 5.5.

5.2 Concepts sur l'hybridation

Durant ces dernières années, un grand intérêt est porté sur l'utilisation des méthodes hybrides dans la résolution des problèmes du monde réel. Cela est dû à leurs preuves

d'efficacité dans la résolution de ces derniers. Cette hybridation consiste à faire coopérer plusieurs méthodes de résolution afin de combiner leurs avantages. Plusieurs types de coopération existent telles que : les métaheuristiques avec des méthodes exactes, les métaheuristiques avec data mining, les métaheuristiques avec les réseaux de neurones, les métaheuristiques avec métaheuristiques, etc. Ce dernier type d'hybridation (métaheuristiques avec des métaheuristiques) est utilisé dans le cadre de notre travail, où les avantages respectifs des P-métaheuristiques et des S-métaheuristiques ont été combinés. Avant de présenter les méthodes hybrides proposées, nous allons d'abord donner un aperçu sur la classification des méthodes hybrides donnée dans la littérature.

5.2.1 classification des méthodes hybrides

Plusieurs classifications des méthodes hybrides ont été proposées dans la littérature [19][174][175][176]. Dans cette section, nous allons présenter celle donnée par E-G.Talbi [19][13]. Selon E-G.Talbi, l'hybridation de métaheuristiques se distinguent en deux classifications principales : (1) classification à plat et (2) classification hiérarchique. Nous nous focalisons seulement sur cette dernière classification (classification hiérarchique) dans la suite de cette section.

5.2.1.1 classification hiérarchique

La classification hiérarchique donnée par E-G.Talbi [19][13] est montrée en Figure 5.1. Cette dernière est caractérisée selon le niveau ainsi que selon le mode de coopération comme suit :

- Selon le niveau : deux types d'hybridations se distinguent : (1) hybridation de Bas niveau (Low level) et (2) hybridation de Haut niveau (High level). L'hybridation de Bas niveau consiste à remplacer une fonction donnée d'une métaheuristique (par exemple : les opérateurs de reproduction) par une autre métaheuristique. Dans les algorithmes hybrides de Haut niveau, les différentes métaheuristiques sont autonomes. Il n'y a pas de relation directe avec le fonctionnement interne d'une métaheuristique.
- Selon le mode : pour chaque niveau cité précédemment (Bas ou Haut niveau), deux modes différents de coopération se distinguent à savoir : (1) mode relais et (2)

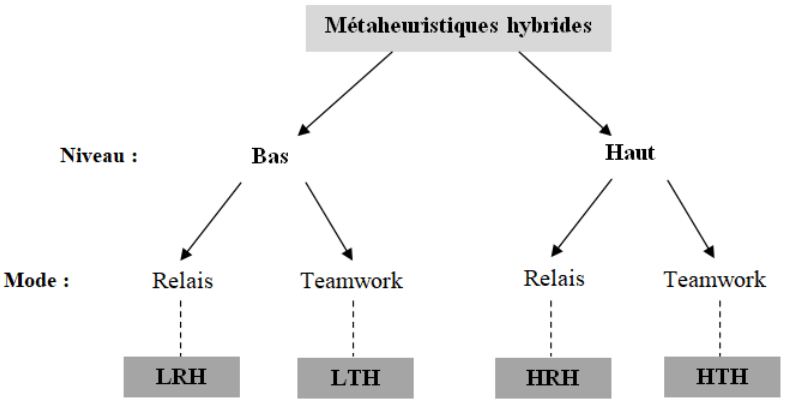


FIGURE 5.1 – Classification hiérarchique des métaheuristiques hybrides [19][13]

mode teamwork. Dans l'hybridation par relais, un ensemble de métaheuristiques est appliqué l'une après l'autre ; chacune utilisant la sortie de la précédente comme son entrée et agissant de manière pipeline. l'hybridation par teamwork représente des modèles d'optimisation coopératifs dans lesquels de nombreux agents coopérants évoluent en parallèle ; chaque agent effectue une recherche dans un espace de solution.

Quatre classes sont dérivées de cette taxonomie hiérarchique :

1. Coopération Bas niveau - mode relais (LRH)
2. Coopération Bas niveau - mode teamwork (LTH)
3. Coopération Haut niveau - mode relais (HRH)
4. Coopération Haut niveau - mode teamwork (HTH)

Suivant la taxonomie (classification hiérarchique) donnée ci-dessus, l'approche hybride proposée dans le cadre de notre travail est de type coopération Haut niveau - mode relais (HRH).

5.3 Approche hybride proposée

Les métaheuristiques s'imposent comme des approches très appropriées pour traiter les problèmes multiobjectifs (MOPs) et trouver de bonnes solutions de compromis [13]. Ces dernières sont classées en deux grandes catégories : (1) métaheuristiques basées sur une

solution unique (S-métaheuristiques), et (2) métaheuristiques basées sur une population de solutions (P-métaheuristiques). Comme présenté précédemment en chapitre 2, les S-métaheuristiques partent d'une seule solution en essayant de trouver d'autres meilleures solutions dans son voisinage (recherche locale), donc ce type de métaheuristiques se concentrent sur l'exploitation. D'autre part, les P-métaheuristiques commencent avec un ensemble de solutions appelée population. La puissance des P-métaheuristiques est leur capacité d'explorer un vaste espace de recherche, permettant ainsi à ces métaheuristiques de se concentrer sur l'exploration. Dans le cadre de notre thèse, nous avons tenté de combiner les avantages de ces deux classes de métaheuristiques, et ainsi donner de nouvelles méthodes hybrides bien équilibrées en termes d'exploitation et d'exploration. Les détails des schémas de coopération de ces méthodes seront donnés dans ce qui suit.

5.3.1 Hybridation des algorithmes évolutionnaires NSGAII [1], SPEA2 [2], PESAII [3], FastPGA [4] et IBEA [5] avec AMOSA [6]

Le principe de l'approche d'hybridation proposée est donné en Figure 5.2. Elle est divisée en deux phases principales :

- Phase 1 : dans cette première phase, un ensemble d'algorithmes évolutionnaires multiobjectifs (MOEAs) bien connus, dont NSGAII [1], SPEA2 [2], PESAII [3], FastPGA [4] et IBEA [5] ont été utilisés afin de générer des solutions du mapping prometteuses (un compromis de solutions de mapping).
- Phase 2 : dans la seconde phase, les solutions obtenues lors de la première phase en plus d'un ensemble de solutions générées de manière aléatoire sont utilisées pour initialiser l'archive d'AMOSA, et cela dans le but d'améliorer le front de Pareto (solutions compromis) retourné par les P-metaheuristiques. Les solutions générées aléatoirement ont pour rôle d'assurer la diversité des solutions résultantes.

Ces nouvelles métaheuristiques hybrides sont nommées suivant le nom de la métaheuristique utilisée dans la première phase précédée d'un H, qui signifie Hybrid. Par exemple, l'algorithme hybride nommé HNSGAII, donné par l'algorithme 7, utilise l'algorithme AMOSA afin d'améliorer davantage les solutions de compromis générées par l'algorithme standard NSGAII [1].

Algorithme 7 HNSGAII-main

Etape 1 : exécuter NSGAII
Soit **S** : l'ensemble contenant les solutions non-dominées retournées par NSGAII
Soit **R** : l'ensemble contenant les solutions non-dominées générées aléatoirement
Soit **A** : l'archive de AMOSA
Soit **HL** : la taille maximale de l'archive
Ajouter toutes les solutions de **S** à **A**
pour chaque solution $r \in R$ **faire**
  si $(\exists\, l \in A,\, l \prec r)$ **alors**
    Supprimer r
  sinon
    Ajouter r à **A**
  fin si
fin pour
si (**A**.$size()$ >**HL**) **alors**
  Appliquer la clusterisation (clustering)
fin si
Etape 2 : exécuter AMOSA

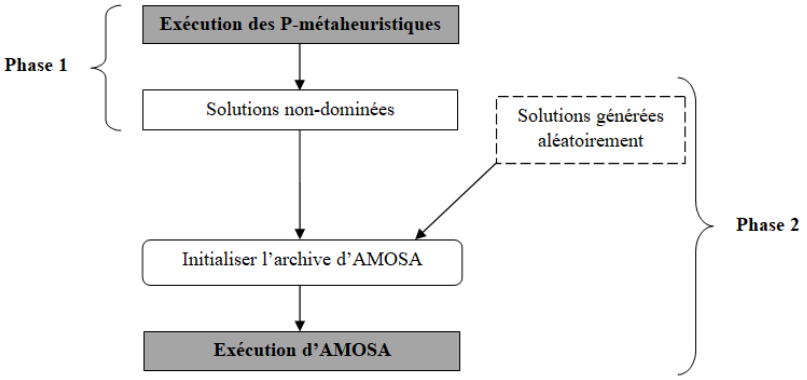


FIGURE 5.2 – Principe de l'approche d'hybridation proposée

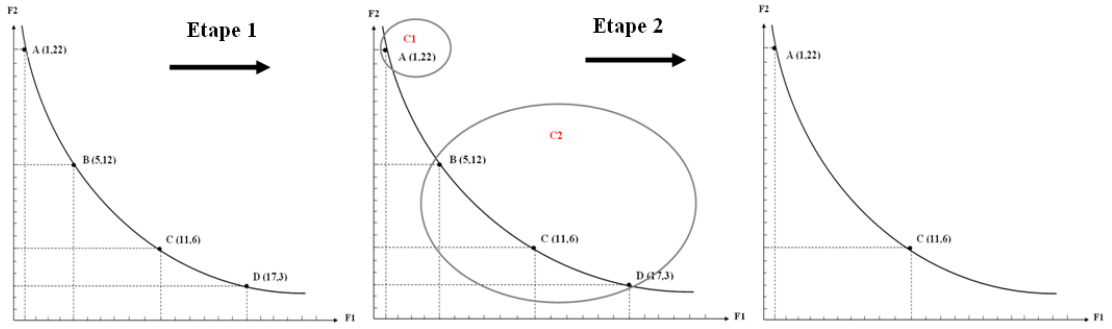


FIGURE 5.3 – Principe de clusterisation

La fonction de clusterisation sert à réduire le nombre de solutions à HL ainsi qu'à renforcer explicitement la diversité des solution non dominées [6]. Le principe de cette fonction est montré en Figure 5.3, où la clusterisation est décrite en deux étapes principales comme suit :

- Étape 1 : construire HL clusters, dans cet exemple la valeur de HL=2 ;
- Étape 2 : choisir un représentant de chaque cluster et omettre les autres solutions, dans cet exemple, le point "C" est le point représentant du cluster C2 (B, C, D).

5.3.2 Évaluation expérimentale de l'approche proposée

Dans cette section, les algorithmes hybrides proposés ont été évalués en considérant divers instances du problème du mapping (petite, moyenne et grande). Ces instances diffèrent l'une de l'autre selon la taille du graphe de tâches et de la plateforme utilisés, comme le montre la TABLE 5.1. TGFF [173] est utilisé pour générer les graphes de tâches aléatoirement, et cela en variant le nombre de noeuds (tâches), d'autre part le modèle d'architecture (Platformes) utilisé dans ces expériences se compose de k types de processeurs interconnectés en utilisant un réseau sur puce (NoC). Les caractéristiques de ce réseau sont résumées dans la TABLE 5.2.

Nous avons évalué les algorithmes hybrides proposés suivant : (1) leur qualité de solutions de Pareto retournées, et (2) leur temps de calcul requis (runtime). Afin de mesurer la qualité des solutions retournées, deux indicateurs de qualité ont été utilisés à savoir : Epsilon et IGD définis précédemment en section 2.2.4. Il est à noter que plus les valeurs

TABLE 5.1 – Exemples d'instances du problème de mapping

problème du mapping	Graphes de tâches GTs	Plateformes
Petit	6 tâches	3x3 2D-torus
Moyen	10 tâches	4x4 2D-torus
Grand	100 tâches	8x8 2D-torus

d'Epsilon et d'IGD d'une métaheuristique donnée sont petites, plus l'approximation du front de Pareto retourné est proche du front de Pareto de référence (c.à.d. meilleure convergence). Pour l'indicateur IGD, cette petite valeur signifie aussi une bonne diversité des solutions obtenues. Lorsque les valeurs IGD et Epsilon sont égales à 0, cela implique que toutes les solutions générées par un algorithme donné sont dans l'ensemble optimal de Pareto du problème. Ces métriques appliquées nécessitent le calcul d'un ensemble de Pareto optimal, à cette fin, l'ensemble de référence est calculé comme suit :

- Pour de petits et moyens problèmes de mapping, le front de référence (front optimal) est celui retourné par la méthode exacte implémentée nommée Multiobjective Branch & Bound (MBB).
- Pour de grands problèmes du mapping, le front de Pareto optimal est obtenu en recueillant (concaténant) les résultats de plusieurs exécutions des différents algorithmes.

Pour chacune de nos expériences, 30 runs ont été effectués, et les TABLES 5.4, 5.5, 5.6, 5.7, B.5 et B.6 représentent les informations statistiques sur l'indicateur de qualité appliqué ainsi que le temps d'exécution des algorithmes. La TABLE 5.3 donne le paramétrage des algorithmes utilisés dans la suite de nos expériences. Ces derniers ont été choisis après une phase primaire où une analyse de sensibilité est effectuée pour tenter de trouver les paramètres qui donnent de bons résultats pour les trois instances du problème du mapping, et cela en termes de qualité des solutions et de leur durée d'exécution. Il est à noter qu'une seule variable à la fois est variée et les autres restent fixes. Cette méthode est appelée dans la littérature One-at-time (OAT) [177]. Tous les calculs ont été effectués sur un PC Intel(R) Core(TM) i7 CPU, 2.7GHz, avec 8 Go de RAM. Au cours de toutes ces expériences, nous avons spécifié deux fonctions de coût à optimiser à savoir : le temps d'exécution (Schedule Length) et l'énergie de consommation. Ces deux dernières fonctions ont été définies en utilisant un modèle de simulation défini en section 4.3 (chapitre 4).

TABLE 5.2 – Paramètres du NoC

Topologie	2D-torus
Technique de commutation	Wormhole (WH)
Technique de routage	XY
Technique d'arbitrage	Round Robin (RR)
Technique de contrôle de flux	credit-based

TABLE 5.3 – Paramétrage des algorithmes

NSGAII [1]/ FastPGA [4]	
Taille de la population	100
Max Evaluation	10000
Probabilité de mutation (pm)	1.0/L (L : longeur du chromosome)
Probabilité de croisement (pc)	0.9
SPEA2 [2]/ PESAII [3]/ IBEA [5]	
Taille de la population	100
Taille de l'archive	100
Max Evaluation	10000
Probabilité de mutation (pm)	1.0/L (L : longeur du chromosome)
Probabilité de croisement (pc)	0.9
AMOSA	
Température initiale (T_0)	800
Température finale (T_1)	10^{-3}
Taux de refroidissement (α)	0.9
Max Iterations	100
HL	100
SL	110
Gamma	1.8

TABLE 5.4 – Indices statistiques. IGD. algorithmes hybrides vs algorithmes non hybrides (petite instance du mapping)

IGD / Algorithmes	Moyenne	Écart-Type	Médiane	Min	Max	Temps d'exécution
NSGAII	$5.72e-04$	$5.8e-04$	$4.89e-04$	$0.00e+00$	$1.84e-03$	$4,759$s
HNSGAII	$1.42e-04$	$3.6e-04$	$0.00e+00$	$0.00e+00$	$1.83e-03$	$10,683$s
SPEA2	$1.55e-03$	$9.8e-04$	$1.46e-03$	$7.91e-04$	$6.79e-03$	$5,132$s
HSPEA2	$1.83e-04$	$3.8e-04$	$0.00e+00$	$0.00e+00$	$1.56e-03$	$11,127$s
PESAII	$4.66e-03$	$2.4e-03$	$3.13e-03$	$1.21e-03$	$8.98e-03$	$4,140$s
HPESAII	$1.81e-03$	$2.0e-03$	$5.18e-04$	$0.00e+00$	$5.19e-03$	$10,171$s
FastPGA	$1.74e-03$	$1.0e-03$	$1.54e-03$	$5.72e-04$	$6.79e-03$	$7,938$s
HFastPGA	$1.07e-03$	$1.6e-03$	$4.89e-04$	$0.00e+00$	$5.01e-03$	$13,933$s
IBEA	$1.15e-02$	$2.2e-03$	$1.15e-02$	$7.37e-03$	$1.79e-02$	$0,169$s
HIBEA	$3.50e-03$	$2.3e-03$	$5.01e-03$	$0.00e+00$	$8.01e-03$	$6,093$s
MBB	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$43,663$s

TABLE 5.5 – Indices statistiques. Epsilon. algorithmes hybrides vs algorithmes non hybrides (petite instance du mapping)

Epsilon / Algorithmes	Moyenne	Écart-Type	Médiane	Min	Max	Temps d'exécution
NSGAII	$3.53e+00$	$5.1e+00$	$1.00e+00$	$0.00e+00$	$1.40e+01$	$4,759$s
HNSGAII	$6.33e-01$	$2.5e+00$	$0.00e+00$	$0.00e+00$	$1.40e+01$	$10,683$s
SPEA2	$5.83e+00$	$8.6e-01$	$6.00e+00$	$5.00e+00$	$9.00e+00$	$5,132$s
HSPEA2	$9.00e-01$	$2.0e+00$	$0.00e+00$	$0.00e+00$	$8.00e+00$	$11,127$s
PESAII	$2.56e+01$	$3.6e+01$	$1.40e+01$	$5.00e+00$	$1.68e+02$	$4,140$s
HPESAII	$3.87e+00$	$5.1e+00$	$2.00e+00$	$0.00e+00$	$2.10e+01$	$10,171$s
FastPGA	$6.90e+00$	$1.5e+00$	$6.00e+00$	$5.00e+00$	$9.00e+00$	$7,938$s
HFastPGA	$2.03e+00$	$2.0e+00$	$1.00e+00$	$0.00e+00$	$6.00e+00$	$13,933$s
IBEA	$1.86e+02$	$6.2e+01$	$18.85e+01$	$9.1e+01$	$3,06e+02$	$0,169$s
HIBEA	$6.73e+00$	$1.5e+01$	$4.00e+00$	$0.00e+00$	$8.40e+01$	$6,093$s
MBB	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$43,663$s

TABLE 5.6 – Indices statistiques. IGD. algorithmes hybrides vs algorithmes non hybrides (moyenne instance du mapping)

IGD / Algorithmes	Moyenne	Écart-Type	Médiane	Min	Max	Temps d'exécution
NSGAII	$1.28e-03$	$4.6e-04$	$1.38e-03$	$0.00e+00$	$2.55e-03$	$0,584$m
HNSGAII	$3.65e-04$	$5.5e-04$	$0.00e+00$	$0.00e+00$	$1.38e-03$	$1,322$m
SPEA2	$1.98e-03$	$4.9e-04$	$1.83e-03$	$1.38e-03$	$3.21e-03$	$0,574$m
HSPEA2	$9.93e-04$	$5.6e-04$	$1.22e-03$	$0.00e+00$	$1.64e-03$	$1,315$m
PESAII	$4.36e-03$	$3.8e-03$	$2.78e-03$	$8.96e-04$	$1.70e-02$	$0,559$m
HPESAII	$9.53e-04$	$6.5e-04$	$1.24e-03$	$0.00e+00$	$1.87e-03$	$1,328$m
FastPGA	$2.65e-03$	$6.5e-04$	$2.49e-03$	$1.58e-03$	$4.26e-03$	$0,597$m
HFastPGA	$6.12e-04$	$6.3e-04$	$4.06e-03$	$0.00e+00$	$1.56e-03$	$1,344$m
IBEA	$1.24e-02$	$3.4e-03$	$1.21e-02$	$6.82e-03$	$2.08e-02$	$0,018$m
HIBEA	$8.89e-04$	$7.5e-04$	$1.16e-03$	$0.00e+00$	$2.96e-03$	$0,770$m
MBB	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$27,065$m

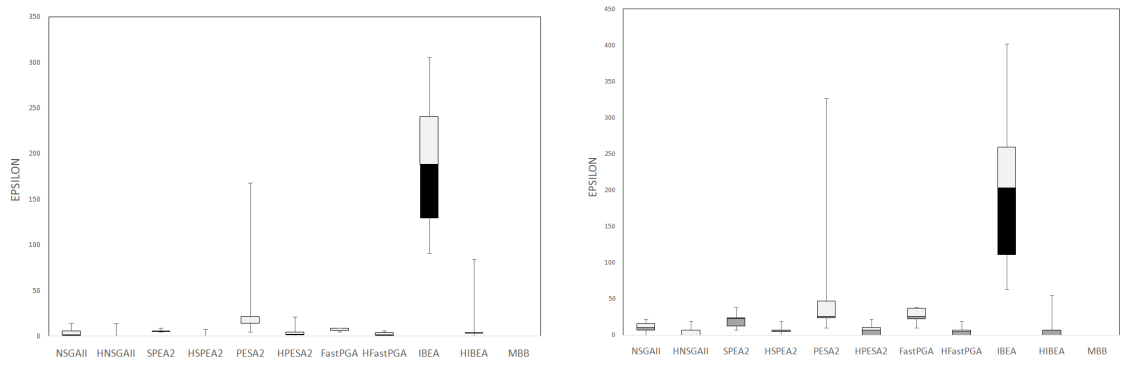
TABLE 5.7 – Indices statistiques. Epsilon. algorithmes hybrides vs algorithmes non hybrides (moyenne instance du mapping)

Epsilon / Algorithmes	Moyenne	Écart-Type	Médiane	Min	Max	Temps d'exécution
NSGAII	$1.13e+01$	$5.9e+00$	$1.05e+01$	$0.00e+00$	$2.20e+01$	$0,584$m
HNSGAII	$3.00e+00$	$5.2e+00$	$0.00e+00$	$0.00e+00$	$1.90e+01$	$1,322$m
SPEA2	$2.09e+01$	$8.3e+00$	$2.35e+01$	$7.00e+00$	$3.80e+01$	$0,574$m
HSPEA2	$6.47e+00$	$4.5e+00$	$7.0e+00$	$0.00e+00$	$1.90e+01$	$1,315$m
PESAII	$6.01e+01$	$8.1e+01$	$2.60e+01$	$1,00e+01$	$3.27e+02$	$0,559$m
HPESAII	$7.07e+00$	$6.1e+00$	$7.00e+00$	$0.00e+00$	$2.20e+01$	$1,328$m
FastPGA	$2.63e+01$	$8.6e+00$	$2.60e+01$	$1.00e+01$	$3.80e+01$	$0,597$m
HFastPGA	$4.47e+00$	$4.9e+00$	$5.00e+00$	$0.00e+00$	$1.90e+01$	$1,344$m
IBEA	$2.03e+02$	$9.1e+01$	$2.03e+02$	$6.30e+01$	$4.02e+02$	$0,018$m
HIBEA	$6.67e+00$	$1.0e+01$	$7.00e+00$	$0.00e+00$	$5.50e+01$	$0,770$m
MBB	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$27,065$m

Les TABLES 5.4, 5.5, 5.6 et 5.7 montrent respectivement une étude comparative entre les algorithmes hybrides proposés (HNSGAII, HSPEA2, HPESAII, HFastPGA et HIBEA) et les algorithmes non hybrides (NSGAII, SPEA2, PESAII, FastPGA et IBEA), et cela pour résoudre un exemple d'une petite et d'une moyenne instance du problème du mapping données en TABLE 5.1. Les fronts retournés par les algorithmes sont comparés avec l'ensemble de Pareto optimal retourné par la méthode exacte implémentée Multiobjective Branch & Bound (MBB).

Sur les TABLES 5.4, 5.5, 5.6 et 5.7, nous observons premièrement que les algorithmes hybrides donnent de meilleurs résultats (en termes d'IGD et d'Epsilon) par rapport aux algorithmes non hybrides sur les deux instances du problème du mapping considérés (petite et moyenne). Pour plus de clarté sur ces améliorations, les boxplots résultants de ces expériences sont donnés en Figure 5.4 , et cela en terme de l'indicateur de qualité Epsilon. Il est très important de souligner qu'à l'exception de l'algorithme HIBEA qui donne de meilleurs résultats sur tous les runs par rapport à IBEA (voir Figure 5.4), ce n'est cependant pas le cas pour d'autres algorithmes hybrides comme illustrés en TABLES B.1, B.2, B.3 et B.4, où les résultats des 30 runs comparant entre les algorithmes hybrides et non hybrides selon l'indicateur de qualité Epsilon ont été donnés d'une manière explicite. Deuxièmement, nous constatons que les résultats des algorithmes proposés sont identiques ou proches de ceux retournés par la méthode exacte (MBB), et cela dans un délai (temps d'exécution) raisonnable. A titre d'exemple, plus de 75% des runs de HNSGAII donnent le front optimal (le même que celui retourné par MBB), tandis que moins de 50% des runs l'atteint en exécutant le NSGAII standard (voir Figure 5.4). Ceci confirme l'optimalité des résultats retournés par les algorithmes hybrides proposés pour la résolution des problèmes du mapping de petite et de moyenne taille. Par conséquent, nous pouvons les utiliser pour la résolution de grandes instances du problèmes du mapping.

La Figure 5.5 montre le résultat d'une étude comparative entre les algorithmes hybrides et non hybrides, et cela pour résoudre une grande instance du problème du mapping (voir TABLE 5.1). Dans cette expérience, un graphe de 100 tâches a été placé sur une plateforme MPSoC hétérogène contenant six types de processeurs interconnectés en utilisant la topologie 2D-Torus (8x8). Les graphes résultants montrés en Figure 5.5 représentent



(a) (b)

FIGURE 5.4 – Epsilon. algorithmes hybrides vs non hybrides : (a) petite instance du problème du mapping. (b) moyenne instance du problème du mapping

les solutions non dominées du mapping pour chaque algorithmes après 30 exécutions. Les détails statistiques de cette expérience sont donnés en TABLES B.5, B.6 et également en Figure B.1.

D'après les TABLES (5.5, B.5 et B.6) et la FIGURE B.1, nous constatons que les algorithmes hybrides proposés donnent des résultats promoteurs (de meilleurs fronts) par rapport aux algorithmes non hybrides en considérant les deux indicateurs de qualité (IGD et epsilon), et cela au détriment du temps. Ce temps supplémentaire des algorithmes hybrides par rapport aux algorithmes non hybrides dans ces expériences est dû au temps d'exécution d'AMOSA utilisé pour l'amélioration des fronts de Pareto résultants des métaheuristiques standard. Il est important de noter que les paramètres des algorithmes proposés sont une combinaison de ceux donnés en TABLE 5.3 (avec les mêmes valeurs de configuration).

D'autres expériences ont été effectuées afin d'étudier la sensibilité des algorithmes hybrides proposés à leurs paramètres d'entrée. Les figures 5.6 (a) et 5.6 (b) présentent deux expériences sur l'effet de certains paramètres comme : (1) nombre d'itération ($MaxIter$) et (2) le taux de refroidissement (α) de AMOSA sur un exemple d'algorithme hybride proposé, soit HNSGAII. Nous avons varié les valeurs de $MaxIter$ (50, 100 et 500) dans la première expérience et celles de α (0.5, 0.7 et 0.9) dans la seconde expérience. Les autres valeurs de paramètres ont été fixées comme montré en TABLE 5.8. Une grande instance du mapping a été utilisée dans cette expérience.

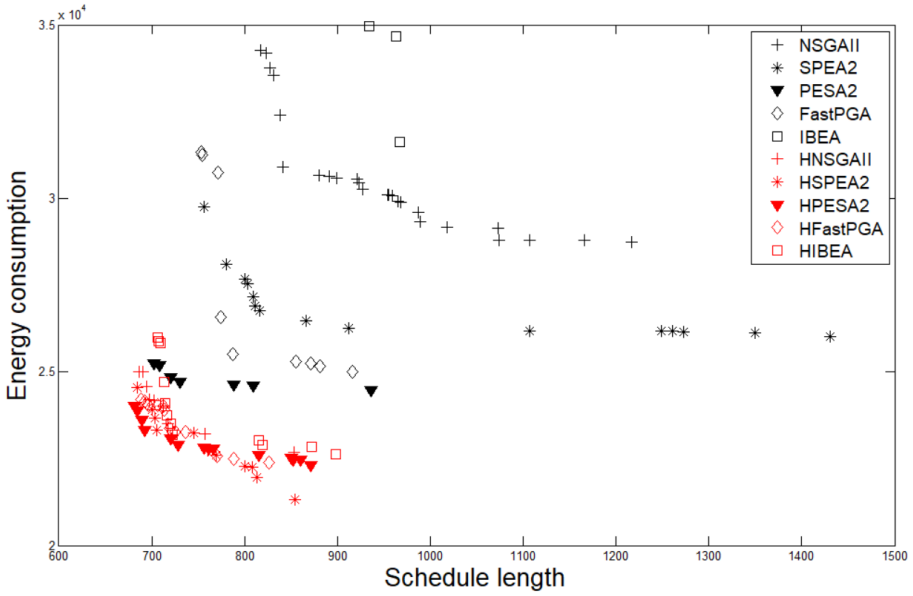


FIGURE 5.5 – Placement d'un graphe de 100 tâches généré aléatoirement sur une plateforme hétérogène avec six types de processeurs interconnectés en utilisant une topologie 2D-Torus (8x8). Deux fonctions de coût sont optimisées (temps d'exécution et énergie de consommation)

D'après les résultats montrés en figures 5.6 (a) et 5.6 (b), on observe clairement que HNSGAII est très sensible à ses paramètres d'entrée. Le choix de ces paramètres détermine fortement la qualité des solutions de placement (mapping) résultantes. Par exemple, la figure 5.6 (a) montre que plus la valeur de $(MaxIter - AMOSA)$ est grande, meilleurs sont les résultats retournés, et cela au détriment du temps (voir TABLE1 de la Figure 5.6 (a)). D'où un compromis entre la qualité des solutions et la durée d'exécution requise doit être prise en compte lors de l'exploration d'un problème du mapping donné.

Comme dernière étude expérimentale, nous avons tenté de comparer entre un exemple parmi les algorithmes hybrides proposés : soit HNSGAII avec l'algorithme non hybride NSGAII (les paramètres choisis pour NSGAII favorisent sa performance maximale). La configuration de ces deux algorithmes est donnée en TABLE B.7. Deux fonctions de coût ont été spécifiées : (1) l'équilibrage de charge, (2) la communication (définies avec des modèles analytiques). Une grande instance du mapping est utilisée dans cette expérience (voir TABLE 5.1). La Figure 5.7 ainsi que les TABLES B.8 et B.9 donnent les résultats de cette expérience. D'après ces résultats, on constate que HNSGAII montrent de meilleurs

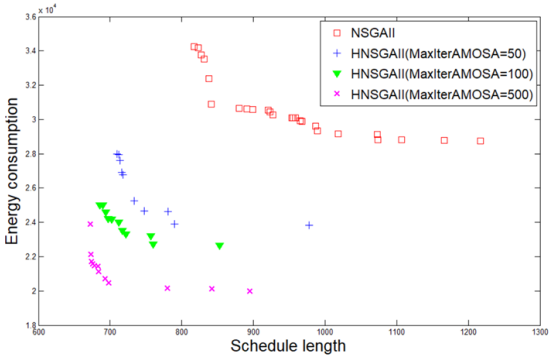


Table 1: temps d'exécution de HNSGAII en minutes

MaxIterAMOSA =50	MaxIterAMOSA =100	MaxIterAMOSA =500
4,819m	6,354m	18,093m

(a)

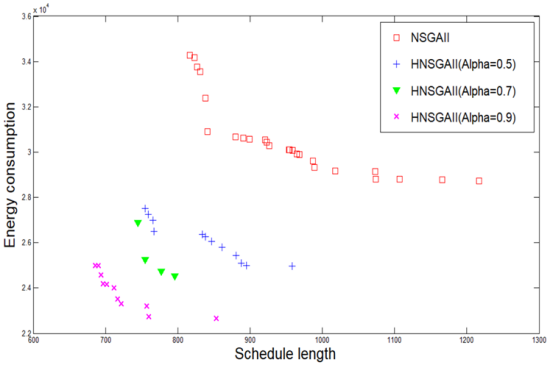


Table 2: temps d'exécution de HNSGAII en minutes

Alpha = 0.5	Alpha = 0.7	Alpha = 0.9
3,683m	4,140m	6,354m

(b)

FIGURE 5.6 – Étude de la sensibilité des paramètres d'un exemple d'algorithmes hybrides proposés (HNSGAII) : (a) variation de $MaxIter - AMOSA$ de HNSGAII, (b) variation du taux de refroidissement (α) de HNSGAII.

TABLE 5.8 – Paramétrage de l'algorithme HNSGAII

HNSGAII	
Taille de la population	100
Max Evaluation (NSGAII)	10000
Probabilité de mutation (pm)	1.0/L (L : longeur du chromosome)
Probabilité de croisement (pc)	0.9
Température initiale (T_0)	800
Température finale (T_1)	10^{-3}
Taux de refroidissement (α)	0.9 (**0.5**, **0.7**)
Max Iterations (AMOSA)	100 (**50**, **500**)
HL	100
SL	110
Gamma	1.8

performances en termes de qualité de solutions et du temps d'exécution comparé avec NSGAII suivant les paramètres donnés en TABLE B.7. Ceci confirme davantage la robustesse des méthodes hybrides proposées.

5.4 Discussion

Les résultats expérimentaux obtenus montrés ci-dessus prouvent la performance de notre proposition. Ce qui signifie que le choix de ces nouvelles approches hybrides dans le flot de conception d'un système donné peut diminuer son coût de mise en œuvre et avoir un bon impact sur son comportement final, puisque ces approches donnent souvent des compromis de haute qualité par rapport à ceux retournés par les approches standards. Cependant, ces

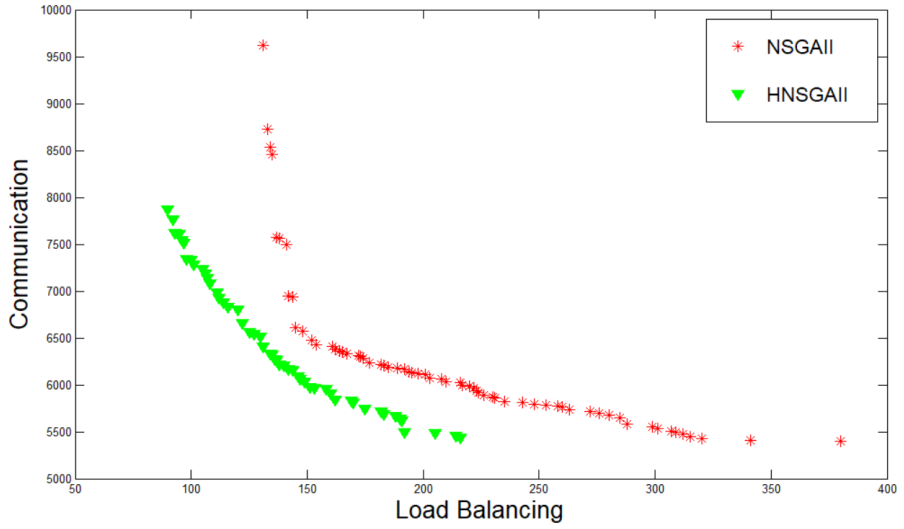


FIGURE 5.7 – Placement d'un graphe de 100 tâches généré aléatoirement sur une plateforme hétérogène avec six types de processeurs interconnectés en utilisant une topologie Torus 8x8. Deux fonctions de coût sont optimisées (Load balancing et Communication)

algorithmes hybrides proposés sont très sensibles à leurs paramètres puisqu'ils combinent deux ensembles de métaheuristiques qui sont tous deux sensibles à leurs paramètres. Pour cela, une première phase expérimentale, au cours de laquelle une analyse de sensibilité pour chaque paramètre de chaque algorithme hybride doit être effectuée afin de déterminer les réglages les plus appropriés pour traiter un type de problème de mapping donné.

5.5 Conclusion

A l'issue de ce chapitre, nous avons proposé une approche d'hybridation pour résoudre le problème de placement (mapping) des applications parallèles sur un MPSoC hétérogène basé NoC. Cette approche consiste à combiner deux classes de métaheuristiques à savoir : les S-métaheuristiques et les P-métaheuristiques. Pour prouver l'optimalité de ces méthodes hybrides résultantes, nous les avons comparées à la méthode exacte MBB dans le cas de la résolution de petites et de moyennes instances du problème du mapping. A partir des résultats expérimentaux donnés en section 5.3.2, les algorithmes hybrides proposés retournent les mêmes résultats (le même front) ou des résultats proches de ceux donnés par la méthode exacte (MBB) en un temps d'exécution très raisonnable comparé à la méthode

exacte. Pour le cas de la résolution de grandes instances du mapping, les approches hybrides proposées donnent de meilleurs résultats par rapport aux approches non hybrides, et cela en termes de qualité des solutions retournées. Ceci confirme que les S-métaheuristiques peuvent effectivement améliorer les solutions retournées pas les P-métaheuristiques.

Conclusion générale et perspectives

Cette thèse traite particulièrement l'une des phases cruciales de la conception des systèmes embarqués de haute performance. Cette phase est celle de placement (mapping) statique des applications parallèles sur un MPSoC hétérogène interconnecté avec un réseau sur puce. Ce problème est NP-difficile et multiobjectif. Afin de le traiter, nous avons proposé dans le cadre de cette thèse plusieurs approches de résolution à savoir : (1) des méthodes exactes (ex. Multiobjective Branch & Bound [51]) ; (2) des métaheuristiques basées sur une population de solutions appelées P-métaheuristiques (ex. NSGAII [1], SPEA2 [2], PESAII [3], FastPGA [4], IBEA [5], MOCell [44], AbYSS [46], OMOPSO [45], SMPSO [47]) et (3) des métaheuristiques basées sur une solution appelées S-métaheuristiques (ex. MOSA [49], AMOSA [6], MOTS [50] et PAES [109]). Nous avons également proposé (4) des métaheuristiques hybrides combinant entre ces deux classes de métaheuristiques (ex. HNSGAII, HSPEA2, HPESA2, HFastPGA, HIBEA, etc.).

À partir de nos résultats expérimentaux, des directives sur l'utilisation de plusieurs algorithmes multiobjectif suivant les caractéristiques du problème du mapping à traiter peuvent être données comme suit :

- Pour de petites (voir de moyennes) instances du problème du mapping, la méthode exacte (MBB) ainsi que les métaheuristiques "avec de bons paramètres" (P-métaheuristiques, S-métaheuristiques et métaheuristiques hybrides) donnent de bons compromis de solutions (c.à.d. de bons fronts de Pareto). La méthode exacte (MBB) donne le front Pareto exact utilisé comme ensemble de Pareto de référence. Donc, la méthode exacte (MBB) est préférable dans ce cas de figure.

- Pour de grandes instances du problème du mapping, seules les métaheuristiques à savoir : les P-métaheuristiques, les S-métaheuristiques et les métaheuristiques hybrides peuvent être utilisées. La méthode exacte (MBB) ne donne pas des résultats dans un temps polynomial en considérant un grand espace de recherche. Par conséquent, la méthode exacte (MBB) ne peut pas résoudre les grandes instances du problème de mapping.
- Bien que les métaheuristiques permettent de traiter toutes instances du problème du mapping (petite, moyenne ou grande), leur principal inconvénient est leur sensibilité aux paramètres. Donc, une analyse de sensibilité doit être faite afin d'estimer les bons paramètres pour chaque algorithme. Il est à noter qu'il n'existe pas de réglage optimal standard pour les métaheuristiques puisque cela est fortement lié au problème du mapping considéré.
- Aucune métaheuristique n'est meilleure qu'une autre pour tout type de problème du mapping. Cela dépend de plusieurs facteurs présentés en section 3.5.2 (chapitre 3).
- Les S-métaheuristiques, en particulier AMOSA [6] peuvent améliorer efficacement les P-métaheuristiques (prouvé en chapitre 5). L'inconvénient majeur de ces métaheuristiques hybrides résultantes est la difficulté de configurer leurs paramètres, puisque ceux-ci consistent en une combinaison de paramètres des métaheuristiques le composant qui sont, elles aussi par leur nature, très sensibles à leurs paramètres.
- Le temps d'exécution d'une méthode de résolution dépend de plusieurs facteurs tels que : la taille du problème (graphe de tâches et plateformes utilisés), le nombre d'objectifs à optimiser, la configuration de paramètres de l'algorithme, le type de fonction objective (analytique vs simulation).

Au terme de notre travail, nous prévoyons une suite de perspectives de recherche à savoir :

- Généraliser la solution de placement proposée dans cette thèse en placement simultané de plusieurs applications (critiques et/ou non critiques) sur la même plateforme cible.
- Sauvegarder les solutions générées statiquement et les utiliser dans les décisions du placement dynamique, donnant ainsi des solutions de placement hybrides (statiques-dynamiques).

Des extensions pour l'outil de placement proposé sont envisageables à savoir :

1. Combiner d'autres méthodes de résolution en plus de celles présentées en chapitre 5, à titre d'exemple : MOTS [50] avec d'autres P-métaheuristiques, des métaheuristiques avec d'autres, ou l'utilisation d'une méthode de recherche locale dans un exemple d'opérateur de reproduction d'une P-métaheuristiques, etc.
2. Implémenter des méthodes plus élaborées d'étude de sensibilité de paramètres des métaheuristiques en prenant en compte l'intéraction entre les différents paramètres d'un algorithme donné.

Annexe A

Le Framework jMetal

Introduction

Dans cette annexe, nous allons d'abord présenter d'une manière brève le framework jMetal sur lequel nous avons implémenté notre outil de mapping. Ensuite nous décrivons les changements (adaptation et extensions) que nous avons apportés pour résoudre notre problématique.

Le framework jMetal

JMetal «Java Metaheuristic Algorithms» est un framework orienté objet développé en Java pour l'optimisation multiobjectif avec les métaheuristiques. Les principales caractéristiques de ce framework sont :

- Il offre plusieurs algorithmes d'optimisation multiobjectif : NSGAII [1], SPEA2 [2], PAES [109], PESAII [3], OMOPSO [45], MOCell [44], AbYSS [46] , FastPGA [4], IBEA [5], SMPSO [47], etc. Dans notre thèse, nous avons adapté ces algorithmes, tout en tirant profit du code déjà implémenté (réutilisation du code).
- Il offre un ensemble d'indicateurs de qualité bien connus pour évaluer la performance des algorithmes multiobjectif tels que : hypervolume (HV), spread (Δ), distance générationnelle (GD), distance générationnelle inversée (IGD) et epsilon. Ces deux derniers ont été utilisés dans cette thèse.
- Il est caractérisé par sa simplicité ainsi que sa facilité d'utilisation.

- Il permet de réaliser des études expérimentales complètes et de générer automatiquement des informations statistiques. Ces dernières sont représentées de divers façons, à savoir :
  - Des fichiers LATEX contenant des tableaux résumant les informations des différents indicateurs de qualité appliqués.
  - Des scripts R pour produire des boxplots des résultats.
  - Des tableaux en latex contenant les valeurs du test de Friedman.
- Enfin, son avantage principal est son extensibilité et sa réutilisabilté.

L'outil de placement proposé dans cette thèse a été implémenté sous ce framework. Par conséquent, il hérite de toutes les caractéristiques citées ci-dessus.

Architecture de jMetal

JMetal a été conçu suivant une architecture orientée objet qui facilite l'intégration de nouveaux composants et la réutilisation de ceux existants. Il fournit à l'utilisateur un ensemble de classes de base pouvant être utilisées pour soit concevoir de nouveaux algorithmes, soit mettre en œuvre d'autres problèmes ou d'autres opérateurs plus complexes.

Dans cette section, on va décrire les principales classes composant ce framework. Son architecture est donnée en Figure A.1. Son principe de fonctionnement est : «qu'un algorithme résout un problème en utilisant une (et probablement plus) SolutionSet et un ensemble d'opérateurs». Une terminologie générique pour nommer les classes à l'intérieur de jMetal a été employée la rendant suffisamment générale pour être utilisé dans n'importe quelle métaheuristique, à titre d'exemple, pour les algorithmes évolutionnaires, les populations et les individus correspondent aux classes SolutionSet et Solution respectivement. Comme montré en Figure A.1, les principales classes composant ce framework sont :

- La classe Algorithm : est une classe abstraite qui doit être héritée par les métaheuristiques incluses dans le framework. Elle comporte un ensemble de méthodes, par exemple : la méthode execute() servant à exécuter l'algorithme et retourne SolutionSet comme résultat, les méthodes addOperator() et getOperator() pour incorporer et utiliser certains opérateurs. De même, certains paramètres spécifiques peuvent être ajoutés et accessibles en utilisant les deux méthodes respectives addParameter()

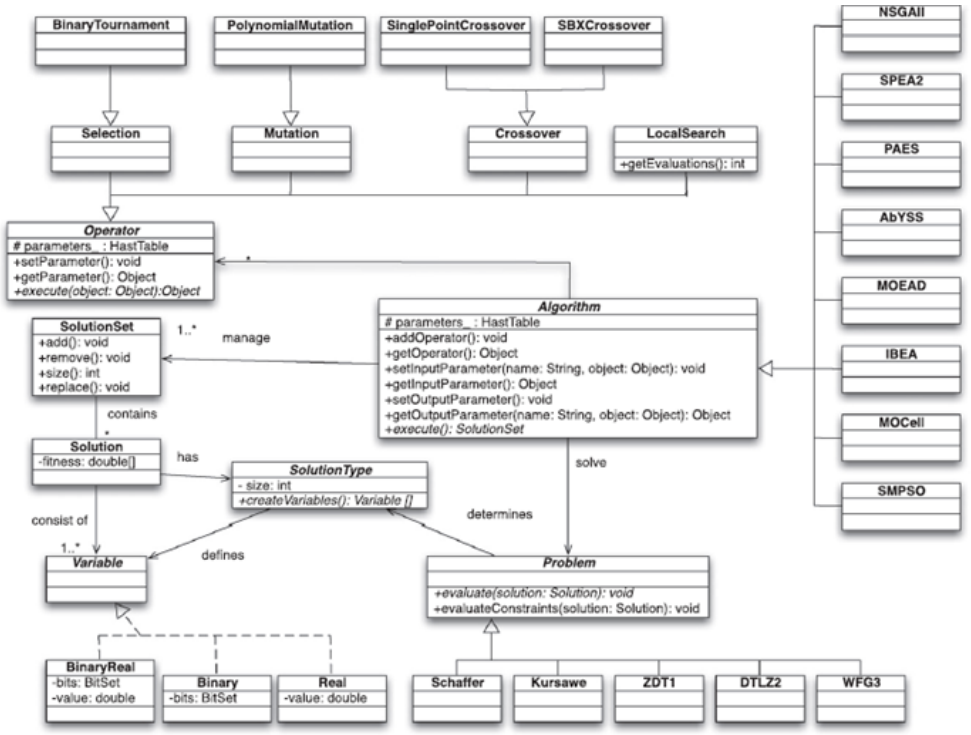


FIGURE A.1 – Architecture générale de jMetal [20]

et getParameter().

- La classe SolutionSet : comme son nom l'indique, la classe SolutionSet représente un ensemble d'objets Solution. Chaque objet Solution est associé à un type (la classe SolutionType). La classe SolutionType permet de définir les types de variables de la solution et les créer en utilisant la méthode createVariables().
- La classe Problem : dans jMetal, tous les problèmes héritent de la classe Problem. Cette classe contient deux méthodes de base : evaluate() et evaluateContraints(). Les deux méthodes reçoivent une Solution représentant une solution candidate au problème : la première l'évalue et la seconde détermine la violation des contraintes de cette solution. Tous les problèmes doivent définir la méthode evaluate() alors que seuls les problèmes ayant des contraintes doivent définir la méthode evaluateContraints().
- La classe Operator : la classe Operator est une superclasse destinée à représenter les opérateurs génériques à utiliser par les différents algorithmes, par exemple, l'opérateur de croisement, l'opérateur de mutation, l'opérateur de sélection, etc.

Pour une description plus détaillée de l'architecture de jMetal, le lecteur est invité à

consulter le manuel d'utilisation de jMetal [178], ainsi que d'autres références telles que [179][20].

Résoudre la problèmatique du mapping en utilisant le framework jMetal

Notre outil de mapping est implémenté sous le framework jMetal. Ce dernier a été adapté comme suit :

- Création de la classe appelé Mapping héritant de la classe Problem.
- Adaptation de plusieurs métaheuristiques présentes dans le framework jMetal pour la résolution de notre problème de placement et cela en (1) définissant un nouveau encodage de solution et (2) en adaptant divers opérateurs de reproduction (mutation et croisement).
- En plus des métaheuristiques données par jMetal, nous avons ajouté d'autres, telles que les S-metaheuristiques (MOSA [49], AMOSA [6], MOTS [50]), une méthode exacte Multiobjective Branch & Bound (MBB) [51] et les méthodes hybrides proposées dans cette thèse. Toutes ces métaheuristiques ajoutées héritent de la classe Algorithm.

Par analogie à l'architecture de jMetal, l'architecture de notre outil du mapping proposé est donnée en Figure A.2.

Conclusion

Dans cette annexe, une brève présentation du framework jMetal sur lequel notre outil du mapping a été implémenté est donnée. En plus du framework jMeta utilisé dans le cadre de cette thèse, d'autres frameworks pour l'optimisation multiobjectif existent dans la littérature. On cite à titre d'exemples : MOEA Framework [180], ParadisEO [181], PISA [182], etc.

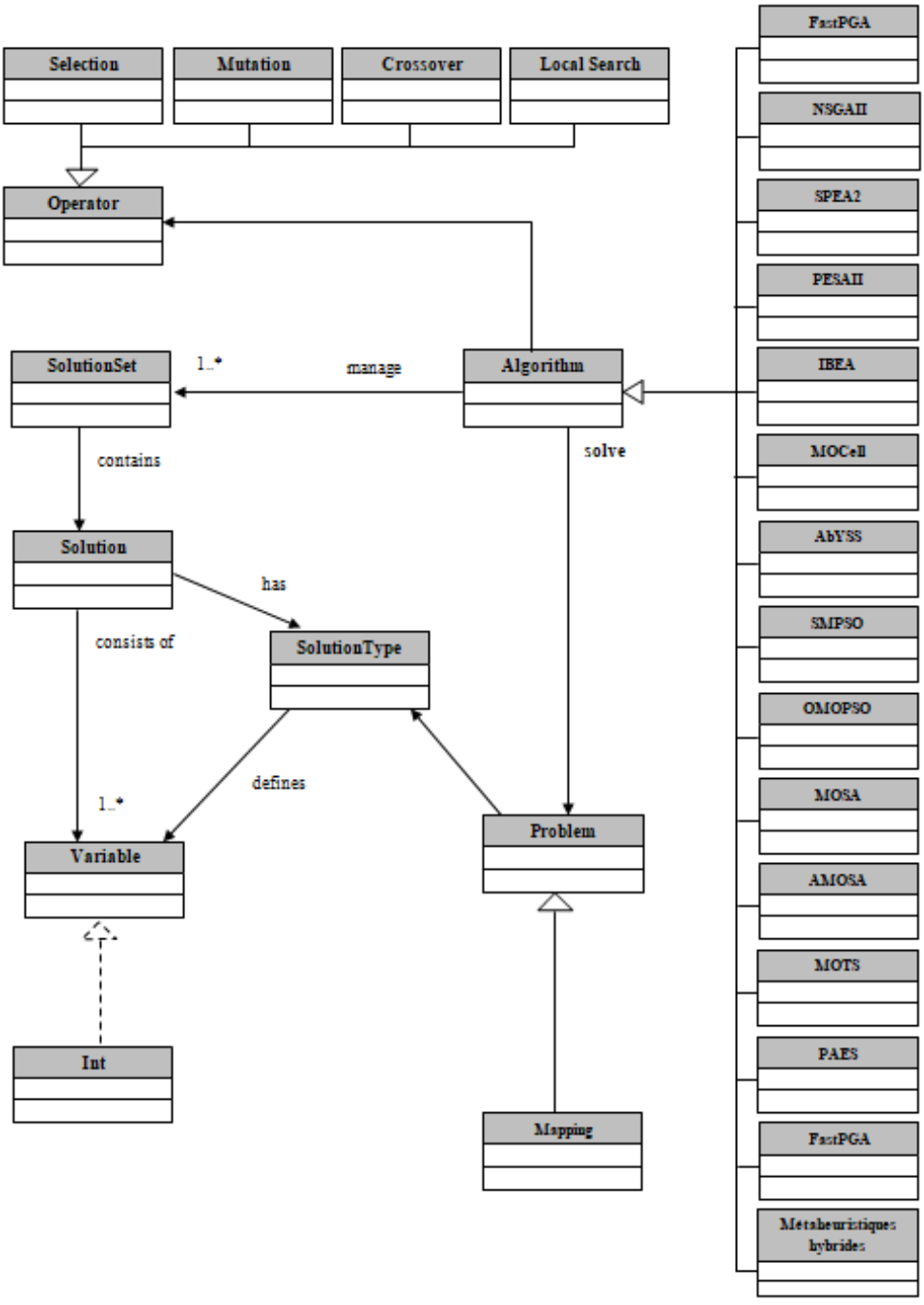


FIGURE A.2 – Architecture générale de l'outil de Mapping proposé

Annexe B

Algorithmes et suite des résultats expérimentaux des chapitres 4 et 5

Introduction

Dans cette seconde annexe, nous présentons des algorithmes détaillés relatifs au 4, ainsi que la suite des résultats expérimentaux du chapitre 5.

Algorithme 8 $TransfertFlits_{NI\_routeur}(j)$

Soit $BSNI$: le buffer de sortie de l'interface réseau (NI)
Soit nb_{BNI} : le nombre de flits au niveau du buffer de sortie de NI ($BSNI$)
Soit $credit_{out}$: la valeur du crédit au niveau du buffer de sortie de NI

```
si (credit_out > 0)&&(nb_BNI > 0) alors
    flit = BSNI.pop()[4]
    credit_out--
    ChangerEtat(flit, 2)
    Schedule (FlitArriveBufferEntree(j, Local, flit), currentTime + T_NI-Router)
    si (nb_BNI > 0) alors
        Schedule (TransfertFlits_NI_routeur(j), currentTime + δ)
    fin si
fin si
```

4. *pop*() : extraire le premier élément de la liste, l'élément récupéré est retiré de celle-ci

Algorithme 9 $FlitArriveBufferEntree(j, direction_{Entree}, flit)$

Soit $Routers$: liste des routeurs composant la plateforme
Soit BE : le buffer d'entrée du routeur j $|BE = Routers.get(j).getBE(direction_{Entree})$
Soit nb_{BE} : le nombre de flits au niveau du BE

```
BE.add(flit)
ChangerEtat(flit, 3)
si (nb_BE == 1) alors
  si (flit == en-tête) alors
    Schedule (CalculerRoute(j, direction_Entree), currentTime)
  sinon
    Schedule (TraverserSwitch(j, direction_Entree, BE.getdirectiondestination()), currentTime)
  fin si
fin si
```

Algorithme 10 $CalculerRoute(j, direction_{Entree})$

Soit BE : le buffer d'entrée du routeur j

```
flit = BE.peek()[1]
si (flit == en-tête et flit.etat == 3) alors
  Exécuter une politique de routage pour identifier la direction du port de sortie
  (direction_Sortie)
  ChangerEtat(flit, 4)
  flit.finRoutage = currentTime + T_calculRoute
  Schedule (AppliquerArbitrage(j, direction_Sortie), flit.finRoutage)
fin si
```

Algorithme 11 *AppliquerArbitrage*($j, direction_{Sortie}$)

Soit BE : le buffer d'entrée du routeur j
Soit PS : le port de sortie du routeur j $|PS = Routers.get(j).getPS(direction_{Sortie})$
Soit $PS.direction_{Gagnante}$: la direction du buffer d'entrée contenant le flit d'en-tête gagnant

pour chaque BE **faire**
  $flit = BE.peek()$[1]
  si ($flit.etat == 4$ et $currentTime == flit.finRoutage$) **alors**
    $ChangerEtat(flit, 5)$
  fin si
fin pour
si ($PS.direction_{Gagnante} == NULL$) et ($PS$ est sollicité par un seul flit (en-tête) dont $flit.etat == 5$) **alors**
  $PS.direction_{Gagnante}$ = la direction du buffer d'entrée du seul flit le sollicitant
  $Routers.get(j).getBE(PS.direction_{Gagnante}).setdirectiondestination(direction_{Sortie})$
  Schedule ($TraverserSwitch(j, PS.direction_{Gagnante}, direction_{Sortie})$, $currentTime$)
sinon
  si ($PS.direction_{Gagnante} == NULL$) et ($PS$ est sollicité par plusieurs *flits d'en-tête* provenant des entrées différentes) **alors**
    Appliquer une *politique d'arbitrage*
    pour tous les flits sollicitant PS **faire**
      $ChangerEtat(flit, 6)$
    fin pour
    $flit.finArbitrage = currentTime + T_{Arbitrage}$
    $PS.direction_{Gagnante}$ = la *direction* du buffer d'entrée du flit (en-tête) gagnant
    $Routers.get(j).getBE(PS.direction_{Gagnante}).setdirectiondestination(direction_{Sortie})$
    Schedule ($TraverserSwitch(j, PS.direction_{Gagnante}, direction_{Sortie})$, $flit.finArbitrage$)
  fin si
fin si

Algorithme 12 $TraverserSwitch(j, direction_{Entree}, direction_{Sortie})$

Soit PS : le port de sortie du routeur j
Soit BE : le buffer d'entrée du routeur j
Soit $credit_{in}(direction_{Sortie})$: la valeur du crédit au niveau du buffer d'entrée pour la $direction_{Sortie}$
Soit $credit_{out}$: la valeur du crédit au niveau du port de sortie du routeur précédent ou du $BSNI$

pour chaque BE **faire**
  $flit = BE.peek()$[1]
  si $(flit.etat == 6)$ et $(currentTime == flit.finArbitrage)$ et $(PS.direction_{Gagnante} \neq BE.direction_{Entree})$ **alors**
    $ChangerEtat(flit, 5)$
  fin si
fin pour
si $(Routers.get(j).getBE(direction_{Entree}).getcredit_{in}(direction_{Sortie}) > 0)$ **alors**
  $flit = Routeurs.get(j).getBE(PS.direction_{Gagnante}).pop()$[3]
  si $(((flit ==$ *en-tête*$)$ et $(flit.etat == 6)$ et $(currentTime \geq flit.finArbitrage))$ ou $((flit.etat == 5)et(flit$ est le seul flit sollicitant $PS)$) ou $(flit ==$ *non en-tête* (data ou queue))) **alors**
    $ChangerEtat(flit, 7)$
    $credit_{out}^{++}$
    si $(credit_{out} == 1)$ **alors**
      si $(direction_{Entree} == local)$ **alors**
        Schedule($TransfertFlits_{NI\_routeur}(j)$,$currentTime$)
      sinon
        $k = Routers.get(j).getNeighbor(direction_{Entree})$
        Schedule($TraverserLien_{InterRouteurs}(k, inverse(direction_{Entree})$,$currentTime$)
      fin si
    fin si
    pour (chaque $direction$ $|direction \neq direction_{Sortie})$ **faire**
      $Routers.get(j).getBE(direction).credit_{in}(direction_{Sortie})^{--}$
    fin pour
    Schedule($FlitArriveBufferSortie(j, direction_{Sortie}, flit), currentTime+T_{TraverseSwitch}$)
    si $(flit ==$ *queue*$)$ **alors**
      $flit_{suivant} = Routeurs.get(j).getBE(PS.direction_{Gagnante}).peek()$[1]
      si $(flit_{suivant} \neq NULL)$ et $(flit_{suivant} ==$ *en-tête*$)$ **alors**
        Schedule($CalculerRoute(j, direction_{Entree}), currentTime$)
      fin si
      $Routers.get(j).getBE(PS.direction_{Gagnante}).setdirectiondestination(NULL)$
      $PS.direction_{Gagnante} = NULL$
      Schedule($AppliquerArbitrage(j, direction_{Sortie}), currentTime$)
    sinon
      Schedule($TraverserSwitch(j, direction_{Entree}, direction_{Sortie}), currentTime + \omega$)
    fin si
  fin si
fin si

Algorithme 13 $FlitArriveBufferSortie(j, direction_{Sortie}, flit)$:

Soit BS : le buffer de sortie du routeur j
Soit nb_{BS} : le nombre de flits dans le buffer de sortie du routeur j

$BS.add(flit)$
$ChangerEtat(flit, 8)$
si $(nb_{BS} == 1)\&\&(credit_{out} > 0)$ **alors**
  si $(BS.getdirection() == Local)$ **alors**
    Schedule $(TransfertFlits_{(Routeur\_NI)}(j), currentTime)$
  sinon
    Schedule $(TraverserLien_{InterRouteurs}(j, direction_{Sortie}), currentTime)$
  fin si
fin si

Algorithme 14 $TraverserLien_{InterRouteurs}(j, direction_{Sortie})$

Soit BS : le buffer de sortie du routeur j (routeur source)
Soit K : l'indice du routeur destination
Soit nb_{BS} : le nombre de flits à l'intérieur du BS
Soit $credit_{out}$: la valeur du crédit au niveau de BS

si $(crédit_{out} > 0)$ && $(nb_{BS} > 0)$ **alors**
  $flit = BS.pop()$[3]
  $credit_{out}^{--}$
  pour (chaque $direction$ $|direction \neq direction_{Sortie})$ **faire**
    $Routers(j).getBE(direction).credit_{in}(direction_{Sortie})^{++}$
  fin pour
  si $(PS.direction_{Gagnante} \neq NULL)$ && $(Routers.get(j).getBE(PS.direction_{Gagnante}).credit_{in}$ $(direction_{Sortie}) == 1)$ **alors**
    Schedule $(TraverserSwitch(j, PS.direction_{Gagnante}, direction_{Sortie}), currentTime)$
  fin si
  $ChangerEtat(flit, 9)$
  $k = Routers.get(j).getNeighbor(direction_{Sortie})$
  Schedule $(FlitArriveBufferEntree(k, Inverse(direction_{Sortie}), flit), currentTime+$ $T_{Traversee_{InterRouteurs}})$
  si $(nb_{BS} > 0)$ &&$(credit_{out} > 0)$ **alors**
    Schedule $(TraverserLien_{InterRouteurs}(j, direction_{Sortie}), currentTime + \Phi)$
  fin si
fin si

Algorithme 15 $TransfertFlits_{(Routeur\_NI)}(j)$

Soit BL : le buffer local d'un routeur donné
Soit nb_{BL} : le nombre de flits à l'intérieur du BL
  $flit = BL.pop()$[3]
  pour (chaque $direction$ $|direction \neq Local$) **faire**
    $Routers.get(j).getBE(direction).credit_{in}(Local)^{++}$
  fin pour
  si $(PS.direction_{Gagnante} \neq NULL)$ && $(Routers.get(j).getBE(PS.direction_{Gagnante}).credit_{in}$ $(direction_{Sortie}) == 1)$ **alors**
    Schedule $(TraverserSwitch(j, PS.direction_{Gagnante}, Local), currentTime)$
  fin si
  $ChangerEtat(flit, 10)$
  Schedule $(FlitArriveBufferEntree_{NI}(j, flit)$, $currentTime + T_{Traversee_{Routeur-NI}})$
  si $(nb_{BL} > 0)$ **alors**
    Schedule $(TransfertFlits_{(Routeur\_NI)}(j)$, $currentTime + \Omega)$
  fin si

Algorithme 16 $FlitArriveBufferEntree_{NI}(j, flit)$

Soit $BENI$: le buffer d'entrée de l'interface réseau (NI)
  $BENI.add(flit)$
  $ChangerEtat(flit, 11)$
  $flit.TempsArrivee = currentTime$ {temps d'arrivée du flit}
  $paquet = flit.getPaquet()$ {retourne à quel paquet appartient le flit}
  $paquet.flitArrive^{++}$ {incrémenter le nombre de flits arrivés d'un paquet donné}
  si $(paquet.flitArrive == paquet.flitGenere)$ **alors**
    $paquet.TempsArrivee = currentTime$ {temps d'arrivée du paquet}
    e_i = $paquet.getCommunication()$ {retourne à quelle communication appartient le paquet}
    $e_i.paquetArrive^{++}$ {incrémenter le nombre de paquets arrivés d'une communication e_i donnée}
    si $(e_i.paquetArrive == e_i.paquetGenere)$ **alors**
      $e_i.finCommunication = currentTime$ {temps de fin d'une communication donnée}
      $T_i = Snk(ei)$
      $nb(Ti.pred()^{--})$ {décrémenter le nombre de communications entrantes de la tâche T_i }
      si $(nb(Ti.pred()) == 0)$ **alors**
        $Schedule(ExécuterTâche(T_i, PE_j), currentTime)$
      fin si
    fin si
  fin si

Algorithme 17 *ChangerEtat* (FLIT flit, int NouvelEtat)

si $(flit.etat \neq NULL)$ **alors**
$flit.setFlitDelaiParEtat(flit.etat, flit.getFlitDelaiParEtat(flit.etat)+(currentTime-flit.PrevTime))$;
switch $(flit.etat)$
case 1**:**
$EnergieParEtat = (currentTime - flit.PrevTime) * E_{NI}$;
case 2**:**
$EnergieParEtat = E_{Lien_{(NI-ROUTEUR)}}$;
case 3**:**
$EnergieParEtat = (currentTime - flit.PrevTime) * E_{BE}$;
case 4**:**
$EnergieParEtat = ((currentTime - flit.PrevTime) * E_{BE}) + E_{Routage}$;
case 5**:**
$EnergieParEtat = (currentTime - flit.PrevTime) * E_{BE}$;
case 6**:**
$EnergieParEtat = ((currentTime - flit.PrevTime) * E_{BE}) + E_{Arbitrage}$;
case 7**:**
$EnergieParEtat = E_{Crossbar}$;
case 8**:**
$EnergieParEtat = (currentTime - flit.PrevTime) * E_{BS}$;
case 9**:**
$EnergieParEtat = E_{Lien_{(ROUTEUR-ROUTEUR)}}$;
case 10**:**
$EnergieParEtat = E_{Lien_{(NI-ROUTEUR)}}$;
case 11**:**
$EnergieParEtat = (currentTime - flit.PrevTime) * E_{NI}$;
end switch
$flit.setFlitEnergieParEtat(flit.etat, flit.getFlitEnergieParEtat(flit.etat) + EnergieParEtat)$;
fin si
$flit.etat = NouvelEtat$
$flit.PrevTime = currentTime$

Algorithmes \ Runs	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
NSGAII	1.0	0.0	0.0	1.0	14.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	6.0	1.0	14.0
HNSGAII	1.0	0.0	0.0	0.0	14.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
SPEA2	5.0	6.0	6.0	6.0	5.0	6.0	6.0	6.0	5.0	6.0	6.0	5.0	6.0	6.0	6.0
HSPEA2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
PESAII	8.0	21.0	14.0	14.0	14.0	14.0	14.0	168.0	19.0	14.0	14.0	9.0	21.0	9.0	21.0
HPESAII	3.0	1.0	4.0	1.0	6.0	4.0	0.0	4.0	0.0	1.0	3.0	4.0	1.0	1.0	1.0
FastPGA	8.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	5.0	9.0	6.0	5.0	9.0	9.0	8.0
HFastPGA	3.0	0.0	1.0	0.0	5.0	1.0	1.0	0.0	0.0	3.0	1.0	3.0	3.0	4.0	1.0
IBEA	257.0	211.0	251.0	122.0	154.0	125.0	109.0	182.0	240.0	168.0	91.0	195.0	105.0	197.0	241.0
HIBEA	1.0	4.0	3.0	4.0	5.0	4.0	4.0	4.0	4.0	4.0	1.0	4.0	4.0	4.0	84.0
MBB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

TABLE B.1 – Valeurs de l'indicateur Epsilon selon la TABLE 5.5

Algorithmes \ Runs	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29
NSGAII	0.0	1.0	0.0	6.0	1.0	0.0	0.0	14.0	9.0	1.0	5.0	0.0	14.0	1.0	14.0
HNSGAII	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
SPEA2	5.0	6.0	5.0	6.0	5.0	5.0	6.0	6.0	5.0	6.0	9.0	8.0	6.0	6.0	5.0
HSPEA2	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	4.0	8.0	0.0	0.0	0.0
PESAII	21.0	14.0	21.0	14.0	14.0	14.0	152.0	21.0	14.0	19.0	21.0	21.0	5.0	24.0	19.0
HPESAII	14.0	1.0	21.0	4.0	4.0	4.0	1.0	14.0	14.0	1.0	0.0	0.0	0.0	3.0	1.0
FastPGA	9.0	9.0	5.0	6.0	9.0	5.0	9.0	5.0	6.0	6.0	9.0	8.0	6.0	6.0	8.0
HFastPGA	4.0	1.0	0.0	0.0	4.0	0.0	4.0	1.0	5.0	6.0	0.0	6.0	1.0	0.0	3.0
IBEA	266.0	152.0	109.0	256.0	146.0	231.0	256.0	142.0	282.0	213.0	150.0	203.0	111.0	100.0	306.0
HIBEA	4.0	4.0	4.0	4.0	4.0	1.0	0.0	14.0	4.0	1.0	4.0	14.0	4.0	3.0	3.0
MBB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

TABLE B.2 – Valeurs de l'indicateur Epsilon selon la TABLE 5.5 (suite)

Algorithmes \ Runs	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
NSGAII	3.0	19.0	19.0	7.0	7.0	7.0	19.0	4.0	13.0	7.0	19.0	7.0	19.0	7.0	7.0
HNSGAII	3.0	19.0	7.0	0.0	0.0	0.0	0.0	0.0	7.0	0.0	0.0	7.0	0.0	0.0	7.0
SPEA2	21.0	19.0	11.0	24.0	24.0	24.0	20.0	10.0	7.0	11.0	23.0	24.0	19.0	23.0	24.0
HSPEA2	7.0	7.0	0.0	0.0	11.0	19.0	4.0	7.0	7.0	11.0	0.0	7.0	7.0	7.0	12.0
PESAII	38.0	24.0	24.0	24.0	26.0	10.0	16.0	327.0	26.0	202.0	26.0	38.0	10.0	24.0	38.0
HPESAII	10.0	0.0	14.0	13.0	19.0	0.0	0.0	7.0	13.0	7.0	7.0	7.0	0.0	11.0	0.0
FastPGA	38.0	38.0	20.0	24.0	10.0	38.0	38.0	26.0	24.0	38.0	34.0	13.0	26.0	38.0	24.0
HFastPGA	0.0	7.0	7.0	0.0	7.0	3.0	0.0	7.0	0.0	0.0	7.0	10.0	7.0	7.0	0.0
IBEA	180.0	63.0	220.0	361.0	83.0	246.0	132.0	148.0	104.0	248.0	104.0	342.0	210.0	402.0	106.0
HIBEA	0.0	55.0	0.0	0.0	7.0	3.0	7.0	0.0	0.0	0.0	0.0	7.0	0.0	7.0	7.0
MBB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

TABLE B.3 – Valeurs de l'indicateur Epsilon selon la TABLE 5.7

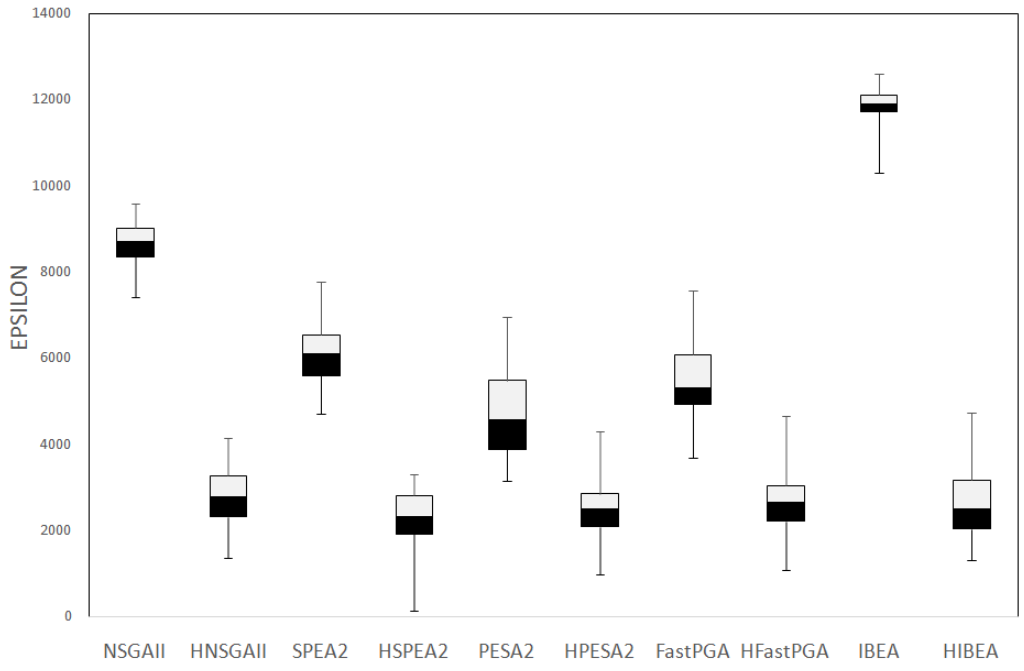


FIGURE B.1 – Algorithmes hybrides vs non hybrides pour une grande instances du mapping

Algorithmes \ Runs	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29
NSGAII	16.0	16.0	10.0	3.0	13.0	10.0	7.0	7.0	16.0	11.0	13.0	13.0	0.0	19.0	22.0
HNSGAII	0.0	0.0	0.0	0.0	7.0	0.0	7.0	0.0	0.0	7.0	0.0	0.0	0.0	0.0	19.0
SPEA2	38.0	10.0	10.0	11.0	24.0	7.0	24.0	19.0	38.0	26.0	24.0	24.0	38.0	26.0	24.0
HSPEA2	7.0	7.0	0.0	7.0	14.0	7.0	11.0	7.0	7.0	7.0	0.0	7.0	7.0	0.0	0.0
PESAII	212.0	26.0	291.0	50.0	24.0	23.0	57.0	14.0	23.0	38.0	52.0	38.0	26.0	50.0	26.0
HPESAII	0.0	7.0	7.0	0.0	0.0	19.0	11.0	7.0	7.0	0.0	22.0	0.0	3.0	7.0	7.0
FastPGA	26.0	26.0	24.0	24.0	38.0	26.0	13.0	13.0	19.0	16.0	38.0	26.0	26.0	24.0	22.0
HFastPGA	0.0	19.0	13.0	0.0	0.0	0.0	7.0	7.0	12.0	7.0	0.0	7.0	0.0	0.0	0.0
IBEA	264.0	336.0	299.0	97.0	320.0	204.0	202.0	187.0	215.0	263.0	91.0	110.0	184.0	242.0	113.0
HIBEA	7.0	7.0	7.0	7.0	0.0	7.0	11.0	0.0	7.0	0.0	22.0	11.0	7.0	7.0	7.0
MBB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

TABLE B.4 – Valeurs de l'indicateur Epsilon selon la TABLE 5.7 (suite)

IGD / Algorithmes	Moyenne	Écart-Type	Médiane	Min	Max	Temps d'exécution
NSGAII	$7.97e-01$	$4.2e-02$	$7.87e-01$	$7.21e-01$	$8.75e-01$	3, 12m
HNSGAII	$1.80e-01$	$5.0e-02$	$1.70e-01$	$9.10e-02$	$2.82e-01$	6, 354m
SPEA2	$5.28e-01$	$6.3e-02$	$5.43e-01$	$3.99e-01$	$6.30e-01$	2, 933m
HSPEA2	$1.46e-01$	$3.2e-02$	$1.45e-01$	$8.95e-02$	$2.08e-02$	6, 121m
PESAII	$3.61e-01$	$9.5e-02$	$3.49e-01$	$1.98e-01$	$5.81e-01$	2, 639m
HPESAII	$1.57e-01$	$5.1e-02$	$1.48e-01$	$3.57e-02$	$2.99e-01$	5, 83m
FastPGA	$4.76e-01$	$9.6e-02$	$4.67e-01$	$2.74e-01$	$6.84e-01$	2, 87m
HFastPGA	$1.63e-01$	$5.6e-02$	$1.59e-01$	$4.35e-02$	$3.21e-01$	6, 049m
IBEA	$1.14e+00$	$5.8e-02$	$1.15e+00$	$9.04e-01$	$1.22e+00$	0, 1m
HIBEA	$1.77e-01$	$5.4e-02$	$1.66e-01$	$9.25e-02$	$3.21e-01$	3, 339m

Table B.5 – Indices statistiques. IGD. algorithmes hybrides vs algorithmes non hybrides (grande instance du mapping)

Epsilon / Algorithmes	Moyenne	Écart-Type	Médiane	Min	Max	Temps d'exécution
NSGAII	$8.62e+03$	$5.5e+02$	$8.72e+03$	$7,41e+03$	$9.59e+03$	3, 12m
HNSGAII	$2.79e+03$	$7.0e+02$	$2.78e+03$	$1.36e+03$	$4.14e+03$	6, 354m
SPEA2	$6.07e+03$	$7.6e+02$	$6.11e+03$	$4.70e+03$	$7.76e+03$	2, 933m
HSPEA2	$2.22e+03$	$7.4e+02$	$2.32e+03$	$1.44e+02$	$3.30e+03$	6, 121m
PESAII	$4.67e+03$	$9.8e+02$	$4.57e+03$	$3.14e+03$	$6.96e+03$	2, 639m
HPESAII	$2.48e+03$	$7.6e+02$	$2.50e+03$	$9.82e+02$	$4.31e+03$	5, 83m
FastPGA	$5.52e+03$	$9.4e+02$	$5.32e+03$	$3.70e+03$	$7.57e+03$	2, 87m
HFastPGA	$2.61e+03$	$7.3e+02$	$2.65e+03$	$1.08e+03$	$4.66e+03$	6, 049m
IBEA	$1.18e+04$	$4.9e+02$	$1.190e+03$	$1.03e+04$	$1.26e+04$	0, 1m
HIBEA	$2.65e+03$	$7.9e+02$	$2.52e+03$	$1.32e+03$	$4.73e+03$	3, 339m

Table B.6 – Indices statistiques. Epsilon. algorithmes hybrides vs algorithmes non hybrides (grande instance du mapping)

Table B.7 – Paramétrage des algorithmes NSGAII et HNSGAII

NSGAII [1]	
Taille de la population	100
Max Evaluation	100000
Probabilité de mutation (pm)	1.0/L (L : longeur du chromosome)
Probabilité de croisement (pc)	0.8
HNSGAII proposé	
Taille de la population	100
Max Evaluation (NSGAII)	50000
Probabilité de mutation (pm)	1.0/L (L : longeur du chromosome)
Probabilité de croisement (pc)	0.8
Température initiale (T_0)	500
Température finale (T_1)	10^{-3}
Taux de refroidissement (α)	0.9
Max Iterations (AMOSA)	200
HL	100
SL	110
Gamma	1.1

epsilon / Algorithmes	Moyenne	Écart-Type	Médiane	Min	Max	Temps d'exécution
NSGAII	$3.88e+02$	$2.0e+02$	$3.72e+02$	$9.6e+01$	$7.08+02$	2, 538s
HNSGAII	$3.53e+02$	$2.2e+02$	$3.42e+02$	$3.6e+01$	$8.16e+02$	1, 569s

Table B.8 – Indices statistiques. epsilon. HNSGAII vs NSGAII (grande instance du mapping)

IGD / Algorithmes	Moyenne	Écart-Type	Médiane	Min	Max	Temps d'exécution
NSGAII	$3.77e-02$	$8.8e-03$	$3.56e-02$	$2.50e-02$	$5.68e-02$	$2,538$s
HNSGAII	$1.76e-02$	$4.0e-03$	$1.70e-02$	$1.21e-02$	$2.96e-02$	$1,569$s

TABLE B.9 – Indices statistiques. IGD. HNSGAII vs NSGAII (grande instance du mapping)

Bibliographie

[1] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm : Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2) :182–197, 2002.

[2] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2 : Improving the strength pareto evolutionary algorithm. *TIK-report*, 103, 2001.

[3] David W Corne, Nick R Jerram, Joshua D Knowles, and Martin J Oates. Pesa-ii : Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 283–290, 2001.

[4] Hamidreza Eskandari, Christopher D Geiger, and Gary B Lamont. Fastpga : A dynamic population sizing approach for solving expensive multiobjective optimization problems. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 141–155. Springer, 2007.

[5] Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In *International conference on parallel problem solving from nature*, pages 832–842. Springer, 2004.

[6] Sanghamitra Bandyopadhyay, Sriparna Saha, Ujjwal Maulik, and Kalyanmoy Deb. A simulated annealing-based multiobjective optimization algorithm : Amosa. *IEEE transactions on evolutionary computation*, 12(3) :269–283, 2008.

[7] Med Aymen Siala and Slim Ben Saoud. A survey on existing mp-socs architectures. *International Journal of Computer Applications*, 19(3) :28–41, 2011.

[8] Milica Mitić and Mile Stojčev. An overview of on-chip buses. *Facta universitatis-series : Electronics and Energetics*, 19(3) :405–428, 2006.

[9] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, 38(1) :1–es, 2006.

[10] José Duato, Sudhakar Yalamanchili, and Lionel M. Ni. *Interconnection networks : an engineering approach*. Morgan Kaufmann, San Francisco, CA, rev. printing edition, 2003.

[11] Érika Cota, Alexandre de Morais Amory, and Marcelo Soares Lubaszewski. *Reliability, Availability and Serviceability of Networks-on-chip*. Springer Science & Business Media, 2011.

[12] Fayez Gebali, Haytham Elmiligi, and Mohamed Watheq El-Kharashi. *Networks-on-chips : theory and practice*. CRC press, 2011.

[13] El-Ghazali Talbi. *Metaheuristics : from design to implementation*, volume 74. John Wiley & Sons, 2009.

[14] Yann Collette and Patrick Siarry. *Optimisation multiobjectif*. Editions Eyrolles, 2002.

[15] El-ghazali Talbi. Métaheuristiques pour l'optimisation combinatoire multi-objectif : Etat de l'art. *Rapport CNET (France Telecom) Octobre*, 1999.

[16] Marino Widmer. Les métaheuristiques : des outils performants pour les problèmes industriels. In *3ème Conférence Francophone de MOdélisation et SIMulation MOSIM*, volume 1, pages 25–27, 2001.

[17] Ilhem Boussaid. *Perfectionnement de métaheuristiques pour l'optimisation continue*. PhD thesis, Paris Est, 2013.

[18] D Hadka. Beginner's guide to the moea framework, 2016.

[19] E-G Talbi. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(5) :541–564, 2002.

[20] Juan J. Durillo and Antonio J. Nebro. jMetal : A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10) :760–771, October 2011.

[21] Gustavo Sanchez, Luciano Agostini, Leonel Sousa, and César Marcon. Parallelism exploration for 3d high-efficiency video coding depth modeling mode one. *Journal of Real-Time Image Processing*, 17(4) :787–797, 2020.

[22] Paolo Meloni, Giuseppe Tuveri, Danilo Pani, Luigi Raffo, and Francesca Palumbo. Exploring custom heterogeneous mpsocs for real-time neural signal decoding. In *2015 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–8. IEEE, 2015.

[23] Mohammed Karim et al. A microblaze-based multiprocessor system on chip for real-time cardiac monitoring. In *2014 International Conference on Multimedia Computing and Systems (ICMCS)*, pages 331–336. IEEE, 2014.

[24] Arman Iranfar, Ali Pahlevan, Marina Zapater, Martin Žagar, Mario Kovač, and David Atienza. Online efficient bio-medical video transcoding on mpsocs through content-aware workload allocation. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 949–954. IEEE, 2018.

[25] Bisrat Tafesse and Venkatesan Muthukumar. Framework for simulation of heterogeneous mpsoc for design space exploration. *VLSI Design*, 2013, 2013.

[26] Julien Hascoet. *Contributions to Software Runtime for Clustered Manycores Applied to Embedded and High-Performance Applications.* PhD thesis, 2018.

[27] Robert R Schaller. Moore's law : past, present and future. *IEEE spectrum*, 34(6) :52–59, 1997.

[28] Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1) :82–85, 1998.

[29] Mohamed Fehmi Chatmen. *Conception d'un réseau sur puce optimisé en latence.* PhD thesis, 2016.

[30] Wayne Wolf. The future of multiprocessor systems-on-chips. In *Proceedings. 41st Design Automation Conference, 2004.*, pages 681–685. IEEE, 2004.

[31] Jeronimo Castrillon, Weihua Sheng, Ralph Jessenberger, Lothar Thiele, Lars Schorr, Ben Juurlink, Mauricio Alvarez-Mesa, Angela Pohl, Victor Reyes, and Rainer Leupers. Multi/many-core programming : Where are we standing ? In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1708–1717. IEEE, 2015.

[32] A Kouadri-Mostefaoui. *Architectures Flexibles pour la Validation et L'exploration de Réseaux-sur-Puce.* PhD thesis, 2009.

[33] Kiril Georgiev. *Débogage des systèmes embarqués multiprocesseur basé sur la ré-exécution déterministe et partielle.* PhD thesis, Université de Grenoble, 2012.

[34] Georgios Kornaros. *Multi-core embedded systems.* CRC Press, 2018.

[35] Rainer Leupers, Miguel Angel Aguilar, Jeronimo Castrillon, and Weihua Sheng. Software compilation techniques for heterogeneous embedded multi-core systems. In *Handbook of Signal Processing Systems*, pages 1021–1062. Springer, 2019.

[36] Bernd Hoefflinger. Itrs : The international technology roadmap for semiconductors. In *Chips 2020*, pages 161–174. Springer, 2011.

[37] Ahmed Hemani, Axel Jantsch, Shashi Kumar, Adam Postula, Johnny Oberg, Mikael Millberg, and Dan Lindqvist. Network on chip : An architecture for billion transistor era. In *Proceeding of the IEEE NorChip Conference*, volume 31, page 0. sn, 2000.

[38] William J Dally and Brian Towles. Route packets, not wires : on-chip inteconnection networks. In *Proceedings of the 38th annual Design Automation Conference*, pages 684–689, 2001.

[39] Luca Benini and Giovanni De Micheli. Networks on chips : A new soc paradigm. *computer*, 35(1) :70–78, 2002.

[40] Giovanni De Micheli and Luca Benini. Networks on chips : 15 years later. *Computer*, (5) :10–11, 2017.

[41] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[42] Ewerson Carvalho, César Marcon, Ney Calazans, and Fernando Moraes. Evaluation of static and dynamic task mapping algorithms in noc-based mpsocs. In *2009 International Symposium on System-on-Chip*, pages 087–090. IEEE, 2009.

[43] jmetal. http ://jmetal.sourceforge.net/. Consulté : 2020.

[44] Antonio J Nebro, Juan J Durillo, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. Mocell : A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24(7) :726–746, 2009.

[45] Margarita Reyes Sierra and Carlos A Coello Coello. Improving pso-based multi-objective optimization using crowding, mutation and ϵ-dominance. In *International conference on evolutionary multi-criterion optimization*, pages 505–519. Springer, 2005.

[46] Antonio J Nebro, Francisco Luna, Enrique Alba, Bernabé Dorronsoro, Juan J Durillo, and Andreas Beham. Abyss : Adapting scatter search to multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 12(4) :439–457, 2008.

[47] Antonio J Nebro, Juan José Durillo, Jose Garcia-Nieto, CA Coello Coello, Francisco Luna, and Enrique Alba. Smpso : A new pso-based metaheuristic for multi-objective optimization. In *2009 IEEE Symposium on computational intelligence in multi-criteria decision-making (MCDM)*, pages 66–73. IEEE, 2009.

[48] Dihia Belkacemi, Youcef Bouchebaba, Mehammed Daoui, and Mustapha Lalam. Network on chip and parallel computing in embedded systems. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 146–152. IEEE, 2016.

[49] Dongkyung Nam and Cheol Hoon Park. Multiobjective simulated annealing : A comparative study to evolutionary algorithms. *International Journal of Fuzzy Systems*, 2(2) :87–97, 2000.

[50] Katia Jaffres-Runser, Jean-Marie Gorce, and Cristina Comaniciu. A multiobjective tabu framework for the optimization and evaluation of wireless systems. *arXiv preprint arXiv :0907.3777*, 2009.

[51] J Carlos Soto-Monterrubio, Alejandro Santiago, Héctor J Fraire-Huacuja, Juan Frausto-Solís, and David Terán-Villanueva. Branch and bound algorithm for the heterogeneous computing scheduling multi-objective problem. *International Journal of Combinatorial Optimization Problems and Informatics*, 7(3) :7–19, 2016.

[52] Dihia Belkacemi, Mehammed Daoui, Samia Bouzefrane, and Youcef Bouchebaba. Parallel applications mapping onto network on chip based on heterogeneous mp-socs using hybrid algorithms. *International Journal of Distributed Systems and Technologies (IJDST)*, 10(2) :37–63, 2019.

[53] Marilyn Wolf. *Multiprocessor systems-on-chips*. Morgan Kaufmann, 2005.

[54] Katalin Popovici, Frédéric Rousseau, Ahmed Amine Jerraya, and Marilyn Wolf. *Embedded software design and programming of multiprocessor system-on-chip*. Springer, 2010.

[55] Geoffrey Blake, Ronald G Dreslinski, and Trevor Mudge. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6) :26–37, 2009.

[56] Haris Javaid and Sri Parameswaran. *Pipelined multiprocessor system-on-chip for multimedia*. Springer, 2014.

[57] Sabri Pllana and Fatos Xhafa. *Programming Multicore and Many-core Computing Systems*. John Wiley & Sons, 2017.

[58] Michael Hübner and Jürgen Becker. *Multiprocessor system-on-chip : hardware design and tool integration*. Springer Science & Business Media, 2010.

[59] Shuvra S Bhattacharyya, Ed F Deprettere, Rainer Leupers, and Jarmo Takala. *Handbook of signal processing systems*. Springer, 2018.

[60] Nvidia tegra. https ://www.toradex.com/computer-on-modules/nvidia-tegra. Consulté : 2019.

[61] Texas Instruments,"omap mobile processors". https ://www.ti.com/. Consulté : 2019.

[62] Bhargavi Gedela and Lohitha Bheemisetty. A Literature Review on Uniprocessor to Multiprocessor in Computer Architecture. 2(2) :4, 2019.

[63] Jingcao Hu, Yangdong Deng, and Radu Marculescu. System-level point-to-point communication synthesis using floorplanning information [soc]. In *Proceedings of ASP-DAC/VLSI Design 2002. 7th Asia and South Pacific Design Automation Conference and 15h International Conference on VLSI Design*, pages 573–579. IEEE, 2002.

[64] Hyung Gyu Lee, Naehyuck Chang, Umit Y Ogras, and Radu Marculescu. On-chip communication architecture exploration : A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(3) :1–20, 2008.

[65] Abir M'Zah. *ASIC Design Methodology for 3D NOC Based Heterogeneous Multi Processor on Chip*. PhD thesis, 2012.

[66] Sudeep Pasricha and Nikil Dutt. *On-chip communication architectures : system on chip interconnect*. Morgan Kaufmann, 2010.

[67] Florentine Dubois. *Une méthodologie de conception de modèles analytiques de surface et de puissance de réseaux sur puce hautement paramétriques basée sur une méthode d'apprentissage automatique*. PhD thesis, Grenoble, 2013.

[68] Samuel Evain. *μSpider Environnement de Conception de Réseau sur Puce*. PhD thesis, 2006.

[69] Chrysostomos Nicopoulos, Vijaykrishnan Narayanan, and Chita R Das. *Network-on-chip architectures : A holistic design exploration*, volume 45. Springer Science & Business Media, 2009.

[70] Abderazek Ben Abdallah. *Advanced multicore Systems-On-Chip*. Springer, 2017.

[71] Pierre Guerrier and Alain Greiner. A generic architecture for on-chip packet-switched interconnections. 2000.

[72] Jack Browne. On-chip communications network report. *Mountain View, CA, USA : Sonics*, 2012.

[73] Benoıt Dupont de Dinechin, Renaud Ayrignac, Pierre-Edouard Beaucamps, Patrice Couvert, Benoıt Ganne, Pierre Guironnet de Massas, François Jacquet, Samuel Jones, Nicolas Morey Chaisemartin, Frédéric Riss, et al. A clustered manycore processor architecture for embedded and accelerated applications. In *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2013.

[74] Anish Varghese, Bob Edwards, Gaurav Mitra, and Alistair P Rendell. Programming the adapteva epiphany 64-core network-on-chip coprocessor. *The International Journal of High Performance Computing Applications*, 31(4) :285–302, 2017.

[75] Giovanni De Micheli, Ciprian Seiculescu, Srinivasan Murali, Luca Benini, Federico Angiolini, and Antonio Pullini. Networks on chips : From research to products. In *Proceedings of the 47th Design Automation Conference*, pages 300–305, 2010.

[76] Natalie Enright Jerger and Li-Shiuan Peh. On-chip networks. *Synthesis Lectures on Computer Architecture*, 4(1) :1–141, 2009.

[77] Abderazek Ben Abdallah. *Advanced Multicore Systems-On-Chip*. Springer Singapore, Singapore, 2017.

[78] Santanu Kundu and Santanu Chattopadhyay. *Network-on-Chip : The Next Generation of System-on-Chip Integration*. CRC Press, December 2014.

[79] Ahmed Ben Achballah and Slim Ben Saoud. A Survey of Network-On-Chip Tools. *International Journal of Advanced Computer Science and Applications*, 4(9), 2013.

[80] Sudeep Pasricha and Nikil Dutt. *On-chip communication architectures : system on chip interconnect*. Elsevier / Morgan Kaufmann Publishers, Amsterdam ; Boston, 2008.

[81] Wang Zhang, Ligang Hou, Jinhui Wang, Shuqin Geng, and Wuchen Wu. Comparison research between xy and odd-even routing algorithm of a 2-dimension 3x3 mesh

topology network-on-chip. In *2009 WRI Global Congress on Intelligent Systems*, volume 3, pages 329–333. IEEE, 2009.

[82] Hazem Moussa. *Architectures de réseaux sur puce pour décodeurs canal multiprocesseurs*. PhD thesis, 2009.

[83] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[84] Grant Martin. Overview of the mpsoc design challenge. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 274–279. IEEE, 2006.

[85] Eric Stotzer. Towards using openmp in embedded systems. In *OpenMPCon : Developers Conference*, volume 178, 2015.

[86] Adnan Agbaria, Dong-In Kang, and Karandeep Singh. Lmpi : Mpi for heterogeneous embedded distributed systems. In *12th International Conference on Parallel and Distributed Systems-(ICPADS'06)*, volume 1, pages 8–pp. IEEE, 2006.

[87] Ruxandra Pop and Shashi Kumar. A survey of techniques for mapping and scheduling applications to network on chip systems. *School of Engineering, Jonkoping University, Research Report*, 4(4), 2004.

[88] Jerónimo Castrillón Mazo and Rainer Leupers. *Programming Heterogeneous MPSoCs*. Springer, 2013.

[89] ASL ELNAZ ALIKHAH and Midia Reshadi. Application mapping onto network-on-chip using bypass channel. 2016.

[90] Mohammed Kamal Benhaoua. *Placement Dynamique D'Applications Embarquées Intensives sur des Réseaux de Processeurs sur Puce*. PhD thesis, 2014.

[91] Haider Ali, Umair Ullah Tariq, Yongjun Zheng, Xiaojun Zhai, and Lu Liu. Contention & energy-aware real-time task mapping on noc based heterogeneous mpsocs. *IEEE Access*, 6 :75110–75123, 2018.

[92] Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen, and Vincent T'kindt. *Metaheuristics for multiobjective optimisation*, volume 535. Springer Science & Business Media, 2004.

[93] Adiel Teixeira de Almeida, Cristiano Alexandre Virgínio Cavalcante, Marcelo Hazin Alencar, Rodrigo José Pires Ferreira, Adiel Teixeira de Almeida-Filho, and Thalles Vitelli Garcez. *Multicriteria and Multiobjective Models for Risk, Reliability and Maintenance Decision Analysis*, volume 231 of *International Series in Operations Research & Management Science*. Springer International Publishing, Cham, 2015.

[94] Nyoman Gunantara. A review of multi-objective optimization : Methods and its applications. *Cogent Engineering*, 5(1) :1–16, 2018.

[95] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms : A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1) :32–49, 2011.

[96] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

[97] Lam Thu Bui and Sameer Alam. An introduction to multi-objective optimization. In *Multi-objective optimization in computational intelligence : Theory and practice*, pages 1–19. IGI Global, 2008.

[98] François-Xavier Irisarri. *Stratégies de calcul pour l'optimisation multiobjectif des structures composites*. PhD thesis, 2009.

[99] Michael TM Emmerich and André H Deutz. A tutorial on multiobjective optimization : fundamentals and evolutionary methods. *Natural computing*, 17(3) :585–609, 2018.

[100] Imen Harbaoui Dridi. *Optimisation heuristique pour la résolution du m-PDPTW statique et dynamique*. PhD thesis, 2010.

[101] Christos Zaroliagis. Recent advances in multiobjective optimization. In *International Symposium on Stochastic Algorithms*, pages 45–47. Springer, 2005.

[102] El-Ghazali Talbi, Matthieu Basseur, Antonio J Nebro, and Enrique Alba. Multi-objective optimization using metaheuristics : non-standard algorithms. *International Transactions in Operational Research*, 19(1-2) :283–305, 2012.

[103] Rachida Abounacer, Monia Rekik, and Jacques Renaud. An exact solution approach for multi-objective location–transportation problem for disaster response. *Computers & Operations Research*, 41 :83–93, 2014.

[104] Philippe Lacomme, Christian Prins, and Marc Sevaux. *Algorithmes de graphes*, volume 28. Eyrolles Paris, 2003.

[105] Balram Suman and Prabhat Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the operational research society*, 57(10) :1143–1160, 2006.

[106] Khalil Amine. Multiobjective simulated annealing : Principles and algorithm variants. *Advances in Operations Research*, 2019 :1–13, 2019.

[107] Xavier Gandibleux, Nazik Mezdaoui, and Arnaud Fréville. A tabu search procedure to solve multiobjective combinatorial optimization problems. In *Advances in multiple objective and goal programming*, pages 291–300. Springer, 1997.

[108] Yun Yang, Jianfeng Wu, Xiaomin Sun, Jichun Wu, and Chunmiao Zheng. A niched pareto tabu search for multi-objective optimal design of groundwater remediation systems. *Journal of Hydrology*, 490 :56–73, 2013.

[109] Joshua D Knowles and David W Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary computation*, 8(2) :149–172, 2000.

[110] Aberrahmane Bensmaïne. *Algorithmes évolutionnaires et méthodes approchées multicritères pour la génération des processus de fabrication dans un environnement reconfigurable*. PhD thesis, 2013.

[111] Sanghamitra Bandyopadhyay and Sriparna Saha. Some single-and multiobjective optimization techniques. In *Unsupervised Classification*, pages 17–58. Springer, 2013.

[112] Alfredo Arias-Montano, Carlos A Coello Coello, and Efrén Mezura-Montes. Multiobjective evolutionary algorithms in aeronautical and aerospace engineering. *IEEE Transactions on Evolutionary Computation*, 16(5) :662–694, 2012.

[113] Javier Ferrer, Francisco Chicano, and Enrique Alba. Evolutionary algorithms for the multi-objective test data generation problem. *Software : Practice and Experience*, 42(11) :1331–1362, 2012.

[114] Charles Darwin. *On the Origin of Species by Means of Natural Selection Or the Preservation of Favoured Races in the Struggle for Life.* H. Milford ; Oxford University Press, 1859.

[115] Ilhem BoussaïD, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information sciences*, 237 :82–117, 2013.

[116] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3) :221–248, 1994.

[117] Kalyanmoy Deb. Multi-objective optimisation using evolutionary algorithms : an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*, pages 3–34. Springer, 2011.

[118] Enrico Rigoni, Carlos Kavka, Alessandro Turco, Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, and Giovanni Mariani. Optimization algorithms for design space exploration of embedded systems. In *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*, pages 51–74. Springer, 2011.

[119] Abdelhakim Cheriet. *Métaheuristique hybride pour l'optimisation multi-objectif.* PhD thesis, Université Mohamed Khider-Biskra, 2016.

[120] Julien Autuori. *Energie, coopération méta-heuristiques et logique floue pour l'optimisation difficile.* PhD thesis, Troyes, 2014.

[121] Vassil Guliashki, Hristo Toshev, and Chavdar Korsemov. Survey of evolutionary

algorithms used in multiobjective optimization. *Problems of engineering cybernetics and robotics*, 60(1) :42–54, 2009.

[122] Wahabou Abdou, Christelle Bloch, Damien Charlet, and François Spies. Algorithme génétique multi-objectifs adaptatif. 2013.

[123] Mouadh Yagoubi. *Optimisation évolutionnaire multi-objectif parallèle : application à la combustion Diesel*. PhD thesis, 2012.

[124] Juan J Durillo, YuanYuan Zhang, Enrique Alba, and Antonio J Nebro. A study of the multi-objective next release problem. In *2009 1st International Symposium on Search Based Software Engineering*, pages 49–58. IEEE, 2009.

[125] Safae Dahmani. *Modèles et protocoles de cohérence de données, décision et optimisation à la compilation pour des architectures massivement parallèles*. PhD thesis, 2015.

[126] Fred Glover. A template for scatter search and path relinking. *Lecture notes in computer science*, 1363 :13–54, 1998.

[127] David Meignan. *Une approche organisationnelle et multi-agent pour la modélisation et l'implantation de métaheuristiques, Application aux problèmes d'optimisation de réseaux de transports*. PhD thesis, 2008.

[128] Ricardo P Beausoleil. Multiple criteria scatter search. In *4th Metaheuritics International Congress, Porto, Portugal*, pages 539–543, 2001.

[129] Ricardo P Beausoleil. "moss" multiobjective scatter search applied to non-linear multiple criteria optimization. *European Journal of Operational Research*, 169(2) :426–449, 2006.

[130] Ingrida Radziukynienė and Antanas Žilinskas. Evolutionary methods for multi-objective portfolio optimization. In *Proceedings of the World Congress on Engineering*, volume 2, pages 1155–1159, 2008.

[131] Juan J Durillo. *Metaheuristics for multi-objective optimization : design, analysis, and applications*. PhD thesis, Universidad de Málaga, 2011.

[132] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[133] Xin-She Yang, Gebrail Bekdaş, and Sinan Melih Nigdeli. *Metaheuristics and optimization in civil engineering*. Springer, 2016.

[134] Saptarshi Sengupta, Sanchita Basak, and Richard Alan Peters. Particle swarm optimization : A survey of historical and recent developments with hybridization perspectives. *Machine Learning and Knowledge Extraction*, 1(1) :157–191, 2019.

[135] Roger Serra and Julien Olivier. Apport de l'optimisation par essaims particulaires pour la détection de modifications structurelles à partir des propriétés dynamiques. In *Congrès français de mécanique*. AFM, Association Française de Mécanique, 2017.

[136] A Kaveh. Particle swarm optimization. In *Advances in Metaheuristic Algorithms for Optimal Design of Structures*, pages 11–43. Springer, 2017.

[137] Soniya Lalwani, Sorabh Singhal, Rajesh Kumar, and Nilama Gupta. A comprehensive survey : Applications of multi-objective particle swarm optimization (mopso) algorithm. *Transactions on combinatorics*, 2(1) :39–101, 2013.

[138] Susana Cecilia Esquivel and Leticia Cagnina. Multi-objective optimization with a gaussian pso algorithm. In *XIV Congreso Argentino de Ciencias de la Computación*, 2008.

[139] Adriana Cortes Godinez, Luis Ernesto Mancilla Espinosa, and Efrén Mezura Montes. An experimental comparison of multiobjective algorithms : Nsga-ii and omopso. In *2010 IEEE Electronics, Robotics and Automotive Mechanics Conference*, pages 28–33. IEEE, 2010.

[140] Farid Bourennani, Shahryar Rahnamayan, and Greg F Naterer. Leaders and speed constraint multi-objective particle swarm optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 908–915. IEEE, 2013.

[141] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. Quality assessment of pareto set approximations. In *Multiobjective Optimization*, pages 373–404. Springer, 2008.

[142] Miqing Li and Xin Yao. Quality evaluation of solution sets in multiobjective optimisation : A survey. *ACM Computing Surveys (CSUR)*, 52(2) :1–38, 2019.

[143] Arnaud Liefooghe and Bilel Derbel. A correlation analysis of set quality indicator values in multiobjective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 581–588, 2016.

[144] Carlos A Coello Coello and Margarita Reyes Sierra. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In *Mexican international conference on artificial intelligence*, pages 688–697. Springer, 2004.

[145] Guido Arnau Antoniucci. Analysis of the distribution of pareto optimal solutions on various multi-objective evolutionary algorithms. B.S. thesis, Universitat Politècnica de Catalunya, 2016.

[146] Juan J Durillo and Antonio J Nebro. jmetal : A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10) :760–771, 2011.

[147] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. Mapping on multi/many-core systems : survey of current and emerging trends. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–10. IEEE, 2013.

[148] Maribell Sacanamboy, Freddy Bolanos, and Ruben Nieto. A primer for mapping techniques on noc systems. In *ESA (2014) Proceedings 12th International Conference on Embedded Systems and Applications*, pages 70–75, 2014.

[149] Ruxandra Pop and Shashi Kumar. A Survey of Techniques for Mapping and Scheduling Applications to Network on Chip Systems. *Research Report*, page 45.

[150] Pradip Kumar Sahu and Santanu Chattopadhyay. A survey on application mapping strategies for network-on-chip design. *Journal of systems architecture*, 59(1) :60–76, 2013.

[151] Toubaline Nesrine, Bennouar Djamel, and Mahdoum Ali. A classification and evaluation framework for noc mapping strategies. *Journal of Circuits, Systems and Computers*, 26(02) :1730001, 2017.

[152] Jingcao Hu and Radu Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, pages 233–239, 2003.

[153] Jingcao Hu and Radu Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 234–239. IEEE, 2004.

[154] Tang Lei and Shashi Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Euromicro Symposium on Digital System Design, 2003. Proceedings.*, pages 180–187. IEEE, 2003.

[155] Srinivasan Murali and Giovanni De Micheli. Bandwidth-constrained mapping of cores onto noc architectures. In *Proceedings design, automation and test in Europe conference and exhibition*, volume 2, pages 896–901. IEEE, 2004.

[156] César Augusto Missio Marcon, Edson Ifarraguirre Moreno, Ney Laert Vilar Calazans, and Fernando Gehm Moraes. Comparison of network-on-chip mapping algorithms targeting low energy consumption. *IET Computers & Digital Techniques*, 2(6) :471–482, 2008.

[157] Jian Wang, Yubai Li, Song Chai, and Qicong Peng. Bandwidth-aware application mapping for noc-based mpsocs. *Journal of Computational Information Systems*, 7(1) :152–159, 2011.

[158] F Wang and Zhibiao Zhang. An adaptive genetic algorithm for mesh based noc application mapping. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 12(11) :7869–7875, 2014.

[159] Xinyu Wang, Haikuo Liu, and Zhigang Yu. A novel heuristic algorithm for ip

block mapping onto mesh-based networks-on-chip. *The Journal of Supercomputing*, 72(5) :2035–2058, 2016.

[160] Giuseppe Ascia, Vincenzo Catania, and Maurizio Palesi. Mapping cores on network-on-chip. *International Journal of Computational Intelligence Research*, 1(1) :109–126, 2005.

[161] Wenbiao Zhou, Yan Zhang, and Zhigang Mao. Pareto based multi-objective mapping ip cores onto noc architectures. In *APCCAS 2006-2006 IEEE Asia Pacific Conference on Circuits and Systems*, pages 331–334. IEEE, 2006.

[162] Cagkan Erbas, Selin Cerav-Erbas, and Andy D Pimentel. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *IEEE Transactions on Evolutionary Computation*, 10(3) :358–374, 2006.

[163] Rabindra Kumar Jena and Gopal K Sharma. A multiobjective evolutionary algorithm-based optimisation model for network on chip synthesis. *International Journal of Innovative Computing and Applications*, 1(2) :121–127, 2007.

[164] Rafael Tornero, Valentino Sterrantino, Maurizio Palesi, and Juan M Orduna. A multi-objective strategy for concurrent mapping and routing in networks on chip. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8. IEEE, 2009.

[165] Nadia Nedjah, Marcus Vinicius Carvalho Da Silva, and Luiza de Macedo Mourelle. Customized computer-aided application mapping on noc infrastructure using multi-objective optimization. *Journal of Systems Architecture*, 57(1) :79–94, 2011.

[166] Carlos A Coello Coello Coello and Gregorio Toscano Pulido. A micro-genetic algorithm for multiobjective optimization. In *International conference on evolutionary multi-criterion optimization*, pages 126–140. Springer, 2001.

[167] Ning Wu, Yifeng Mu, and Fen Ge. Ga-mmas : an energy-and latency-aware mapping algorithm for 2d network-on-chip. *IAENG Int J Comput Sci*, 2194(1) :1–6, 2012.

[168] Tao He and Yunfei Guo. Power consumption optimization and delay based on ant colony algorithm in network-on-chip. *Engineering Review : Međunarodni časopis namijenjen publiciranju originalnih istraživanja s aspekta analize konstrukcija, materijala i novih tehnologija u području strojarstva, brodogradnje, temeljnih tehničkih znanosti, elektrotehnike, računarstva i građevinarstva*, 33(3) :219–225, 2013.

[169] Navonil Chatterjee, Sheshivardhan Reddy, Shilpa Reddy, and Santanu Chattopadhyay. A reliability aware application mapping onto mesh based network-on-chip. In *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, pages 537–542. IEEE, 2016.

[170] Jaison Valmor Bruch, Eduardo Alves Da Silva, Cesar Albenes Zeferino, and Leandro Soares Indrusiak. Deadline, energy and buffer-aware task mapping optimization in noc-based socs using genetic algorithms. In *2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 86–93. IEEE, 2017.

[171] Antonio J Nebro, Juan J Durillo, Francisco Luna, Bernabe Dorronsoro, and Enrique Alba. A Cellular Genetic Algorithm for Multiobjective Optimization. page 12, 2006.

[172] Youcef Bouchebaba, Ali-Erdem Ozcan, Pierre Paulin, and Gabriela Nicolescu. Mpassign : a framework for solving the many-core platform mapping problem. *Software : Practice and Experience*, 42(7) :891–915, 2012.

[173] Robert P Dick, David L Rhodes, and Wayne Wolf. Tgff : task graphs for free. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign.(CODES/CASHE'98)*, pages 97–101. IEEE, 1998.

[174] Günther R Raidl, Jakob Puchinger, and Christian Blum. Metaheuristic hybrids. In *Handbook of metaheuristics*, pages 469–496. Springer, 2010.

[175] Christian Blum, Jakob Puchinger, Günther R Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization : A survey. *Applied soft computing*, 11(6) :4135–4151, 2011.

[176] Nadia Abd-Alsabour and S Ramakrishnan. Hybrid metaheuristics for classification problems. *Pattern Recognition-Analysis and Applications*, 10 :65253, 2016.

[177] Peio Loubiere. *Amélioration des métaheuristiques d'optimisation à l'aide de l'analyse de sensibilité*. PhD thesis, Paris Est, 2016.

[178] Antonio J Nebro and Juan J Durillo. jmetal 4.5 user manual. 2014.

[179] Juan J Durillo, Antonio J Nebro, and Enrique Alba. The jmetal framework for multi-objective optimization : Design and architecture. In *IEEE congress on evolutionary computation*, pages 1–8. IEEE, 2010.

[180] Moea framework. http ://moeaframework.org/. Consulté : 2019.

[181] Paradiseo. http ://paradiseo.gforge.inria.fr/. Consulté : 2019.

[182] Pisa. https ://sop.tik.ee.ethz.ch/pisa/ ?page=principles.php. Consulté : 2019.

Abstract : To meet the growing requirements of today's applications, multiprocessor architectures (MPSoCs) interconnected with a network on chip (NoC) are considered as a major solution for future powerful embedded systems. One of the most critical challenges of a NoC-based MPSoC is how to map an application on this platform. Due to the large solutions' research space generated by both the application complexity and the platforms, this mapping phase can no longer be done manually, hence il requires powerful exploration tools called DSE (Design Space Exploration Environment). In this thesis, we propose several approaches included in an exploration tool for solving the problem of static mapping of parallel applications on NoC-based heterogeneous MPSoC. This tool has many advantages : (1) it integrates several multiobjective optimization algorithms that can be specified in order to explore different solutions' spaces, mainly : exact methods, metaheuristics (P-metaheuristics and S-metaheuristics) as well as new hybrid ones proposed in this thesis ; (2) it offers different cost functions (defined using analytical or simulation models) and constraints. The user can specify them or define others easily (easily extensible) ; (3) it provides an easy way to evaluate the performance of the Pareto front returned by different algorithms using multiple quality indicators. We also present a series of experiments by considering several scenarios, therefore, we give designers guidelines on choosing the appropriate algorithm based on the characteristics of the mapping problem considered.
Keywords
Static Mapping, Multiobjective Optimisation, Network on Chip(NoC), Multi-processor System on Chip (MPSoC).

Résumé : Pour répondre aux exigences croissantes des applications actuelles, les architectures multiprocesseurs (MPSoCs) interconnectées avec un réseau sur puce (NoC) sont considérées comme une solution majeure des futurs systèmes embarqués de haute performance. L'un des défis les plus critiques d'un MPSoC basé NoC est de savoir comment placer (mapper) une application sur cette plateforme. En raison du grand espace de recherche de solutions engendré par la complexité conjointe des applications et de plateformes d'aujourd'hui, cette phase (mapping) ne peut plus être effectuée manuellement, d'où la nécessité d'outils d'exploration performants appelés DSE (Design Space Exploration Environment). Dans cette thèse, nous proposons plusieurs approches incluses dans un outil d'exploration pour la résolution du problème du mapping statique d'applications parallèles sur un MPSoC hétérogène basé NoC. Cet outil renferme plusieurs avantages : (1) il intègre plusieurs algorithmes d'optimisation multiobjectif pouvant être spécifiés afin d'explorer différentes solutions à savoir : des méthodes exactes, des métaheuristiques (les P-métaheuristiques et les S-métaheuristiques) ainsi que de nouvelles méthodes hybrides proposées dans le cadre de cette thèse ; (2) il offre de différentes fonctions de coûts (définies en utilisant un modèle analytique ou par simulation) et de contraintes. L'utilisateur peut les spécifier ou en définir d'autres et les intégrer facilement dans l'outil (facilement extensible) ; (3) il offre un moyen facile d'évaluer la performance du front retourné par différents algorithmes via plusieurs indicateurs de qualité. Nous présentons également une série d'expériences en considérant plusieurs cas de figures. Ainsi, nous donnons aux concepteurs des lignes directrices sur le choix de l'algorithme approprié selon les caractéristiques du problème du mapping considéré.
Mots clés
Placement statique, Optimisation multiobjectif, Réseaux sur puce, Multiprocesseurs sur puce.