Placement Dynamique D'Applications Embarquées Intensives sur des Réseaux de Processeurs sur Puce

Mohammed Kamal Benhaoua

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة وهران
Université d'Oran
كلية العلوم الدقيقة والتطبيقية
Faculté des Sciences Exactes et Appliquées

Département d'Informatique

THÈSE

pour l'obtention du
Diplôme De Doctorat En Sciences
(INFORMATIQUE)

Placement Dynamique D'Applications Embarquées Intensives sur des Réseaux de Processeurs sur Puce

par

MOHAMMED KAMEL BENHAOUA

soutenue publiquement le 11 Juin 2014

Jury

B. BELDJILALI	Pr. à l'Université d'Oran	**Président du Jury**
A. E. H. BENYAMINA	MCA à l'Université d'Oran	**Directeur de Thèse 1**
P. BOULET	Pr. à l'Université de Lille1	**Directeur de Thèse 2**
M. BENMOHAMED	Pr. à l'Université de Constantine	**Examinateur**
M. BENYETTOU	Pr. à l'Université USTO	**Examinateur**
M. SENOUCI	Pr. à l'Université d'Oran	**Examinateur**

Résumé

Aujourd'hui les utilisateurs demandent d'avoir des systèmes embarqués performants, capables d'offrir de grande puissance de calcul pour un large spectre d'applications, tout en respectant les contraintes du monde de l'embarqué. L'évolution des systèmes embarqués pose un problème au niveau des conceptions car ils doivent trouver un compromis entre leurs capacités (puissances de calcul, reconfigurabilité) et les différentes contraintes (surface de silicium, consommation). La solution à envisager pour le problème de puissance de calcul, est donc de passer à des systèmes multiprocesseurs (MPSoCs). Les systèmes multiprocesseurs sur puce (MPSoC) constituent une solution incontournable pour les exigences de performance des applications complexes embarquées futures. En outre, les réseaux sur puce ont vu le jour pour faire face aux limites des supports de communications telles que les bus, bus hiérarchiques, point à point. L'infrastructure d'interconnexion fondée sur les réseaux sur puce (NoC) est en passe de devenir une approche privilégiée pour faciliter la communication entre les éléments de traitements (PEs) des MPSoCs. Il est plus efficace d'intégrer plusieurs petits processeurs spécialisés, ou non, interconnectés par un réseau sur puce (NoC) dont la puissance de calcul et l'efficacité énergétique sont meilleurs plutôt que d'améliorer les performances d'un seul processeur. L'hétérogénéité des MPSoCs est également en augmentation en employant différents types d'éléments de traitements, afin de répondre aux exigences fonctionnelles et non fonctionnelles. Le flot de conception de ce type de système est complexe et est caractérisé par la co-modélisation. Une des étapes importante de ce flot de conception est l'association logiciel/matériel lors de l'exploration architecturale. Durant cette phase d'association il ne suffit pas d'affecter les tâches (codes et données) aux ressources physique du MPSoC (processeurs, liens de communication, mémoire, etc.) mais de chercher une affectation qui doit satisfaire certaines exigences (objectifs à optimiser, contraintes à satisfaire). Ce problème d'affectation est dit placement de tâche (Mapping). Les décisions de placements peuvent considérablement influencer les performances du système. Selon le moment où il est défini, le placement des tâches peut

être classé comme statique (off-line), ou dynamique (on-line). Dans le premier cas, le placement des tâches est défini au moment de la conception, ses techniques ne sont pas appropriées pour les charges de travail de scénario dynamique en raison de leur complexité et du temps d'exécution conséquent. Pour faire face aux limites des techniques de placement statique, le vrai challenge est de passer aux techniques de placement dynamique durant l'exécution. Le but de notre travail est d'étudier les techniques de placement dynamique qui existent et d'essayer de proposer d'autres méthodes permettant d'optimiser les coûts (consommation d'énergie, temps d'exécution, latence, etc.) et de répondre aux contraintes du monde de l'embarqué, tout en prenant en considération l'optimisation des communications. Dans cette thèse, quatre techniques efficaces ont été proposées pour réaliser des algorithmes de placement dynamique pour les plateformes MPSoC hétérogènes basée NoC. Afin d'évaluer les performances de ces techniques, et vu le manque de simulateurs libres et à haut niveau d'abstraction qui pourraient nous aider à tester et à valider nos propositions algorithmiques, la nécessité de développer un simulateur à haut niveau d'abstraction s'avère impérative dans notre travail. En premier, nous proposons une nouvelle plateforme de tests et de simulations qui permet de simuler n'importe quelles plateformes MPSoCs hétérogènes, homogènes et à n'importe quelles dimensions dont les processeurs sont interconnectées via un réseau sur puce. Le simulateur permet aussi d'exécuter, de comparer les différents algorithmes de placement existants avec ceux qui seront proposés dans cette thèse et de simuler le trafic sur des réseaux sur puce en vue d'étudier leurs performances. Il permet aussi de supporter des plateformes mono-tâche c'est à dire que chaque unité de traitement ne permet l'exécution que d'une seule tâche. Dans un deuxième travail une nouvelle stratégie de packing en spirale a été proposée pour le placement des différentes tâches d'une même application sur le NoC cible. Dans ce placement on essaye de placer les tâches qui communique le plus sur deux processeurs voisins est libres affin de réduire les coûts des communications. En plus, une technique de placement des communications en dynamique basée sur l'algorithme du plus court chemin de Dijkstra modifié, a été proposée et implémentée. Les évaluations par simulation de nos deux premières pro-

positions ont montré de bons résultats par rapport aux techniques existantes. Dans un troisième travail, proposé ici, nous ne contentons pas seulement d'optimiser le placement des tâches, mais de chercher parmi les techniques permettant de le réaliser celle nécessitant le plus petit temps de recherche. Autrement dit, au contraire du placement statique, le solveur du placement dynamique, a son code embarqué avec celui de l'application à exécuter. Sans oublier que le temps de recherche de la solution optimale (par le placement dynamique) doit être ajouté au temps totale d'exécution de l'application. Ce qui nous amène à trouver la technique de placement dynamique permettant de trouver une ou plusieurs solutions optimales de bonne qualité et en un temps très court. Pour répondre à cet objectif nous proposons la stratégie de packing basée sur la distance de Manhattan pour le placement des tâches d'application sur des processeurs voisins ayant un temps de recherche minimum par rapport aux techniques existantes. Vu que la plupart des techniques proposées pour le placement des communications dans la littérature dans ce type de système sont statiques, une autre technique de placement dynamique des communications multi-objectifs (MORA) qui vise à minimiser le temps d'exécution et la consommation d'énergie a été proposée. Les résultats obtenus par simulation sur différents types d'applications : applications générées par l'outil TGFF, applications : Multi-Window Display (MWD), Video Object Plane Decoder (VOPD), Pecture-In Picture (PIP) et multiples applications MPEG4 ont montrés que nos propositions sont bénéfiques tout en offrant des gains considérables.

Remerciement

Ce document reflète l'aboutissement de plusieurs années de travaux de thèse entamés entre les universités d'ORAN et Lille1. Ce résultat n'est qu'une étape dans le processus de recherche qu'on a réellement lancé depuis plusieurs années avec P. Boulet et A. E. H. Benyamina dans l'équipe DART de l'université de Lille1 et LAPECI de l'université d'Oran. Mon plus grand espoir est que les propositions présentées ici pourront être utiles en servant de base à nos futurs travaux et à d'autres propositions scientifiques plus avancées.

Je tiens à remercier vivement mon Co-encadreur le professeur Pierre Boulet de m'avoir fait confiance et de m'avoir encadré dans ce thème. Sa générosité scientifique et sa disponibilité m'ont permis d'avancer dans mes travaux et de surmonter avec lui toutes les difficultés.

Mes remerciements vont aussi, avec la même intensité, à mon Encadreur Mr Benyamina Abbou El Hassen qui n'a jamais cessé de croire en moi. Sa confiance témoignée et ces critiques objectives. Il s'est montré toujours disponible pour m'apporter conseils et motivation. Plus qu'un encadrant, un grand frère qui m'a dirigé dans mes travaux en définissant mes priorités ainsi que mes objectifs. Grâce à ses encouragements, ses conseils judicieux et son aide pour rédiger ce manuscrit que notre but est atteint.

Je tiens à remercier vivement le Dr. A. Kumar et Dr. A. Singh d'avoir accepté la collaboration avec notre laboratoire LAPECI. Je tiens à remercier en particulier Dr. A. Singh pour le temps qui m'a réservé durant plus d'une année. Ces débats ont été fructueux pour moi, ce qui m'a permet de bien ce positionnée sur cet axe de recherche.

Merci au Professeur B. Beldjilali qui nous a fait l'honneur d'accepter de présider le jury de soutenance. Merci aux professeurs M. Benmohamed et M. Senouci et Dr. M. Benyetou d'avoir accepté d'examiner ce travail.

Enfin je tiens à exprimer ma profonde gratitude à l'équipe DART de LIFL, qui

avec P.Boulet m'ont accueilli au sein de leur équipe en mettant à ma disposition tous les moyens de l'équipe. Enfin, à toutes celles et tous ceux qui ont contribué de prés ou de loin à l'accomplissement de ce travail.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Révolution des Multiprocesseurs sur puce (MPSoCs)

L'évolution extraordinaire des systèmes électroniques modernes nous a permis d'entrer dans l'ère des Systèmes Multiprocesseurs sur puce (MPSoC). Permettant le traitement d'applications complexes. En 1965, la loi de Moore prédisait que le nombre de transistors dans la même surface de la puce allez croître de façon exponentielle [5]. Cette tendance, illustrée à la figure.1.1 à permis l'augmentation des performances des appareils électroniques telles que la vitesse de traitement, la capacité de la mémoire, le nombre de pixels sur un écran d'affichage, etc. Les premières conséquences de cette tendance sont d'ordre matériel et logiciel :

— D'une part, les concepteurs du matériel sont en mesure actuellement de fournir des moyens de traitement plus efficace et plus rapide,

— d'autre part, les développeurs d'applications doivent maximiser l'utilisation de la puissance de traitement par une accélération des flots de conception et de réalisation du logiciel.

La haute densité d'intégration permettait la mise en œuvre de plusieurs processeurs sur une seule puce a entrainé le développement des MPSoC [6]. Le processeur

d'Intel, d'abord publié en 1971, 4004, comptait environ 2.300 transistors. Ce processeur fonctionne à la vitesse de 400 KHz. En revanche, un processeur à cœur unique par exemple, Intel Pentium 4 a plus d'un milliard de transistors, fonctionnant à plus de 3 GHz. Les constructeurs ont été obligés de limiter la fréquence maximale du processeur et la transition vers la conception de puces de plusieurs processeurs, fonctionnant à des fréquences plus basses. Dans la figure.1.1, il est intéressant d'observer l'introduction du processeur Dual-core, à partir de 2005 : C'est le début de l'ère multi-cœurs. En outre, la complexité croissante des applications temps-réels, ne peut être traitée simplement en augmentant la fréquence d'un processeur à cœur unique. Au lieu de celà, il a besoin de plusieurs processeurs cohérents, pouvant communiquer et fournir un parallélisme accru. Le concept fondamental est de considérer l'application constituée de nombreuses petites tâches qui peuvent être distribuées efficacement sur plusieurs processeurs afin d'être exécutées en parallèle et ainsi de répondre à la demande croissante de performances des applications complexes.

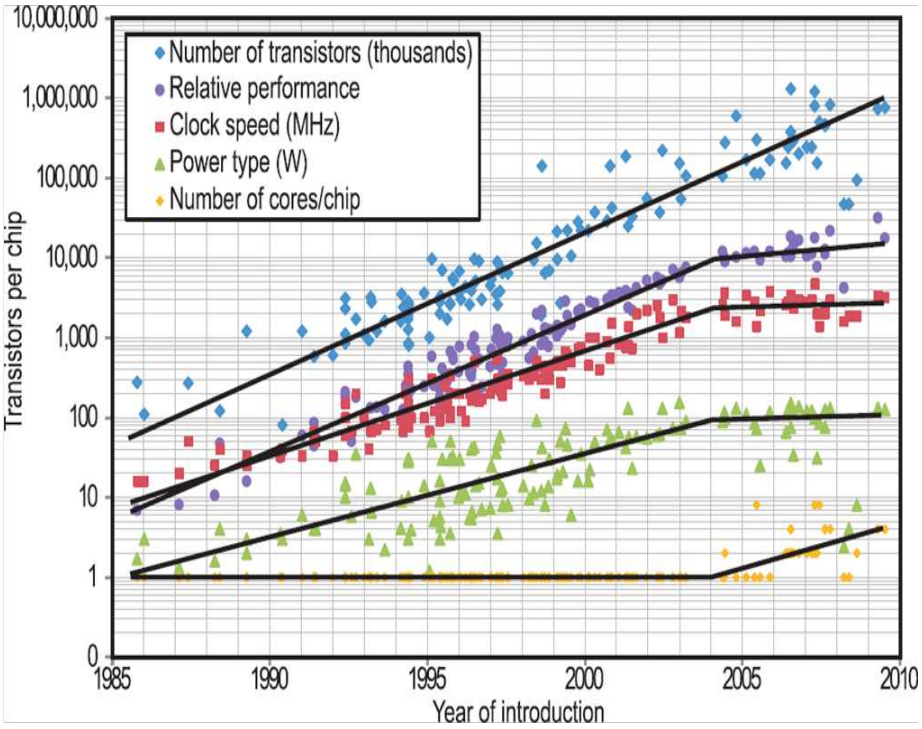


FIGURE 1.1 – Poursuite de la loi de Moore

1.2 Motivation

C'est un fait bien connu, que la personnalisation d'un seul processeur pour l'application, peut améliorer les performances. Toutefois, les exigences de performance des applications embarquées complexes modernes ont considérablement augmentées, ce qui rend incontournable l'utilisation de plusieurs processeurs offrant un parallélisme accru. L'exigence de communication d'un grand nombre de processeurs, peut être satisfaite par l'efficacité des réseaux sur puce (Networks-on-Chip (NoC)). Les processeurs eux-mêmes doivent être de types différents pour exécuter différentes tâches de manière efficace afin d'atteindre une meilleure performance en présence de blocs matériels reconfigurables dans les MPSoCs hétérogènes. Ces blocs de matériel peuvent être configurés au moment de l'exécution en fonction de la fonctionnalité nécessaire à calculer des tâches intensives en offrant également une grande flexibilité au même niveau que celui des processeurs à usage général. L'accélération fournie par le matériel peut être utilisée pour satisfaire les exigences de performances imposées. De même, il est possible de configurer d'autres types de processeurs afin de les exploiter au maximum de leur capacité. Ainsi, Les MPSoCs hétérogènes seront nécessaires à mesure que la performance demandée augmente. Les systèmes embarqués modernes (par exemple, les téléphones intelligents, PDA, tablettes) emploient des MPSoCs afin de prendre en charge plusieurs applications en même temps : par exemple, un téléphone intelligent peut être utilisé pour afficher une image en utilisant une application de décodage JPEG sur Internet et en même temps pour écouter de la musique en utilisant une application de décodage MP3. Les composants (tâches et leurs communications) d'applications doivent être placés et ordonnancés efficacement sur les ressources du MPSoC, afin de répondre à des contraintes de performance pour chaque application. Le problème de placement et d'ordonnancement est similaire à un problème d'affectation quadratique connue comme un problème NP-difficile [7]. Par conséquent, trouver une solution optimale répondant à toutes les contraintes données est très difficile et prend du temps. Ainsi, en explorant toutes les tâches pour des combinaisons de ressources, de manière exhaustive et ensuite choisir la combinaison optimale, peut prendre des jours ou des semaines pour un

grand nombre de tâches et de processeurs. C'est pourquoi, les heuristiques basées sur la connaissance du domaine d'application, doivent être prises en compte pour trouver une solution approchée (presque optimale). Le placement d'applications sur une plateforme MPSoC peut être accompli soit au moment de la conception soit lors de l'exécution : Les techniques de placement au moment de la conception (design-time) ne conviennent que pour les scénarios à charge de travail statique et sont donc incapables de gérer le dynamisme dans les applications à charge de travail dynamique (run-time) (telles les applications réseau et multimédia). En effet, souvent les applications sollicitées sont de type de téléchargement multimédia, applications Java dans un téléphone mobile ou d'autres applications prisent en charge durant l'exécution. Pour ce type d'applications (dites dynamiques car sollicitées durant l'exécution) le placement doit lui aussi se faire pendant l'exécution, ce qui explique sa désignation par "placement dynamique". Les techniques, pour ce type de placement, doivent affecter de nouvelles applications sur les ressources de la plateforme avec une connaissance précise de leur occupation afin de satisfaire leurs exigences de performance. Le placement dynamique de nouvelles applications peut être considéré, selon les différents types de scénarios utilisés, avec ou sans résultats pré-analysés. Lorsque les applications qui doivent être prises en charge sur une plateforme ne sont pas connues au moment de la conception, aucune pré-analyse n'est nécessaire. Cela exige des heuristiques efficaces à définir pour placer de nouvelles tâches sur les ressources de la plateforme et pour effectuer tous les traitements au moment de l'exécution. Elles ne peuvent pas garantir les échéances temporelles strictes en raison de l'absence de toute analyse préalable et la puissance de calcul limitée au moment de l'exécution. Toutefois, ces heuristiques sont indépendantes de la plateforme car n'utilisant pas les résultats d'analyse spécifiques de la plateforme calculés à l'avance. Les applications qui doivent être prises en charge par la plateforme, devraient être connues au moment de la conception afin de les placer grâce aux résultats pré-analysés. Dans de tels cas, des heuristiques légères sont nécessaires pour sélectionner les placements les plus efficaces pour chaque application au moment de la conception (offline) à partir des placements analysés et stockés sur le système. La

sélection devrait être imposée aux ressources système disponibles et la performance souhaitée. Les placements devraient contenir les allocations et ordonnancements et être sélectionnés pour configurer la plateforme. L'analyse au moment de la conception doit effectuer tout le traitement de calcul intensif, ce qui facilite au gestionnaire (Manager) de plateforme dynamique la configuration des applications de manière efficace. L'analyse au moment de la conception pour explorer les placements, à savoir l'affectation exhaustive des tâches aux ressources, n'est pas réalisable dans un temps limité pour des applications et plateforme de grande taille, par conséquent, des stratégies d'analyse plus rapides que d'explorer tous les placements efficaces doivent être développées en prenant en compte les spécifications de la plateforme pour effectuer l'exploration, de sorte que les résultats d'analyse ne seront pas applicables à toutes les plateformes.

1.3 But de la thèse

L'objectif principal de cette thèse est de développer des techniques et des méthodes efficaces pour le placement dynamique d'applications embarquées sur des plateformes MPSoC hétérogènes basées NoC afin de maximiser les performances. Ce choix a été fait dans le souci de placer des applications quelconques sur des plateformes les plus générales possibles et extensibles (scalables).

1.4 Principales contributions de la thèse

Nous donnons un aperçu des principales contributions qui ont été faites au cours de cette thèse et qui peuvent être résumées comme suit :

— Réalisation d'une plateforme de simulation qui permet de simuler n'importe quelle plateforme multi-cœur interconnectée par un réseau sur puce homogène ou hétérogène. Elle permet aussi d'exécuter et de tester les propositions algorithmiques de placement dynamique d'applications sur des plateformes simulées.

— Heuristique de placement dynamique en spirale pour le placement efficace

des tâches d'applications sur une plateforme MPSoC hétérogène basée NoC contenant des éléments de traitement (PE) qui ne supportent qu'une seule tâche. La technique proposée place les tâches d'applications sur les éléments de traitement MPSoC de manière à ce que toutes les tâches esclaves par maître soient placées le plus proche possible de la tâche maître (les tâches esclaves sont placées autour de la tâche maître), conduisant à un placement des tâches efficaces.

— Une autre heuristique de placement des communications est proposée afin de réduire les coûts des communications. Cette technique est basée sur l'algorithme Dijkstra modifié afin de trouver le chemin le plus court et le moins chargé afin d'optimiser les coûts des communications. Ces techniques montrent une amélioration des performances par rapport aux techniques existantes.

— Heuristique de placement dynamique pour le placement efficace d'applications sur une plateforme MPSoC hétérogène basée NoC contenant des éléments de traitement (PE) qui ne supportent qu'une seule tâche. On constate que les techniques de placement existantes ne peuvent pas fournir de bons résultats dans le cas des applications plus larges (applications qui contiennent un nombre de tâches important). La technique proposée place les tâches d'applications sur les éléments de traitement MPSoC de manière très systématique. L'approche proposée utilise une stratégie de packing basée sur la distance de Manhattan et met le point sur le temps de placement (le temps de recherche d'une ressource libre) qui peut influencer les performances du système. Ce temps est minimisé d'une façon considérable pendant que les performances sont optimisées.

— La plupart des travaux existants traitant la même problématique, utilisent pour le placement des communications, le routage statique à savoir le XY, pour sa simplicité d'implémentation. Nous avons proposé une heuristique de routage multi-objectifs (MORA) pour le placement dynamique des communications. Le but du MORA est d'acheminer les paquets échangés par le lien le moins chargé sur les liaisons point à point et vers la destination de façon à réduire

les coûts des communications. L'approche MORA montre une amélioration des performances par rapport aux techniques existantes à savoir le routage statique XY.

1.5 Organisation de la thèse

Le reste de la thèse est organisé comme suit :

— **Chapitre 2 : État de l'Art**

Ce chapitre présente une synthèse sur l'importance des architectures MPSoC basée NoC dans les systèmes embarqués modernes. Les défis et méthodologies pour concevoir les MPSoCs ainsi que les techniques et les méthodologies de placement d'applications sur les architectures MPSoCs basée NoC seront présentées. Par la suite, les stratégies de placement existantes sont analysées pour mettre en évidence leurs limites et les stratégies les plus appropriées sont identifiées pour une éventuelle investigation.

— **Chapitre 3 : Simulateur de placement dynamique d'applications Embarquées**

Dans ce chapitre, Nous donnons une vue globale sur les simulateurs NoCs, puis un état de comparaison. Par la suite nous détaillons notre simulateur de placement dynamique d'applications sur une architecture MPSoC basée NoC : conception, modélisation, fonctionnement, etc. L'extension du simulateur afin de supporter une simulation de plateforme multi-tâches sera considérée dans notre futur publication.

— **Chapitre 4 : Heuristiques pour le routage et le placement de tâche dynamique en spirale sur des MPSoCs hétérogènes basée NoC**

Dans ce chapitre des définitions sur les modèles de calcul seront présentées. Le modèle utilisé Maître-esclave sera détaillé et l'architecture MPSoC basée NoC mono tâche. Comme contribution dans ce chapitre notre premier papier publié intitulé "Heuristics for Routing and Spiral Run-time Task Mapping in NoC-based Heterogeneous MPSOCs" est détaillé. La contribution est basée sur un placement mono-tâche en spirale et placement des communications basé sur

l'algorithme dijkstra modifié.

— **Chapitre 5 : Heuristiques pour le placement dynamique de tâche et communications sur des MPSoCs hétérogènes basée NoC**

Ce chapitre sera consacré aux définitions des modèles de calcul utilisés, l'architecture MPSoC basée NoC mono-tâche. Comme contribution dans ce chapitre notre papier publié intitulé "Heuristics For Dynamic Task And Communications Mapping In NoC-Based Heterogeneous MPSoCs" sera détaillé. La contribution est basée sur un placement mono-tâche en utilisant une stratégie de packing Manhattan afin de minimiser le temps de placement en plus de la prise en charge des communications par la proposition d'une heuristique de placement dynamique des communications multi-objectifs (MORA).

1.6 Liste des publications résultant

Le travail présenté dans cette thèse a été publié dans les revues internationales suivantes :

Revues Internationales

[J-1] The Mediterranean Journal of Computers and Networks, Vol. 9, No 4, October 2013, ISSN : 1744-2397 **"Heuristics For Dynamic Task And Communications Mapping In Noc-Based Heterogeneous Mpsocs"** M. K. Benhaoua, A. K. Singh, A. E. H. Benyamina, A. Kumar, P. Boulet.

[J-2] IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, No 1, July 2013 ISSN (Print) : 1694-0814 | ISSN (Online) : 1694-0784 : **"Heuristics for Routing and Spiral Run-time Task Mapping in NoC-based Heterogeneous MPSOCs"** Abbou El Hassen Benyamina, Mohammed kamel Benhaoua and Pierre Boulet.

Chapitre 2

État de l'Art

Sommaire

Dans ce chapitre, nous examinons les tendances des systèmes MultiProcesseurs sur puce à la section 2.1. La section 2.2 présente quelques plateformes MPSoC homogènes et hétérogènes existantes, ciblées pour différents domaines d'applications embarquées. Dans la même section, nous discuterons sur l'interconnexion sur puce nécessaires pour répondre aux besoins en communication des différentes PEs et les challenges dans la conception des plateformes MPSoC avec diverses méthodologies de conception existants. Diverses techniques de placement proposées dans la littérature plaçant les applications sur les plateformes MPSoC seront discutées à la section 2.3. Une étude approfondie de ces techniques est réalisée pour mettre en évidence leurs limites, ce qui nous a donné la motivation principale de cette thèse.

Les systèmes embarqués modernes (par exemple, les téléphones intelligents, PDA, tablettes) emploient les Systèmes MultiProcesseurs sur puce (MPSoCs) afin de satis-

faire les exigences de performance toujours croissante des applications embarquées complexes modernes, tout en réduisant la consommation d'énergie. Par conséquent, les plateformes MPSoC constituées de plusieurs processeurs embarqués sont plus présentes dans les traitements embarqués [8]. Ces plateformes peuvent augmenter les performances par l'exécution parallèle des tâches d'applications sur des processeurs différents.

En outre, les processeurs fonctionnent à des fréquences inférieures, contrairement aux systèmes à cœur unique basé sur un processeur où la fréquence est très élevée, répondant ainsi à l'exigence de faible puissance. Intel rapporte que sous horloge (under-clocking) de 20 % d'un seul cœur économise la moitié de l'énergie en sacrifiant juste 13 pour cent de la performance [9]. Donc, si le travail est réparti sur deux processeurs fonctionnant à 80 % de la fréquence d'horloge, on obtient 74 % de performances meilleurs pour la même énergie. Cependant, la chaleur est dissipée en deux points plutôt qu'un seul. Les plateformes MPSoC sont tenues à contenir un plus grand nombre d'éléments de traitement (PEs) vu que la technologie progresse. Les plateformes peuvent être homogènes ou hétérogènes selon le type de PEs présents dans la plateforme. Les plateformes homogènes contiennent des PEs identiques, ce qui les rend très appropriées pour la mise en œuvre VLSI. D'autre part, les plateformes hétérogènes contiennent différents types d'éléments de traitements (PEs) qui satisfont les exigences de performance plus élevée en exploitant des caractéristiques distinctes des PEs.

La plateforme d'éléments de traitement (PEs) nécessite une infrastructure de communication pour avoir une bonne communication entre plusieurs éléments de traitement (PEs). Le placement des applications sur la plateforme de multiples éléments de traitement (PEs) utilisant des techniques de placement efficace, est un sujet de recherche très actif traité par plusieurs organismes de recherche [10, 11].

2.1 Tendances des Multiprocesseurs sur puce (MPSoCs)

Cette section décrit les tendances dans les MPSoC, cette technologie a évolué et la demande de performances est croissante.

2.1.1 Nombre de cœurs de traitement

En continuant avec la loi de Moore, le nombre de transistors va croître de façon exponentielle, ainsi que le nombre de cœurs à peu près. En outre, International Technology Roadmap for Semiconductors (ITRS) prévoit que cette tendance se poursuivra comme le montre la figure.2.1. Ainsi, comme la nanotechnologie évolue, il deviendra possible d'intégrer des milliers de cœurs sur la même puce comme prédit par les sources telles qu'Intel et Berkeley [12, 13]. Les cœurs sont envisagés comme des portes logiques du 21ème siècle.

Presque toutes les sociétés d'informatique ont annoncé les puces avec multiples cœurs de processeurs. En outre, les fournisseurs de feuilles de route assure que le nombre de cœurs par puce se double à plusieurs reprises. Ces puces doivent être utilisées à l'avenir et sont diversement appelées MultiProcesseurs, Multi-Core et Multi-Cœurs sur puce. Les systèmes complets sont généralement appelés MPSoCs.

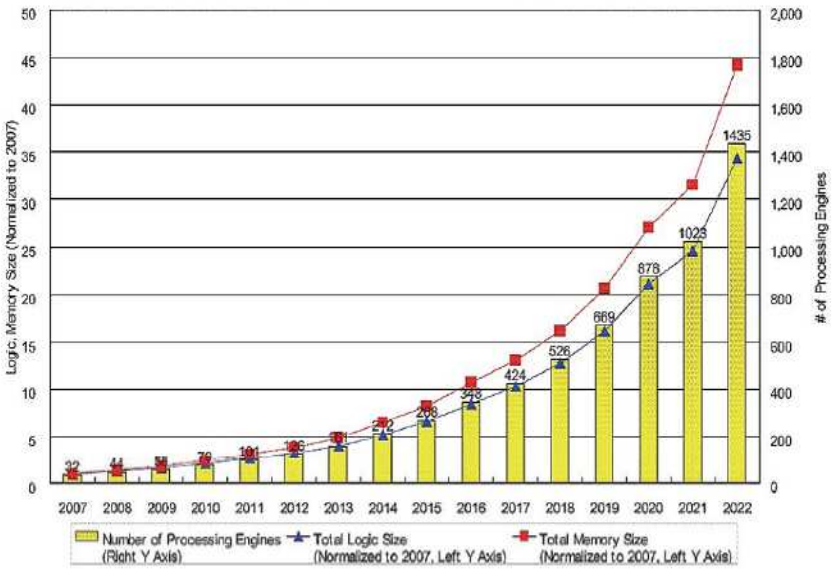


FIGURE 2.1 – ITRS Roadmap montre le nombre croissant de cœurs de traitement [1]

2.1.2 Réseau sur puce (NoC) pour la scalabilité

Les processeurs présents dans les MPSoCs font appel à une certaine infrastructure pour avoir une bonne communication entre eux. Celle ci peut être basée sur les bus, les liaisons point à point ou les réseaux sur puce (NoC) [14]. Dans les infrastructures à base de bus, comme le nombre de processeurs augmente, le goulot d'étranglement de l'arbitrage augmente d'où le besoin de hausse du nombre de bus maîtres. En plus, la bande passante du bus est partagée par tous les processeurs connectés, ce qui le rend non évolutif (scalable). Dans des liaisons point à point, avec l'augmentation du nombre de processeurs, des fils plus longs sont nécessaires, ce qui provoque des retards dans la communication. Ainsi, cette infrastructure est aussi non évolutive. Cependant, c'est dans un réseau sur puce (Network On chip (NoC)) que l'arbitrage est distribué et que des segments de fil sont obligatoires. De plus, la bande passante se redimensionne à la taille du réseau, c'est à dire le nombre de processeurs. Ainsi, l'infrastructure de communication NoC est efficace et hautement évolutive [15, 16].

2.1.3 Hétérogénéité dans les cœurs de traitement

La loi d'Amdahl évalue les bénéfices réels du calcul multi-cœur [17]. Il indique que l'accélération de l'application par le traitement MPSoC est limitée par le temps nécessaire pour l'exécution de la partie séquentielle de l'application. La figure 2.2 montre la vitesse obtenue en utilisant un nombre de processeurs différents à divers niveaux de parallélisations. Il est clair que si 5% de l'application ne peut être parallélisée (95% parallélisée), l'accélération maximale obtenue est multipliée par 20, même si un plus grand nombre de processeurs est utilisé. L'accélération peut être augmentée par l'exécution accélérée de la partie non parallélisée (5%) ou par l'exécution de la partie séquentielle en temps minime à l'aide d'un processeur à performance séquentielle meilleure.

La loi de Gustafson [18] est une loi théorique qui permet d'obtenir une borne maximale au gain que l'on peut avoir en utilisant plusieurs processeurs pour paralléliser du code via le parallélisme de données.

Cette loi donne le gain que l'on peut avoir avec N processeurs en exécutant

un programme parallélisé dans le cas où il est possible d'augmenter la quantité de données traitées, contrairement à la loi d'Amdahl, qui s'applique à quantité de données fixe. Elle traduit le fait qu'on peut traiter plus de données en augmentant le nombre de processeurs.

Elle dit que le gain de vitesse est égal à $S + (N \text{ x } P)$, avec :

S : le pourcentage de code non parallélisable.

P : le pourcentage de code parallélisable.

N : le nombre de processeurs.

Quand la plateforme MPSoC hétérogène est utilisée, les caractéristiques distinctes des différents types de processeurs peuvent être exploitées par les différentes parties de l'application, afin d'augmenter l'accélération. Ainsi, les MPSoCs hétérogènes sont devenus des alternatives de calculs formidables où les preuves ont été montrées avec l'amélioration apportée sur les grandes applications par rapport aux plateformes homogène. En outre, les MPSoCs hétérogènes peuvent contenir des processeurs à usage général (GPP) pour la flexibilité, des accélérateurs pour le calcul des tâches intensives, des blocs matériels reconfigurables pour l'efficacité, la flexibilité et le calcul de traitement intensif et des processeurs spécialisés comme les processeurs de traitements de signaux numériques (DSP) pour les tâches de traitement de signal, offrant ainsi une flexibilité, l'augmentation des performances de calcul et la réduction de la consommation d'énergie en même temps. À l'avenir, l'hétérogénéité peut être encore augmentée pour atteindre les exigences de performance élevée des applications complexes.

2.2 Architectures Multiprocesseurs sur puce

Un système MultiProcesseurs sur puce (MPSoC) peut être vu comme un ensemble de deux parties. Une matérielle constituée de plusieurs processeurs et l'autre software appelée système sur puce. L'approche d'architecture Multi-Cœurs peut être adoptée pour mettre en œuvre les éléments de traitements dans les Cœurs, comme ils ont montré leurs succès pour la plupart des domaines d'applications [8]. Ces Cœurs de

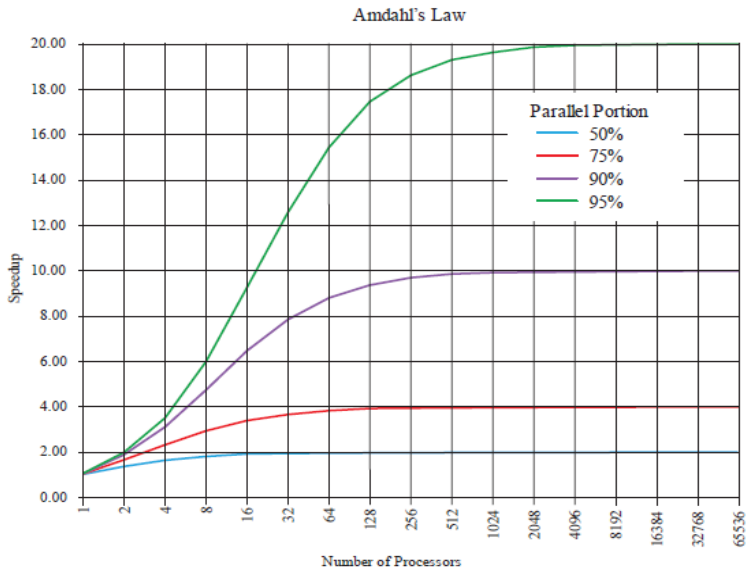


FIGURE 2.2 – La loi d'Amdahl indiquant que l'accélération obtenue en utilisant plusieurs processeurs est limitée par la partie séquentielle du programme [2]

traitement peuvent être mis en œuvre dans une puce afin de développer un MPSoC. L'architecture Multi-Cœur a un certain nombre d'avantages [8] :

- La méthode la plus appropriée pour les futures technologies de traitement : plus de Cœurs seront disponibles avec l'avancement de la technologie mais la complexité des Cœurs ne peut augmenter.
- Les petits Cœurs peuvent être optimisés largement.
- Échelles de performance de calcul fonctionnant presque linéairement avec le nombre de Cœurs.
- Certains Cœurs peuvent être commutés ON/OFF en fonction des exigences.
- Les Cœurs défectueux peuvent être écartés pour rendre le concept Multi-Cœur tolérant aux pannes.
- Plusieurs Cœurs peuvent être configurés en parallèle pour améliorer les performances.
- Le domaine d'horloge individuelle par Cœur est possible, et la reconfiguration dynamique partielle éventuelle sur la base d'un Cœur pour les Cœurs reconfigurables.

2.2.1 Architectures homogènes

Tous les Cœurs présents dans une architecture MPSoCs homogènes sont identiques. Ainsi, les Cœurs peuvent être facilement reproduits, ce qui les rend très appropriés pour l'implémentation VLSI. De plus, ils sont faciles à programmer par rapport aux plateformes hétérogènes. Les recherches Académiques proposent souvent des architectures MPSoCs homogènes et les plus reconnues sont introduites dans cette partie. Massachusetts Institute of Technology a proposé une architecture de 16 Cœurs de processeur Raw Architecture Workstation (**RAW**) [19]. 167 Cœurs de processeurs Asynchronous Array of Simple Processors (**AsAP**) a été proposé par University of California Davis [20]. L'université de Texas d'Austin proposes Tera-op, Reliable, Intelligently adaptive Processing System (**TRIPS**) qui utilise 32 chips chacun contenant 2-Cœurs [21]. Le processeur **WaveScalar** a été proposé par l'université de Washington qui contient approximativement 2K d'éléments de calcul simple (PEs) arrangés dans 16 clusters [22]. Dans [23], un MPSoC scalable pour l'architectures de future génération a été proposé. Cette architecture est basée sur un processeur RISC et une mémoire distribué. Cette puce a été développée pour offrir des performances élevées. Récemment, les sociétés Intel et Tilera ont proposé des MPSoCs homogènes [23, 24]. Dans [23], Intel propose un MPSoC homogène constitué de 80 Cœurs reliés entre eux par un réseau d'interconnexion où chacun contient deux unités flottantes. Le réseau d'interconnexion est disposé en tant que matrice 2D de 10x8 dans une zone de 275mm qui contient 80 Cœurs et routeurs à commutation par paquets, fonctionnant à 4 GHz. Le MPSoC est conçu en technologie de 65nm. Intel a également mis sur le marché **Core i3**, **Core i5** et **Core i7**, une famille de processeurs Multi-Cœurs pour ordinateurs de bureau et mobiles [25]. Dans [24], la société Tilera a annoncé « **TILE-Gx100** », le premier 100 Cœurs dans le monde, à processeurs d'usage général, qui offre des performances élevées parmi tous les microprocesseurs déjà annoncés par un facteur de quatre. Ils ont également simplifié la programmation Multi-Cœurs avec leur percée « **Environnement de développement Multi-Cœurs (MDE)** » qui dispose d'un déploiement rapide des produits. Hewlett-Packard a annoncé un MPSoC composé de plusieurs Cœurs

de processeurs MIPS [26]. Kalray [27] propose un MPSoC, le MPPA-256 (Multi-Purpose Processor Array) qui renvoi à l'idée d'un processeur multi-fonctions et non spécialisé type FPGA, il est avant tout destiné aux applications dont le critère numéro un est la puissance par watt et orientées basse consommation puisqu'on parle d'une consommation typique de 5 W (10 Watts au maximum). Mais à un niveau de performances bien supérieur à celui d'un FPGA. MPPA intègre 256 cœurs VLIW (Very Long Instruction Word) 32-bit organisés en 16 groupes de 16 cœurs chacun (+ 1 cœur système). la société propose un environnement logiciel adéquat le MMPA Accesscore, un SDK utilisant un langage de programmation de haut niveau proche du C. Il y a beaucoup d'autres fabricants qui ont annoncé leurs puces Multi-Cœurs ciblant différents domaines de calcul tel que le calcul scientifique, embarqués et à usage général. Certaines puces Multi-Cœurs sont répertoriés dans [25].

2.2.2 Architectures hétérogènes

Les MPSoCs hétérogènes sont constitués de différents types d'éléments de traitement (PEs) implémentés dans les Cœurs. Les éléments de traitements peuvent être des processeurs à usage général (*GPPs*), des éléments de traitements spécialisés comme des *DSP*, des *FPGA*, des *IPS* (intellectual property dédiés), des mémoires spécialisées, etc. En exploitant les caractéristiques distinctes des différents types de PEs, la performance de calcul peut être augmentée pendant que la consommation d'énergie peut être diminuée tout en maintenant l'extensibilité de l'architecture.

Actuellement, les chercheurs universitaires visent aussi les architectures MPSoCs hétérogènes. Celles proposées par les universités sont présentées dans [28, 29, 30, 31]. Nollet et al. [28] présente un MPSoCs contenant quatre différents types de PEs : GPPs, DSPs, des accélérateurs et des blocs matériels reconfigurables. Les différents PEs sont reliés entre eux par un réseau maillé 3x3. Smit et al. [29] proposent des architectures reconfigurables où les PEs sont reliés par un réseau sur puce et une architecture particulière nommée « *Annabelle* ». Dans les puces intelligentes pour le projet [30] Environnement intelligents (4S) à l'Université de Twente, une architecture MPSoC dynamiquement reconfigurable a été proposée : le Cœur peut être

à unité reconfigurable au niveau bit (par exemple, FPGA) au niveau mots (par exemple Montium [32]), ou programmable à usage général (GPP). La programmabilité des Cœurs dans les architectures reconfigurables aide le système à être la cible de plusieurs domaines d'application. Arpinen et al. [31] présente un MPSoC composé de plusieurs Cœurs de processeurs Altera Nios II soft-core et accélérateurs matériels personnalisés, reliés par un réseau de communication appelé interconnexion de blocks IP hétérogène (*HIBI*). Le MPSoC est créé sur un FPGA Altera [33]. Deux MPSoCs hétérogènes : *CHAMELEON* et *Pleiades* sont proposés par les universités de Twente et l'UC Berkeley, respectivement. Ces MPSoCs visent les algorithmes DSP. Le CHAMELEON SoC contient le processeur à usage général (Core ARM), la partie reconfigurable à grain fin (Core FPGA) et la partie reconfigurable à gros grains (Core montium [32]). Les parties de calcul régulier d'un algorithme sont exécutées dans des parties reconfigurables et les parties irrégulières sur le processeur à usage général. Les Chips *Pléiades* contiennent des processeurs à domaines spécifiques à faible énergie.

L'Industrie vise également les MPSoCs hétérogènes. Certaines propositions industrielles sont présentées dans [34, 35, 36] et [25]. Dans [34], STMicroelectronics présente un MPSoC flexible appelé StepNP qui contient plusieurs processeurs multithread configurables, PEs configurables et des Entrées / Sorties orientées réseau, tous reliés sur puce. Il répond à l'exigence de flexibilité, un développement rapide et la productivité de l'utilisateur final. Leijten et al. [35] proposent un MPSoC qui est conçu par une méthode connue sous le nom PROPHID. L'architecture PROPHID contient un processeur d'usage général pour le contrôle et pour le traitement du signal de faible et moyenne performance. En plus, des processeurs spécifiques au domaine pour le traitement du signal à haute performance. Rutten et al. [36] proposent un MPSoC appelé Eclipse qui contient des coprocesseurs de domaine spécifiques pour l'exécution des tâches d'une ou plusieurs applications en même temps afin de fournir des performances élevées. Dans [25], les MPSoCs proposés par plusieurs sociétés sont listés. Ils contiennent un nombre varié de PEs implémentés dans des Cœurs en fonction de la nécessité du domaine d'application ciblée. Certains

MPSoCs contemporains montrent que la plupart d'entre eux contiennent plus des PEs spécifiques à l'application que des PEs à usage général et sont introduits par la suite.

— CELL
— Nexperia
— OMAP
— Nomadik
— XPP
— AVISPA CH
— Configurable Cores
— Fine-grain Reconfigurable Hardware
— Exynos Sumsung
— SnapDragon

2.2.3 Interconnexions pour les architectures sur Puce

Une interconnexion sur puce est nécessaire pour connecter les PEs des MPSoCs afin de satisfaire leurs besoins de communication. En dehors des options d'interconnexion disponibles, telles les connexions point à point, bus et réseaux sur puce (NoC), il a été observé que le NoC est le plus efficace et hautement scalable [14, 15, 16]. Le terme NoC est utilisé dans plusieurs contextes variant du bus segmenté en multi-niveaux au réseau sur puce [37]. Plusieurs chercheurs se concentrent sur la conception de NoC efficaces, qui implique plusieurs défis. Marculescu et al. [38] décrivent les problèmes de recherche dans la conception des NoCs et les classes en cinq catégories :

— modélisation de l'application,
— l'optimisation de la communication dans le NoC,
— la sélection de paradigme de communication,
— synthèse de l'infrastructure de communication,
— évaluation et validation.

Certaines solutions efficaces contre chaque problème ont également été suggérées. Elles doivent être étudiées à partir des perspectives de recherche des futurs NoCs.

Des exemples de NoCs de recherche bien connus sont SPIN [39], AEthereal [40], QNoC [41], Xpipes [16], PNoC [42], ProtoNoC [43], Nostrum [44], MANGO [45] et HERMES [46]. Les approches de conception pour les NoCs mentionnées sont décrites dans leurs références respectives. D'autres approches de conception ont été mentionnées dans [47, 48, 49].

Bjerregaard et al. [45] fournissent une excellente étude de recherche des NoCs existants et leurs pratiques. Dernièrement, plusieurs sociétés start-up comme Sonics [50] et Arteris [51] ont commencé à commercialiser les concepts des NoCs avec leurs produits NoCs. Les NoCs nécessitent des mécanismes de routage intelligents pour transférer efficacement des données d'un PE à un autre PE du MPSoC. Certains de ces mécanismes sont présentés dans [52] et [53]. Dans [52], un nouveau schéma de routage intitulé DyAD est proposé, qui prend les avantages à la fois des systèmes de routage déterministe et adaptatif par commutation entre eux en fonction de la congestion du réseau. Dans [53], une stratégie de routage multi-chemin est présentée qui garantit la délivrance de données dans l'ordre par la diffusion optimale du trafic dans le NoC afin de réduire les besoins en bande passante du réseau. L'architecture NoC impose aussi de nouveaux défis de gestion dynamique (run-time). Par exemple, le réacheminement de communication, à savoir : changer le chemin de communication entre un PE source et un PE destination au moment de l'exécution. En outre, les algorithmes de gestion des ressources doivent tenir compte des propriétés de l'interconnexion.

2.2.4 Conception de plateformes Systèmes MultiProcesseurs sur Puce

Comme mentionné précédemment, les MPSoCs peuvent fournir des solutions architecturales des plus efficaces pour soutenir différents domaines d'applications. En conséquence, des outils pour la conception et la simulation de ces systèmes sont nécessaires. La conception des MPSoC implique plusieurs défis [54] en l'occurrence le nombre et le type de processeurs, la taille des mémoires, les infrastructures de communication et différents accélérateurs à prendre en considération dans la conception d'un MPSoC prometteur pour un ensemble d'applications donné. Les universités

et l'industrie ont proposé plusieurs méthodes de conception logicielle ou matérielle, intégrées dans des outils différents.

Techniques de conception basées sur le logiciel

Les approches de conception basées sur les logiciels fournissent des plateformes de simulation pour les MPSoCs, qui sont relativement plus faciles à concevoir que des plateformes matérielles. Toutefois, les plateformes de simulation fournissent des résultats très proches de l'exact et prennent parfois beaucoup de temps dans la simulation. Des travaux existant, dans ce sens, sont présentés dans [55, 56, 57, 58]. Benini et al. [55] proposent une méthode pour la conception de ce genre de plateforme, appelée MP-ARM, qui contient des processeurs ARM sur SystemC [59] et architecture de communication compatible avec le bus AMBA. Paulin et al. [60] présentent une autre technique pour la conception d'une plateforme de simulation MPSoC sur SystemC appelé StepNP. Monchiero et al. [56] présentent un framework de conception appelée GRAPES. Le framework offre flexibilité et modularité en maintenant une vitesse de simulation élevée. Les entités SystemC ou $C++$ sont utilisés pour modéliser les modules IP (intellectual property). Les modules sont capturés en objets $C++$ appelés plugins et sont gérés par le noyau (kernel) GRAPES, qui est le cœur du framework de simulation. Cong et al. [57] présentent une méthodologie pour une génération rapide automatique, vrai cycle, simulateur à base de C pour les coprocesseurs utilisant un outil de synthèse de haut niveau afin de l'intégrer à leur framework de simulation MC-Sim. Le framework est capable de simuler avec précision une gamme de processeurs, mémoires, configurations NoC et coprocesseurs spécifiques à l'application. Atat et al. [58] proposent une approche de conception au niveau système pour le prototypage rapide de MPSoCs à partir de spécifications Matlab / Simulink. Beltrame et al. [61] présentent une méthode à partir de la description d'une application en code séquentiel standard, profilée tout d'abord dans le but de paralléliser, ce qui donne un nombre minimum de processeurs requis pour une contrainte donnée. Ensuite une plateforme de simulation basée StepNP est conçue sur la base des composants parallélisés et le nombre de processeurs. Le débit est

appliqué à un encodeur MPEG4 VGA en temps réel pour l'étude de cas industriel. Interuniversity Microelectronics Centre (IMEC) [3] présente un flot de conception 2.3 utilisant trois outils :

— I) CleanC pour le nettoyage de code source : Le CleanC permet aux concepteurs d'écrire un code séquentiel de haut niveau optimisé pour la parallélisation.

— II) MultiProcessor Assist (MPA) pour la parallélisation du code séquentiel : L'outil MPA permet aux concepteurs d'extraire la parallélisation potentielle présente dans une application en utilisant le parallélisme fonctionnel et des données.

— III) Memory Hierarchy (MH) destiné à effectuer la gestion de la mémoire : L'outil MH ordonnance automatiquement les transferts de données entre la mémoire principale et la mémoire locale par l'analyse du code source d'entrée.

Les MPSoCs conçus basés sur le framework de simulation, sont visés par de nombreux chercheurs car ils sont faciles à concevoir et ont besoin de moins de temps au moment de la conception.

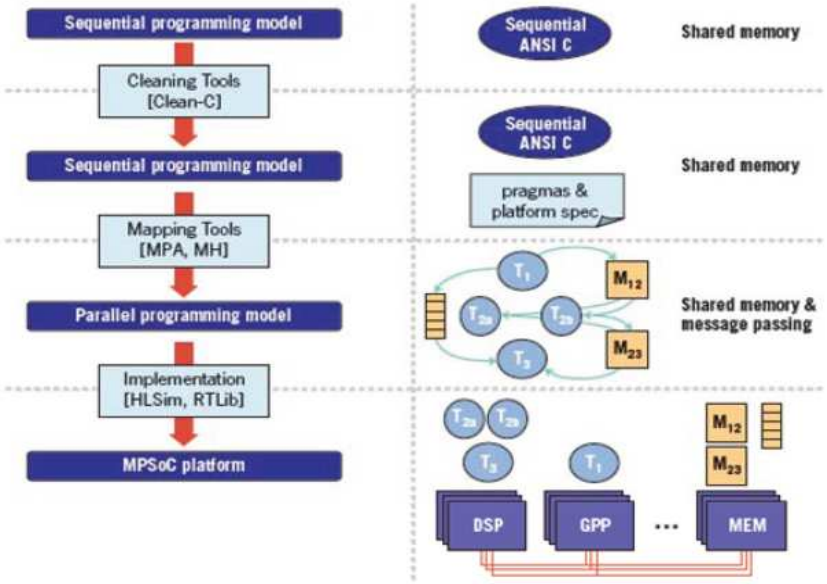


FIGURE 2.3 – Flow de conception MPSoC IMEC [3]

Techniques de conception basées sur le matériel

Les plateformes matérielles sont difficiles à concevoir par rapport aux plateformes de simulation bien qu'elles fournissent une exécution plus rapide que les simulateurs. Quelques méthodes pour concevoir des plateformes matérielles sont données dans [62, 63, 64, 65, 66].

— Nikolov et al. [62] présentent une méthodologie mise en œuvre dans un outil appelé Embedded System level Platform synthesis and Application Mapping (ESPAM) pour la conception automatisée, la programmation et la mise en œuvre de MPSoCs. Cette méthodologie considère une application plateforme au niveau du système, et des spécifications de placement en entrée pour effectuer l'automatisation. La méthode proposée est évaluée par la production et la programmation automatique de plusieurs MPSoCs pour exécuter des applications en temps réel.

— STMicroelectronics [63] proposent une approche modulaire à faible coût pour générer des plateformes matérielles offrant la conception et la vérification de MPSoCs complexes.

— Atienza et al. [64] présentent un framework pour concevoir un MPSoCs sur FPGA, conçu pour fournir une accélération de trois ordres de grandeur par rapport au cycle précis.

— Sun et al. [65] proposent une méthodologie implémentée pour la synthèse MPSoC, utilisant la plateforme Xtensa de Tensilica Inc [67], évaluée en générant automatiquement des MPSoCs personnalisés pour plusieurs benchmarks complexes de logiciels embarqués, les résultats montrent que MPSoCs synthétisés fournissent des résultats plus rapides.

— Kumar et al. [66] présentent une méthodologie mise en œuvre dans un outil appelé Multi-Application Multi-Processor Synthesis (MAMPS) pour générer des MPSoCs de manière systématique et entièrement automatisée pour les applications multimédia. L'outil génère des MPSoCs pour FPGA Xilinx.

Pour des architectures évolutives, les techniques pour générer les MPSoC à base de NoC sur FPGA, sont présentées dans [68, 69, 70]. Jusqu'à présent, les techniques

de conception des MPSoCs ont été décrites, mais certaines techniques de conception de MPSoCs générales devront être ciblées pour le placement d'applications à des stades ultérieurs [55, 58, 63], tandis que d'autres applications spécifiques MPSoCs seront vues dans [61, 3, 62, 65, 66].

Pour gérer le dynamisme, comme l'ajout d'une nouvelle application dans le système au moment de l'exécution, les MPSoCs sont obligatoires. Afin de gérer des applications complexes modernes nécessitant grand nombre de processeurs, les MPSoCs doivent être à base de NoC pour l'efficacité et l'évolutivité. Cependant, il a été observé que la plupart des méthodes de conception produisent des MPSoCs qui ne sont pas à base de NoC et que les MPSoCs sur FPGA pour plateformes matérielles sont une nouveauté à tendance de plus en plus importante [62, 64, 66, 68, 69, 70]. Dorta et al. [71] donnent un bon aperçu des MPSoCs sur FPGA. Cela facilite le prototypage rapide et permet la recherche de nouvelles architectures au contraire de l'approche ASIC. Toutefois, ces performances ont diminué par rapport à leur homologue ASIC mais offrent plusieurs avantages tels que la flexibilité, la reconfiguration, temps de production réduit et à moindre coût. Les FPGA modernes peuvent accueillir 80 à 100 processeurs soft-cores dans une seule puce et le NoC est la meilleure solution pour gérer un si grand nombre de cœurs [72]. Des techniques sont également disponibles pour concevoir mille noyaux systèmes, en utilisant plusieurs FPGA, un tel système a été dénommé Research Accelerator for Multiple Processors (RAMP) [73].

2.3 Placement d'applications sur Multiprocesseurs sur puce

Les applications ont besoin du traitement suivant avant d'entamer leur placement sur les plateformes MPSoC :

— Parallélisations de l'application, ajout de la synchronisation et de la communication inter-tâches. Gestion de la communication de l'environnement externe. Ce travail est décrit par l'état de l'art des outils de parallélisations des applications [74, 3].

— Les tâches peuvent être exécutées en parallèle sur différentes ressources de la

plateforme afin d'accélérer l'exécution de l'application.

— Vérification du code parallélisé (faire en sorte qu'il soit fonctionnellement correct et optimisé pour un ensemble de paramètres de plateforme donnée).

— Dans le cas des plateformes hétérogènes, un procédé de liaison de tâches est requis (task binding). Pour chacune, le processus de liaison définit les types de processeurs sur lesquels elle peut être placée. Il précise également le coût du placement sur les différents types de processeurs.

Le placement des tâches des applications sur une plateforme MPSoC implique leur affectation, leur ordonnancement et le placement des communications sur le NoC en satisfaisons certains critères d'optimisation comme la réduction de la consommation d'énergie, l'amélioration des performances de calcul etc. L'optimisation est nécessaire pour satisfaire les contraintes de performance des applications. Les techniques de placement ont besoin des paramètres suivants :

— Un modèle d'application (par exemple, Task Graph [75], Data Flow Graph [76] etc.)

— Un modèle d'architecture de la plateforme MPSoC (par exemple, la topologie, le nombre d'unités de traitement et leur type, le schéma d'interconnexion, etc.)

— Les contraintes de l'application (par exemple, la performance de calcul et / ou les exigences de puissance etc.).

— Le modèle de performance de communication inter-tâches (temps d'exécution, la consommation d'énergie, etc.).

— Une estimation de cas du pire temps d'exécution des tâches de l'application sur différents PEs.

Le problème de placement est abordé par plusieurs chercheurs communiquant leurs points de vue par le biais de divers forums tels que [11], International Forum on Embedded MPSoC and Multicore [10], diverses conférences et revues internationales. Les techniques de placement peuvent aussi être distinguées selon la spécificité de l'application et le type de la plateforme MPSoC cible. Le placement des tâches des applications sur les ressources de la plateforme MPSoC peut être accompli soit au moment de la conception (statique) ou lors de l'exécution (dynamique). Les

techniques de placement au moment de la conception (design-time) considèrent un ensemble d'applications prédéfinies avec un calcul, comportement de communication et une plateforme statique connus. Par conséquent, ils ne sont pas adaptés pour des charges dynamiques telles que l'ajout d'une nouvelle application dans la plateforme au moment de l'exécution. Les techniques de placement dynamique (run-time) sont nécessaires pour les scénarios où les tâches d'application sont chargées dans la plateforme au moment de l'exécution. Après le placement des tâches, la migration de tâches peut être utilisée pour réviser le placement d'une partie des tâches déjà en cours d'exécution, si l'exigence de l'utilisateur est modifiée ou si une nouvelle application entre en jeu dans le système. Au cours des sous sections suivantes, nous présenterons quelques techniques de placement statique et dynamique rapportées dans la littérature. Mais par la suite, nos travaux se concentreront principalement sur le placement dynamique. Nous explorerons largement les techniques de placement dynamique après avoir fourni une introduction aux techniques de placement statique.

2.3.1 Placement Statique (Design-time Mapping)

Les techniques de placement statique ont une vue globale du système qui aide à prendre de meilleures décisions pour l'utilisation des ressources du système. Ainsi, une meilleure qualité de placement peut être réalisée par rapport aux techniques de placement dynamique qui sont limités à une vue locale. La plupart des travaux liés au placement rapportés dans la littérature traitent le placement statique. Les techniques de placement statique pour les MPSoCs à base de bus et de NoC sont présentées dans [77, 78, 79, 80, 81, 82] respectivement. Les architectures à base de bus ne sont pas scalables et constituent donc une limite pour leurs techniques de placement. Hu et al. [79] en proposent une appelée « Communication Weighted Model (CWM) » et montrent que la consommation d'énergie globale est réduite en diminuant celle de l'énergie des communications. Marcon et al. [80] prolongent le travail dans [79] et proposent une technique appelée « Communication Dependence and Computation Model (CDCM) ». Le CWM considère seulement le volume de la communication,

alors que CDCM considère le volume et la période de la communication. Murali et al. [81] font part d'une méthodologie qui place plusieurs cas d'utilisation sur une architecture NoC où les contraintes de performances pour chaque cas d'utilisation sont respectées. Rhee et al. [82] étudient le problème de « core-switch mapping » (CSM) qui place d'une manière optimale les Cœurs sur l'architecture NoC afin de minimiser la consommation d'énergie ou la congestion NoC. Différentes approches de recherche bien établies sont également utilisées pour élaborer les techniques de placement statique afin de trouver un placement optimal des tâches sur les unités de calculs (PEs) de la plateforme. L'approche génétique est utilisée dans [83, 84], recherche Tabu dans [85] et recuit simulé dans [86, 87]. Lei et al. [83] présentent un algorithme de placement génétique en deux étapes visant à optimiser le temps d'exécution de l'application. Wu et al. [84] présentent un algorithme de placement génétique qui utilise « Dynamic Voltage Frequency Scaling » (DVFS) pour réduire la consommation d'énergie. Manolache et al. [85] étudient le placement de tâches visant à garantir la latence des paquets réseaux afin de garantir le temps de réponse des applications pour le pire de cas. Orsila et al. [87] proposent un algorithme de recuit simulé qui optimise le temps d'exécution et la consommation mémoire, alors que les approches traditionnelles se concentrent uniquement sur le temps d'exécution. D'autres techniques récentes de placement statique sont résumées dans [88, 89, 90, 91].

Benyamina et al. [92] ont traité le placement statique dans les NoCs. La phase de mapping dans les systemes sur puce représente une phase centrale lors de leur mise en œuvre. Ce mapping a une particularité, dans ce contexte, car il ne s'agit pas uniquement d'un placement classique qu'on a l'habitude de rencontrer dans les systèmes parallèles mais il englobe trois phases : Assignation, Affectation et Schedul ing (AAS). Ce type de placement ou mapping dans sa globalité est connu comme étant un placement GILR : Globally Irregular Locally Regular, il consiste à exploiter le parallélisme de tâches et de données dans une architecture multiprocesseurs. Benyamina et al. [92] ont traité le problème dans sa globalité, ils ont utilisé l'algorithme génétique pour le placement de tâches sur des PEs et dijkstra pour les communi-

cations. Un second projet sous la responsabilité de P.Boulet et Benyamina traité le même problème à l'aide de l'algorithme PSO à la place d' AG. La comparaison entre les deux approches a montré que l' AG est mieux que PSO. A. AROUI [93] s'est intéressé à la partie régulière. En effet, dans les applications DSP on a généralement un flot de données infini qui sont traitées par des tâches répétitives. Ces tâches ont la particularité de présenter les mêmes caractéristiques : le nombre de cycles d'exécution et la taille des données traitées. En plus ces tâches s'appliquent sur des données de même dimension et structure, qu'on appelle motifs. Donc il sera bénéfique de placer ces tâches sur la partie homogène (régulière) de l'architecture cible (MPSoC). A.AROUI [93] a proposé une nouvelle technique de placement des tâches répétitives d'une application sur une partie d'une architecture multiprocesseurs sur puce, afin de minimiser le coût de communications et la consommation d'énergie et de respecter les différentes contraintes de conception. Pour cela, sa démarche consistait à proposer et de mettre en œuvre une méthode hybride pour solutionner ce genre de problème, constituée d'une part, d'une méthode exacte : Branch and Bound bi objectifs, permettant de trouver le meilleur placement des taches selon plusieurs objectifs. D'autre part de l'algorithme Dijkstra modifié et amélioré en bi-objectifs pour le mapping des communications.

Toutes les techniques de placement statique trouvent le placement des tâches au moment de la conception. Par conséquent, ces techniques ne conviennent pas pour des charges de travail variables dynamiques dans les systèmes qui exige un placement et un replacement dynamique d'applications (par exemple, les applications réseau et multimédia). Même si les techniques de placement statique sont insuffisantes pour les scénarios à charge de travail dynamique elles peuvent être utiles pour trouver le placement initiales des tâches, ou être optimisées lors de l'exécution.

2.3.2 Placement Dynamique (Run-time Mapping)

Contrairement au placement statique, le placement dynamique doit tenir compte du temps pris pour placer chaque tâche car il contribu à la durée totale de l'exécution de l'application. Par ailleurs, les tâches sont placées une par une, à la différence du

cas statique où elles sont toutes placées à la fois en examinant le système d'une façon globale. Par conséquent, les algorithmes gloutons sont utilisés pour le placement efficace dans le but d'optimiser les mesures de performance telle que la consommation d'énergie, la latence de communication, le temps d'exécution, etc. En plus de l'aptitude des techniques de placement dynamique par rapport aux techniques de placement statique, dans le cas de scénarios de charge de travail dynamique, elles offrent également un certain nombre d'autres avantages :

— Adaptabilité aux ressources disponibles : les ressources disponibles varient au fil du temps, les applications de scénario à charge de travail dynamique entrent au moment de l'exécution.

— Possibilité d'activer des mises à niveau imprévisibles : il est possible de mettre à niveau le système pour de nouvelles applications ou des normes changeantes pas connus au moment de la conception, même après la livraison du système à l'utilisateur final.

— Le vieillissement peut conduire à des unités de traitement défectueuses imprévisibles au moment de la conception.

— Capacité à éviter les parties défectueuses d'un SoC : Si une ou plusieurs unités de traitement ne fonctionnent pas correctement après la production d'un SoC, alors les unités défectueuses peuvent être désactivées et les charges qui leurs ont été affectées seront réaffectées à d'autres processeurs à l'aide du placement dynamique [94].

Les techniques de placement dynamique répartit les tâches et leurs communications aux unités de traitement (PEs) et les liens d'interconnexion, respectivement, pour toutes les applications à placer. Lorsque les applications mappées démarrent leur exécution, le placement d'une ou plusieurs d'entre elles en cours d'exécution, doit être réexaminé en cas d'événements suivants :

— Lorsqu'une nouvelle application ayant besoin de ressources déjà en exécution, est entrée dans le système.

— Lorsque les paramètres d'une application en exécution sont modifiés.

— Quand une application en cours d'exécution doit être tuée pour libérer ses ressources occupées.
— Lorsque les exigences de performance d'une application en cours d'exécution sont modifiées. Elle pourrait avoir besoin de ressources supplémentaires pour exécuter des fonctionnalités supplémentaires.
— Lorsque le placement actuel n'est pas suffisamment optimal, il exige un replacement.

Les problèmes mentionnés ci-dessus ne peuvent être traités que par des techniques de placement dynamique c'est à dire au moment de l'exécution. Au moment de ce dernier, le placement de nouvelles applications qui doivent être soutenues sur une plateforme, peuvent être manipulées soit en effectuant tout le traitement en même temps c'est à dire par un traitement à la volée (on-the-fly processing) soit en utilisant les résultats analysés précédemment. Pour le traitement à la volée, des heuristiques efficaces sont nécessaire pour assigner les nouvelles tâches arrivant sur les ressources de la plateforme. Ces heuristiques ne peuvent garantir que les échéances temporelles strictes respectées en raison des ressources de traitement limitées lors de l'exécution. Toutefois, elles sont applicables à toute plateforme comme elles n'utilisent pas de résultats d'analyse spécifique de plateforme calculés à l'avance. Pour le placement utilisant des résultats précédemment analysés lors de la conception, des heuristiques de placement dynamique sont nécessaires. On peut dire aussi que le placement pré-analysé permet de faire un meilleur placement d'applications que celui du placement à la volée. Mais les résultats d'analyse ne seront pas applicables à toutes les plateformes. Ensuite, nous discutons entre des heuristiques de placement à la volée et les stratégies de placement qui utilisent les résultats d'analyse au moment de la conception, rapportées dans la littérature.

Placement à la volée (Mapping On-the-fly)

Les techniques de placement ciblent les MPSoCs homogènes ou hétérogènes en fonction de l'exigence d'application.

Techniques ciblant les MPSoCs homogènes

Certaines techniques de placement dynamique visant les MPSoCs homogènes sont présentées dans [95, 96, 97, 98, 99, 100, 101, 102]. Chou et al. [95] proposent une technique qui incorpore les informations de comportement de l'utilisateur dans le processus d'allocation de ressources qui permet au système de mieux répondre aux changements en temps réel et de s'adapter dynamiquement aux besoins des utilisateurs. Cette considération économise 60% de l'énergie de communication par rapport à une technique d'allocation des tâches arbitraires.

Peter et al. [96] présentent un algorithme distribué sur les processeurs pouvant donc être appliquées aux systèmes de tailles aléatoires. En outre, les tâches ajoutées au moment de l'exécution peuvent être manipulées sans aucune difficulté. Ce qui permet une optimisation en ligne. La migration des tâches s'effectue sur la base des informations locales sur la charge de travail du processeur, la taille de la tâche, les exigences de communication, et le lien. Les résultats de placement sur plusieurs exemples d'ensemble de tâches montrent que la qualité obtenue par l'algorithme présenté est à moins de 25% de celle de l'algorithme exact, pour un réseau de processeur 3X3.

Briao et al. [97] présentent des stratégies basées sur des algorithmes bin-packing pour exécuter des applications à temps réel souple. Ils combinent différents types d'algorithmes pour obtenir des résultats d'allocation meilleurs. Afin d'économiser l'énergie, le système désactive les processeurs inactifs.

Chou et al. [98] proposent une technique qui considère différentes unités de calcul (PEs) qui opèrent sur des niveaux de voltage différents pour un placement prenant en compte la consommation d'énergie.

Ngouanga et al. [99] décrivent une technique basée sur les forces d'attraction entre les tâches communicantes. La technique tente de placer les tâches proches sur les unités de calculs (PEs) voisines de la plateforme MPSoC afin de réduire le coût des communications.

Mehran et al. [100], quant à eux, proposent une heuristique nommé « Dynamic Spiral Mapping » (DSM) pour une topologie 2D maillée où le placement est recherché dans un chemin en spirale, essayant de placer les tâches communicantes proches les

unes des autres. Avant de commencer la recherche en spirale, le degré de chaque tâche dans une application est trouvé et celle à degré maximal est placée au centre de la maille pour réaliser un placement des plus communicantes le plus proche possible l'une de l'autre.

Sassatelli et al. [101] proposent deux techniques différentes appelées communications proactives et réactives, et procèdent à des évaluations du cycle précis de ces techniques. Les auteurs affirment que les systèmes homogènes à cycle précis, peuvent devenir une alternative viable dans un avenir proche apportant des avantages tels que haute performance, faible consommation d'énergie et équilibrage de charge dynamique.

Moreira et al. [102] présentent une technique affectant d'abord les tâches à des Cœurs virtuels (VCs), tout en essayant d'en minimiser le nombre total et la bande passante utilisée. Par la suite, les VCs sont mappés à des Cœurs réels.

Techniques ciblant les MPSoCs hétérogènes

Actuellement les MPSoCs sont très hétérogènes pour mieux répondre aux exigences des applications. Dans ce cas , le processus de liaison de tâche (Task binding process) est réalisé avant de commencer le placement. Pour chaque tâche, ce processus définit les types de PE sur lequel la tâche peut être mappée avec le coût de son placement. La figure 2.4 présente le processus de liaison (profilage au moment de la conception) pour un exemple d'application, où les tâches sont analysées sur différents types d'unités de calcul telles que GPP, DSP, matériel reconfigurable à gros grain etc. Le profilage fournit la performance, la puissance et l'utilisation de ressources pour chaque tâche sur différents types de PEs.

Smit et al. [4] présentent un algorithme plaçant en premier lieu, les tâches nécessitant des ressources critiques, puis toutes les autres en prenant en compte la disponibilité des ressources de la plateforme. Plusieurs techniques pour le placement des applications de streaming sur des architectures multi-Cœurs sont présentées dans [29].

Dans le projet des puces intelligentes pour un environnement intelligent (4S) [30], un outil de placement spatial nommé SMIT est développé. Il est déclenché par un

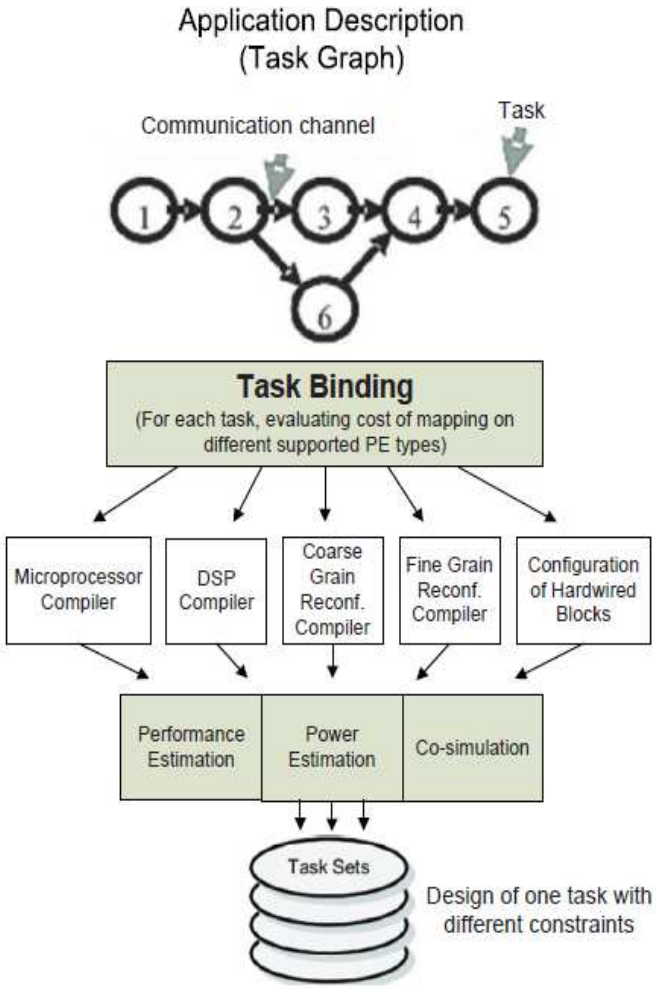


FIGURE 2.4 – Processus de liaison de tâche (Task binding process)

RTOS lorsqu'une application doit être mappée sur le MPSoC. L'outil prend la description de l'architecture du système, la description fonctionnelle de l'application, la réalisation du processus, l'état du système actuel et les contraintes de performance en entrée et en sortie. Un placement fournit le placement des tâches et des communications dans l'architecture et est utilisé pour configurer le système, comme le montre la figure 2.5. La réalisation est obtenue par l'application du processus de liaison de tâche (Task binding process) précédemment décrite. L'outil SMIT effectue l'optimisation sur toutes les unités de traitements (PEs) du système et le réseau de communication.

Holzenspies et al. [103] présentent une technique de placement spatial dynamique composée de quatre étapes pour les applications de streaming sur un MPSoC hé-

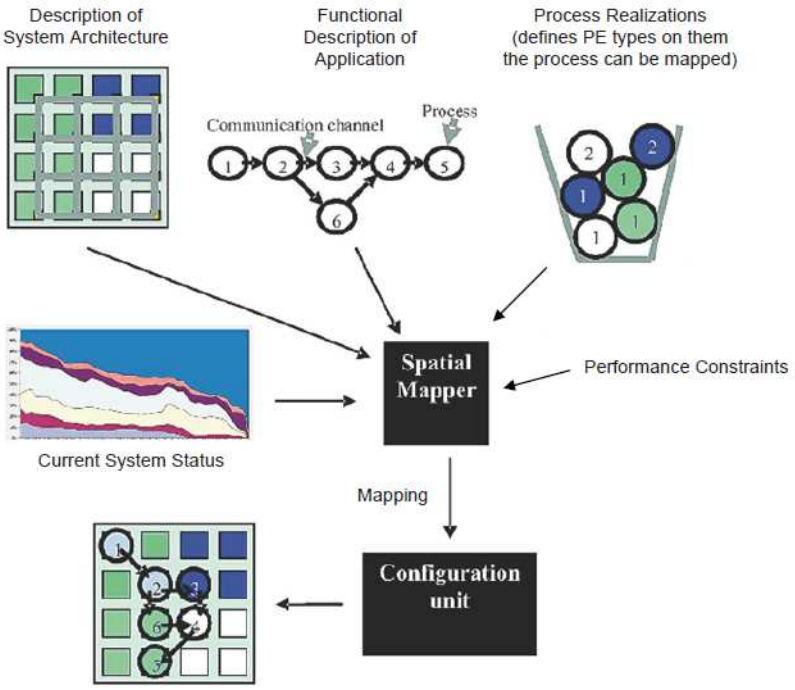


FIGURE 2.5 – Placement dynamique suivie par l'outil SMIT Mapper [4]

térogène. L'algorithme est mis en œuvre sur un ARM926 fonctionnant à 100 MHz (cela prend moins de 4 ms pour exécuter le HIPERLAN / 2 par exemple).

Braak et al. [104] proposent une autre technique de placement spatial dynamique qui couvre à la fois le graphe de tâches et la plateforme MPSoC pour trouver leur placement optimal.

Nollet et al. [28] décrivent une heuristique d'assignation de tâches pour un placement efficace dans un MPSoC contenant de tuiles FPGA, grâce auxquelles l'heuristique est capable de gérer une hiérarchie de configuration permettant d'améliorer la qualité et le taux de succès de l'assignation des tâches.

Faruque et al. [105] présentent une technique de placement dynamique d'applications basée sur des agents distribués ciblant les grands MPSoCs comme les systèmes 32X32 et 32X64. Pour les grands MPSoCs, la technique de placement distribuée est meilleure que les techniques de placement dynamique de l'état de l'art qui utilisent l'approche du manager centralisé (Centralized Manager (CM)). L'approche CM pour les grandes MPSoCs peut faire face aux problèmes suivants :

— Point de défaillance unique.

— Grand volume du trafic de surveillance par le CM

— Le coût élevé pour calculer le placement à l'intérieur du CM.

— Un goulot d'étranglement autour du CM comme chaque Cœur envoie son état au CM après chaque instance de placement. Ainsi, le CM devient un point chaud.

La technique distribuée réduit le trafic de surveillance et de l'effort de calcul.

Schranzhofer et al. [106] proposent une heuristique en plusieurs étapes en temps polynomial constituée de solutions initiales suivies par des algorithmes de re-placement de tâches en prenant en compte les contraintes d'énergie. Premièrement, les solutions initiales pour le scénario d'optimisation d'énergie sont trouvées, puis le re-placement de tâches est réalisé afin d'améliorer les solutions.

Lei et al. [107] présentent un algorithme génétique en deux étapes qui trouve le placement des tâches sur les cœurs disponibles visant à minimiser le temps d'exécution global.

Carvalho et al. [108] présentent des heuristiques où les tâches sont placées à la volée (on-the-fly) en fonction des demandes de communication et de la charge dans les liens du NoC. Au moment de l'exécution, certaines techniques de placement utilisent la migration des tâches d'un PE à un autre lorsque le goulot d'étranglement des performances est détecté ou lorsque la charge de travail doit être répartie de façon homogène dans l'ensemble du système [96, 97]. La migration de tâches peut également être utilisée au cas où le besoin de l'utilisateur est modifié ou une nouvelle application entre dans le système afin de réviser le placement d'une partie des tâches en exécution. La migration des tâches doit être exécutée sans complètement arrêter et redémarrer les applications déjà en exécution.

Placement Basés sur les résultats d'analyse au moment de la conception

Les stratégies de placement basées sur les résultats d'analyse lors de la conception, effectuent l'analyse du calcul intensif au moment du design et utilisent les résultats analysés au moment de l'exécution.

Cela aide pour une légèreté de gestion de plateforme qui mappe dynamiquement

et efficacement les applications basées sur l'état des ressources de la plateforme. L'analyse au moment du design est effectuée en prenant la description de l'application, les spécifications de la plateforme et les objectifs de conception pris en compte pour explorer les points utilisés lors de l'exécution qui contiennent des tâches à des regroupements PEs, à savoir les placements, représentant des compromis entre les différentes mesures de performances. Explorer toutes les combinaisons possibles des tâches sur les PEs n'est exhaustivement pas possible dans un temps limité. Par conséquent, des stratégies d'analyse plus rapides à objectifs de conception sont nécessaires pour explorer des placements pour une application (section 2.3.1) qui peuvent être utilisés pour réaliser une telle analyse. Elles effectuent l'exploration en vue de certains paramètres d'optimisation tels que la performance de calcul et de l'énergie.

L'exploration du placement unique ne peut pas gérer le dynamisme de disponibilité de ressources et les exigences de performances au moment de l'exécution.

Les stratégies d'analyse au moment de la conception génèrant plusieurs placements pour l'application ont récemment été rapportés dans pas mal de travaux. Ces placements peuvent être utilisés pour gérer le dynamisme de la disponibilité des ressources et l'exigence de performance au moment de l'exécution.

Dans [109], l'exploration est effectuée en vue d'optimiser la consommation d'énergie et la performance afin d'identifier le meilleur compromis du rapport performance / puissance consommée.

Stuijk et al. [110] optimisent pour l'utilisation des ressources, Beltrame et al. [111] pour l'énergie et le délai. Ils essaient de minimiser le nombre de simulations nécessaires pour identifier les placements fournissant un compromis d'énergie / délai. Palermo [112] offre un cadre d'exploration multi-objectifs qui fournit aussi des ponts de compromis d'énergie / délai. Jia et al. [113] présentent une infrastructure appelée NASA (Non Ad-hoc Search Algorithm), qui utilise différentes combinaisons de stratégies de recherche pour explorer le placement. Il y a eu beaucoup de recherches dans d'exploration des systèmes lors de la conception (DSE) des applications multiples. Certains chercheurs se concentrent sur une approche fondée sur plusieurs scénarios

de placement d'applications examinés au moment du design pour gérer le dynamisme du nombre de demandes actives au moment de l'exécution.

2.4 Analyse des heuristiques de placement et Conclusion

Dans la section précédente on a fait une description très concise des techniques de placement sur MPSoC connues jusqu'à maintenant. Les techniques existantes, pertinentes pour les placements des applications sur les plateformes MPSoC ont été décrites dans cette section jusqu'à l'heure actuelle. L'objectif de cette sous-section est d'examiner de manière critique les techniques existantes et d'identifier des pistes de recherche. Les critères d'évaluation sont l'aptitude à la charge de travail dynamique dans les systèmes, le gain dans différentes mesures de performances et de la complexité de calcul pour l'évolutivité. Les techniques de placement au moment de la conception décrites dans la section 2.3.1 examinent un ensemble fixe d'applications et une plateforme statique en entrée, et trouvent donc un placement en ayant une vision globale du système. Par conséquent, ils peuvent fournir une meilleure qualité de placement par rapport aux techniques dynamiques ayant normalement une vue locale. Cependant, les techniques de placement statique ne sont pas adaptées pour les scénarios de charge de travail dynamiques tels que l'ajout d'une nouvelle application dans le système au moment de l'exécution, mais elles sont nécessaires pour gérer de tels scénarios. Les techniques de placement dynamique décrites dans la section 2.3.2, réparties en deux directions, chargent des tâches de l'application dans le système au moment de l'exécution lorsque l'application doit être prise en charge. Certains attaquent le problème de placement en définissant des heuristiques efficaces décrites à la section 2.3.2, où de nouvelles tâches sont affectées sur les ressources système lors de l'exécution et tout le traitement se fait soit au même moment, soit à la volée. Ces heuristiques à la volée ne peuvent pas garantir l'ordonnancement, c'est à dire pour les délais du temps strict (strict timing deadlines) en raison de la puissance de traitement limitée au moment de l'exécution. D'autres analysent des applications au moment de la conception (off-line) en définissant des stratégies d'analyse efficaces, décrites à la section 2.3.2, où les allocations et l'ordonnancement

(c'est à dire les placements) sont calculés et vérifiés, puis stockés sur le système. Afin de prendre en charge une application au moment de l'exécution, le meilleur placement est choisi parmi ceux stockés sur la base des ressources système disponibles, et de la performance requise par la suite, utilisés pour configurer le système. Cela est plus efficace pour une légère gestion de système dynamique et placement d'applications que les heuristiques à la volée. Toutefois, la flexibilité dans ces approches est limitée car toutes les applications potentielles doivent être connues intégralement au moment de la conception et les résultats d'analyse applicables uniquement à la plateforme analysée. Par conséquent, l'analyse au moment de la conception doit être répétée lors de changements de l'ensemble d'application ou plateforme. En outre, les résultats d'analyses stockées introduisent une charge supplémentaire. En revanche, les heuristiques à la volée sont applicables à n'importe quel ensemble d'applications et à n'importe quelle plateforme. Il a été observé que les techniques de placement ciblent les MPSoCs hétérogènes pour mieux répondre aux exigences des applications par rapport à leurs homologues homogènes. En outre, les MPSoCs sont basés sur les infrastructures de communication NoC pour des architectures efficaces et évolutives. Au début de cette recherche, nous avons été témoins que la plupart des heuristiques à la volée existantes pour les MPSoCs hétérogènes basés NoC, ne supportent pas d'applications avec un nombre très grand de tâches, et avec l'hypothèse que l'exécution des tâches et des communications sont de même niveau. Les travaux existants ne mettent pas l'accent sur la minimisation du temps de recherche (temps de placement) c.à.d le temps nécessaire pour trouvée une ressources qui peut supporter une tâche prête à l'exécution. Ainsi la plupart des travaux utilisent pour le placement des communications des stratégies de routage statique à savoir le routage XY, vu sa simplicité d'implémentation. Pour une charge de travaille dynamique, l'utilisation de techniques de placement des communications dynamiques ne peut qu'être bénéfique pour de tels systèmes. Les performances fournies par les techniques existantes de placement, demeurent une préoccupation principale en raison des goulots d'étranglement de communication. Par conséquent, des techniques de placements meilleurs devraient être étudiées pour offrir de meilleures performances. Le Chapitre 4 traite

nos techniques de placement dynamique proposées, basées sur une stratégie de packing en spirale et un routage basé sur l' algorithme de plus court chemin de Dijkstra modifié, ciblant les MPSoCs contenant des PEs qui supportent une seul tâche, nous montrons que la performance (temps d'exécution et consommation d'énergie) est améliorée. Le Chapitre 5 traite nos techniques de placement dynamique proposées basées sur un placement mono-tâche en utilisant une stratégie de packing Manhattan afin de minimiser le temps de placement en plus de la prise en charge des communications par la proposition d'une heuristique de placement dynamique des communications. Le chapitre 3 présente une vue d' ensemble sur les simulateurs puis un état de comparaison. Notre contribution dans ce chapitre, sera de détailler notre simulateur de placement dynamique d'applications sur une architecture MPSoC basée NoC : conception, modélisation, fonctionnement etc. L'extension du simulateur afin de supporter une simulation de plateforme multi-tâches sera considérée dans notre futur travail. En fin le Chapitre 6 contient une conclusion de la thèse et une identification de quelques perspectives.

Chapitre 3

Simulateur

Sommaire

3.1 Introduction

Le premier objectif de notre recherche est, de proposer d'autres techniques de placement dynamique permettant de placer les tâches d'applications et leurs communications sur une plateforme MPSoC basée NoC tout en réduisant la consommation d'énergie et le temps d'exécution. Cependant, pour tester ces différentes techniques et les comparer, un outil de tests et de simulation s'avère nécessaire. En l'absence d'une plateforme libre permettant de réaliser ces fonctionnalités, la conception et la réalisation d'un outil qui permet de tester différentes techniques de placement dynamique et simuler le trafic sur un NoC ont été réalisés. Dans ce chapitre, notre simulateur développé en collaboration entre les deux équipes DART et LAPECI

sera détaillé. Ce simulateur permet de simuler à un haut niveau d'abstraction le comportement d'un système MPSoC qui utilse un réseau sur puce pour la communication. Ce niveau, permet de simuler le comportement de placement et d'exécution des tâches d'applications. Le simulateur repose sur des modèles d'architecture et d'application. L'application est un ensembles de tâches qui communiquent entre elles, où toutes les données liées aux tâches et aux communications sont connues. A l'architecture on associé un modèle qui définit les différentes caractéristiques et types d'éléments de traitement. l'avantage du simulateur vis a vis de notre axe de recherche, et qu'il nous permettra d'intégrer facilement nos propositions algorithmiques et d'avoir les mesures de performances souhaitées, qui sont dans notre cas le temps d'exécution et la consommation d'énergie.

3.2 Outils utilisés

Dans ce qui suit, les outils utilisés pour la réalisation de la plateforme de simulation sont présentés.

3.2.1 Langage de développement

Le langage Java a été utilisé comme outil de développement. Ce langage présente les avantages suivants :

— simple à maitriser.

— il est orienté objet, ce qui a permis d'implémenter la plateforme de manière modulaire et de faciliter son extension.

— il est indépendant de la plateforme d'exécution, ce qui rend l'application compatible avec d'autres systèmes d'exploitation.

3.2.2 Environnement de développement

La plateforme de simulation a été développée sous le système d'exploitation Windows avec l'environnement Eclipse.

Eclipse est un environnement de basé sur la démarche IDM (MDE). Il est orientée objet, assurant une grande interopérabilité des outils les constituant. Il permet

le développement d'applications en vue de leur déploiement sous Windows ou sous Linux. Il trouve son origine au sein de la société IBM qui a décidé en 2001 de mettre à disposition de la communauté Open Source l'ébauche d'une plateforme de développement ouverte, entièrement écrite en Java et capable d'intégrer des extensions adaptées à diverses activités (débogage, modélisation, interfaces graphiques, etc.).

3.3 Quelques simulateurs et Motivation

Les réseaux sur puce (NoCs) ont attiré une grande attention au cours des dernières années comme étant un nouveau paradigme d'interconnexion des différents composants d'une puce. En tant que tel, il y a eu un besoin accru de définir et de développer des logiciels de simulation de l'exécution d'applications sur des réseaux de processeurs sur puce. Quelques simulateurs sont présentés dans ce qui suit :

- **GPNoCSim :** Pour General Purpose Simulator for Network-on-Chip [114], c'est un simulateur de NoC open source, développé entièrement en java. Il est flexible car il permet aux designers de rajouter de nouvelles architectures ainsi que la configuration du traffic. Le but final de ce simulateur est de comparer entre les différente topologies du NoC en mesurant la latence, l'utilisation des liens, etc.
- **BookSim :** Est un simulateur d'interconnexion des réseaux [115]. La version actuelle majeure, BookSim 2.0, prend en charge un large éventail de topologies comme le mesh2D, tore et fat tree. Il fournit divers algorithmes de routage et inclut de nombreuses options de personnalisation de micro-architecture routeur du réseau. Il a été développé par des chercheurs de l'université de Stanford aux USA.
- **Noxim :** Ce simulateur a été proposé par l'équipe d'architecture des ordinateurs de l'Université de Catane [116]. Il est développé en Langage SystemC. Il permet à l'utilisateur de définir un NoC ayant une topologie en 2d mesh avec des paramètres différents : taille du réseau, taille du buffer, la taille des paquets, algorithme de routage et taux d'injection de paquets. Noxim per-

met l'évaluation des NoC en termes de débit, de latence et la consommation d'énergie.

- **NS-2 :** A été développé pour le prototypage et la simulation de réseaux informatiques ordinaires. Cependant, puisque les NoCs ont de nombreuses similarités avec les réseaux classiques, NS-2 [117] a été largement utilisé par de nombreux chercheurs pour simuler un NoC. De nombreuses études ont utilisé NS-2 comme un outil de simulation ce qui en fait une référence fiable, surtout lorsque l'on compare les performances de deux architectures différentes. Enfin, NS-2 est open source, axé sur simulateur à événements discrets et développé en C++.
- **Darsim :** Est un simulateur de NoC qui a été développé à Massachusetts Institute of Technology (MIT) [118]. Ce framework permet la simulation d'architectures mailles de 2 et 3 dimensions. Il offre une multitude de configurations de simulation avec des paramètres différents, tels que la taille du réseau, la taille des VC (Virtual channel), le modèle de mémoire, etc.

La plupart des simulateurs existants focalisent sur le trafic réseau sur différentes topologies en ignorant les éléments d'exécution. C'est ce qui nous a motivé à concevoir un simulateur à haut niveau d'abstraction. Notre objectif dans cette thèse est de développer d'autres techniques de placement dynamique sur des réseaux de processeurs sur puce. La simulation et l'évaluation des algorithmes de placement dynamique proposés, nécessitent un outil de simulation qui permet d'accélérer les développements en Co-design à un niveau d'abstraction plus haut. Ce besoin n'est offert par aucun des simulateurs cités précédemment. La plus part des travaux qui traitent la problématique du Co-design utilisent des outils de développement matériel ou logiciel. Pour le logiciel, ils utilisent des outils à bas niveau. Ce qui provoque un temps de conception et un cycle de développement conséquent, vu la complexité de cette phase. Toutes ces limites, nous ont poussé à développer un simulateur de haut niveau d'abstraction. Ce choix se justifie par la simplicité qui pourrait être obtenu pour la conception et le cycle de développement plus au moins, avant de passer au

niveau le plus bas. Nous pensons à ce que notre simulateur permet d'accélérer les pré-résultats des flots de conception. Le simulateur a été développé en collaboration entre les équipes LAPECI et DaRT de Lille1. Il permet le placement des tâches et des communications. Il permet aussi de simuler différents types d'éléments de traitements (GPP, FPGA, DSP, ASIC, etc.) et les réseaux de communication (routeurs, liens, caches, mémoires, etc.). La plus plus petite unité échangée dans le réseau est un paquet. Donc, le simulateur permet de simuler parfaitement le déroulement des propositions algorithmiques de placement des tâches et des communications. La plateforme de simulation supporte le mono et le multi-tâches. Pour le mono-tâche, chaque ressource ne permet l'exécution que d'une seule tâche. Par contre le multi-tâches permet pour chaque ressource d'exécuter plusieurs tâches. A partir de cette plateforme de simulation, deux autres thèses sont en cours pour étendre l'outil. La première thèse, va traiter les communications à grain fin (phit, flit). La migration de tâches pour un équilibrage de charge meilleurs. La deuxièmes thèse va prendre en considération les tâches temps réel. Pour cela, le modèle d'application va être rendu plus général. Une conception pour la simulations d'applications temps réel est prévue. Une étape d'observation pour prendre les bonnes décisions va être ajoutée. Nous voulons arriver à un standard de simulation qui pourrait être un support de validations des futures travaux de nos équipes.
Le tableau 3.1 résume les différentes caractéristiques des simulateurs vus précédemment.

3.4 Conception détaillée

Afin de comprendre les fonctionnalités fournies par la plateforme de simulation, une modélisation UML (Unified Modeling Language) est présentée. Celle-ci comprend une identification des acteurs des différents cas d'utilisations.

3.4.1 Identification des acteurs et des cas d'utilisations

Le simulateur vise une seule classe d'acteurs, à savoir des utilisateurs qui peuvent être des concepteurs.

	Année	Equipe	Topologie	Outil de développement
GPNoCSim	2006	Université de Bengladesh	Mesh2D, Tore, Fat tree	java
BOOKSIM	2010	Université de Stanford	Mesh2D, Tore, Fat tree	C++
Noxim	2010	Université de Catagne	Mesh2D	C++
NS-2	1995	Lawrence Berkley National Laboratory	Mesh2d, Tore, Fat tree	C++
Darsim	2009	MIT	Mesh 2D, Mesh 3D	C++

TABLE 3.1 – Synthèse des simulateurs

L'utilisateur, un demandeur de service, est une personne qui utilise la plateforme pour réaliser une simulation de placement de différentes applications définit selon un modèle bien précis. Ces placements se font lors de l'exécution, c à d pour chaque application les tâches se génèrent au fur et à mesure dans le temps et sont placées sur la plateforme simulée dynamiquement. Pour chaque simulation des mesures de performances sont produites selon un modèles mathématique définit dans la conception du simulateur. Il est possible de simuler le trafic du NoC et ainsi analyser les performances.

le diagramme de cas d'utlisation (3.1) permet de visualiser de façon globale les services offerts par le simulateur. **DesignNOC** est le seul cas d'utilisation, il offre la possibilité de tester les techniques de placement dynamique, de routage, de simuler afin d'obtenir les performances du NoC.

Dans ce qui suit, une vue statique des éléments qui composent le simulateur et un diagramme de classes global qui explique les différentes interactions entre ses composants sont présentés.

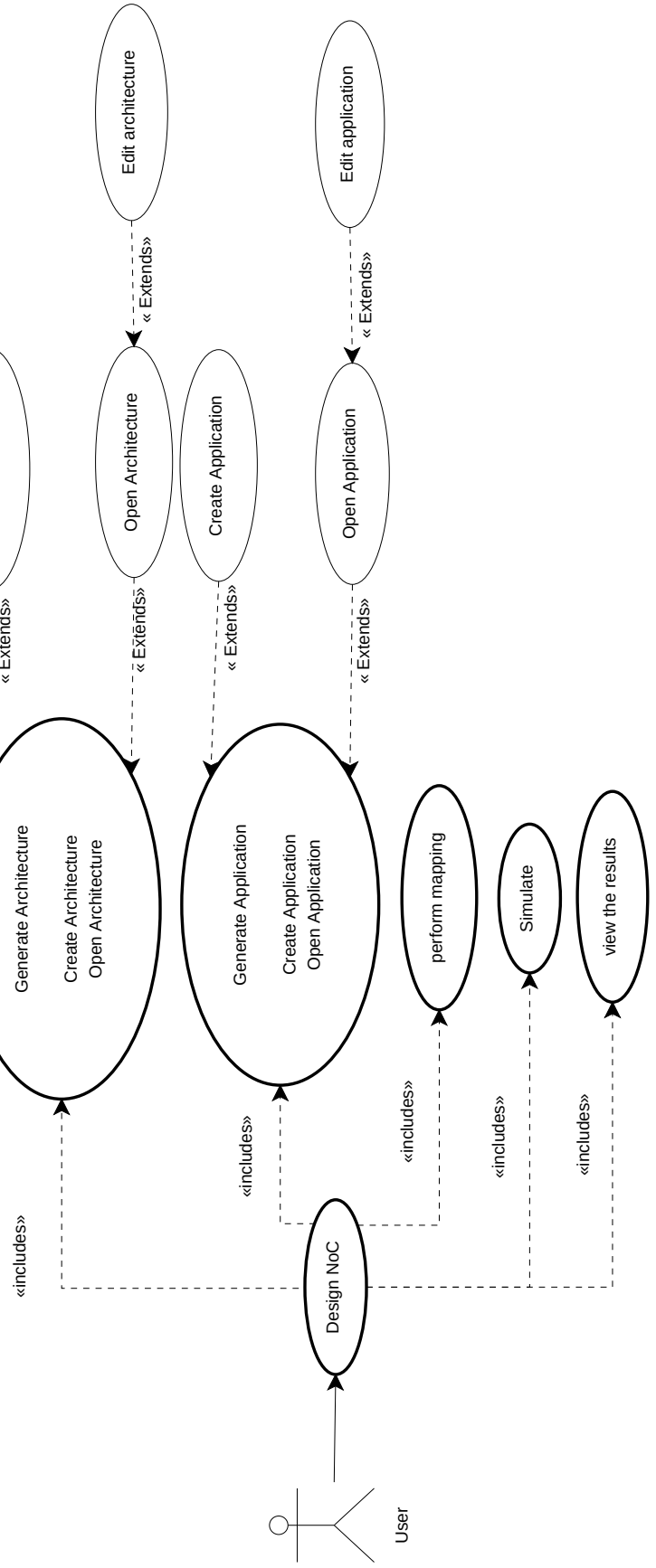


FIGURE 3.1 – Diagramme de cas d'utilisation

3.4.2 Packages de la plateforme

La plateforme a été réalisée autour de cinq grandes fonctionnalités, à savoir :

- la création d'un NoC.
- la création d'applications.
- le placement des applications sur le NoC.
- le routage des communications.
- la simulation du NoC résultant et affichage des performances.

Les différents packages assurant ces fonctionnalités sont :

- **Package Architecture** Ce package permet de générer l'architecture d'un réseau sur puce avec une topologie en mesh 2d. Il est composé de la classe CreateNOC. Cette dernière crée et décrit l'ensemble des tuiles et liens physiques.
- **Package Application** Le package Application charge les différentes applications à partir d'un fichier XML qui seront plus tard placées sur l'architecture. Une application est décrite par un graphe composé de tâches et de liens de communications entre les différentes tâches.
- **Package Dynamic Mapping** Il contient différentes heuristiques de placement dynamique permettant le placement des applications sur une architecture hétérogène.
- **Package Routing** Le package Routage permet le routage des communications entre les tâches. Ce package contient les algorithmes de routage, à savoir : les méthodes statique tel que le XY et dynamiques tel que le MORA.
- **Package Simulation** Ce package se charge de la simulation du placement des différentes tâches et du routage de leurs communications ainsi que toutes les applications sur l'architecture crées. La classe simulateur est le noyau du simulateur, elle permet de gérer les différentes actions menées par une simulation.
- **Package Results** Ce package a pour fonction l'affichage des mesures de performance souhaitées par la simulation.

Diagramme de classes

Le diagramme ci-dessous 3.2 présente les différentes classes du simulateur et les interactions entre elles.

3.4.3 Création de l'architecture

Le simulateur proposé permet de concevoir une architecture NoC hétérogène en Mesh-2D. Cette architecture est composée de plusieurs types de processeurs, à savoir des GP, ASIC, FPGA, DSP. Ces derniers peuvent exécuter du mono ou multi-tâches, on parle alors d'une plateforme Mono ou Multi-tâches. La figure 3.3 montre un exemple d'une architecture qui peut être générée par le simulateur. L'architecture crée est subdivisée en clusters.

Manager :

Un des processeurs (GP) du NoC est utilisé comme un processeur manager. Ce dernier permet le contrôle et la gestion centralisée du NoC. Les principales fonctionnalités assurées par le manager sont :

— **le placement des tâches :** recherche le PE dans l'architecture suivant une heuristique de placement afin de lui allouer une tâche,

— **l'ordonancement des tâches :** définit l'ordre d'exécution des tâches,

— **le routage des communications :** détermine le chemin à emprunter entre deux tâches communicantes (maître/esclave) qui ont été déjà placées auparavant,

— **la mise à jour de la plateforme :** mise à jour en temps réel des tuiles et des liens de communications.

Diagramme de classes

Le diagramme suivant 3.4 illustre les différentes classes et caractéristiques qui composent le package Architecture.

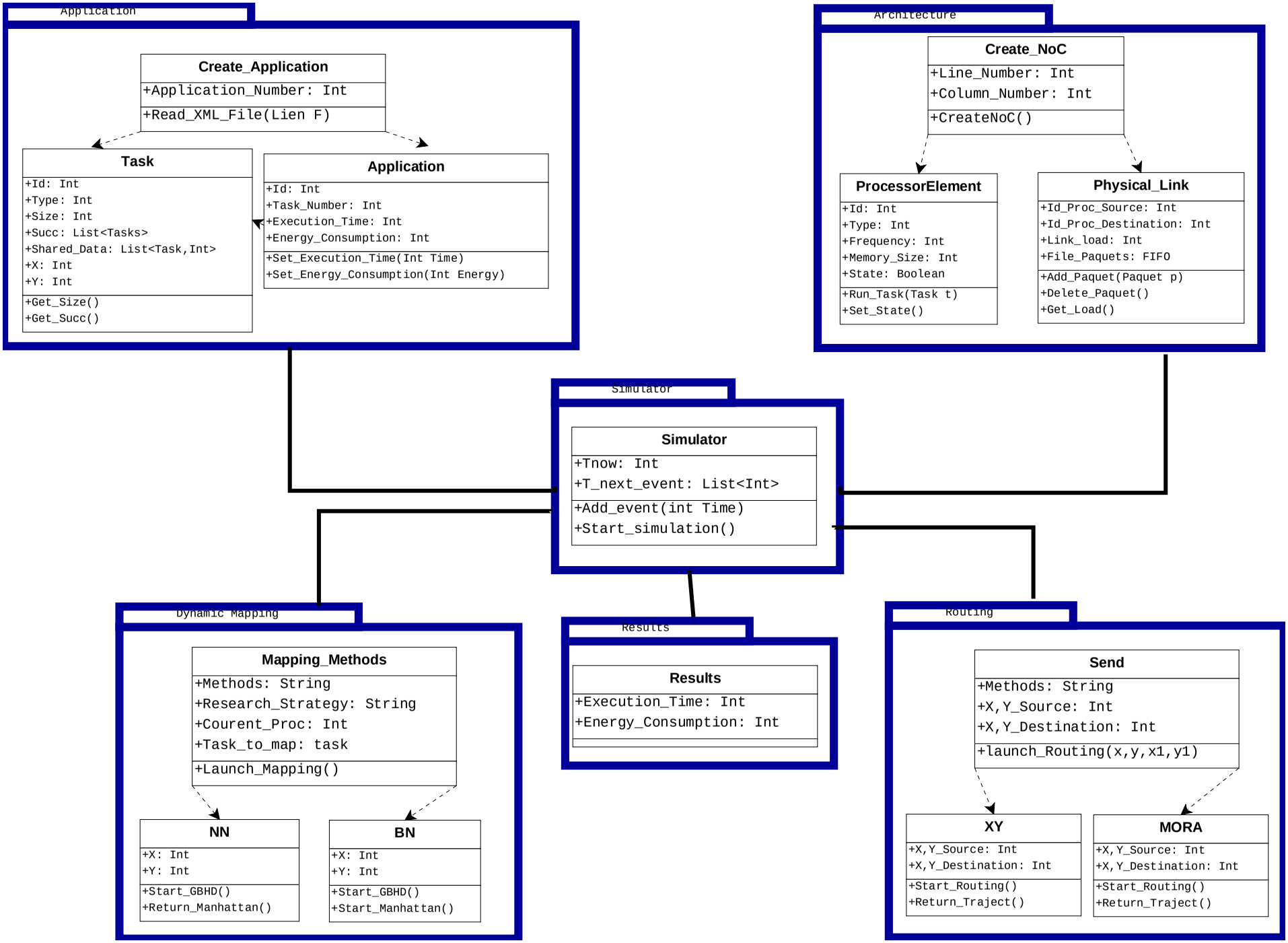


FIGURE 3.2 – Diagramme de classes du simulateur

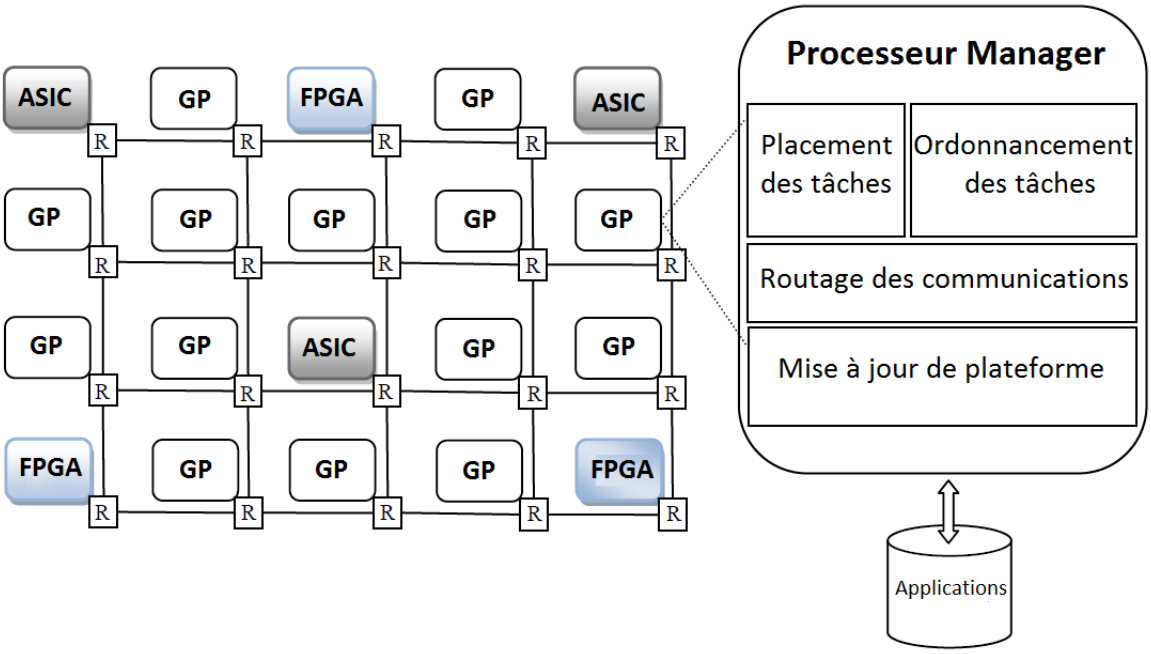


FIGURE 3.3 – Architecture hétérogène d'un NoC

- **ProcessorElement :**

  La classe ProcessorElement représente les processeurs. Chaque processeur est caractérisé par :

  — **Id :** identifiant du processeur,
  — **Type :** cet argument représente le type de processeur(GP, FPGA, ASIC, DSP),
  — **x,y :** c'est les coordonnées de la ligne et la colone dans le NoC,
  — **State :** variable qui détermine si le processeur est libre ou pas,
  — **Memory :** mémoire d'un processeur,

  — **Frequency :** c'est le nombre de cycles d'horloge par unité de temps (seconde, ms, ect),
  — **Energy :** c'est l'énergie consommée pendant l'exécution d'un cycle d'horloge, (en unité d'énergie "joul, mj, ect"),
  — **Mode :** variation de la fréquence d'exécution.

- **Physical_Link :** un lien physique a les arguments suivants :

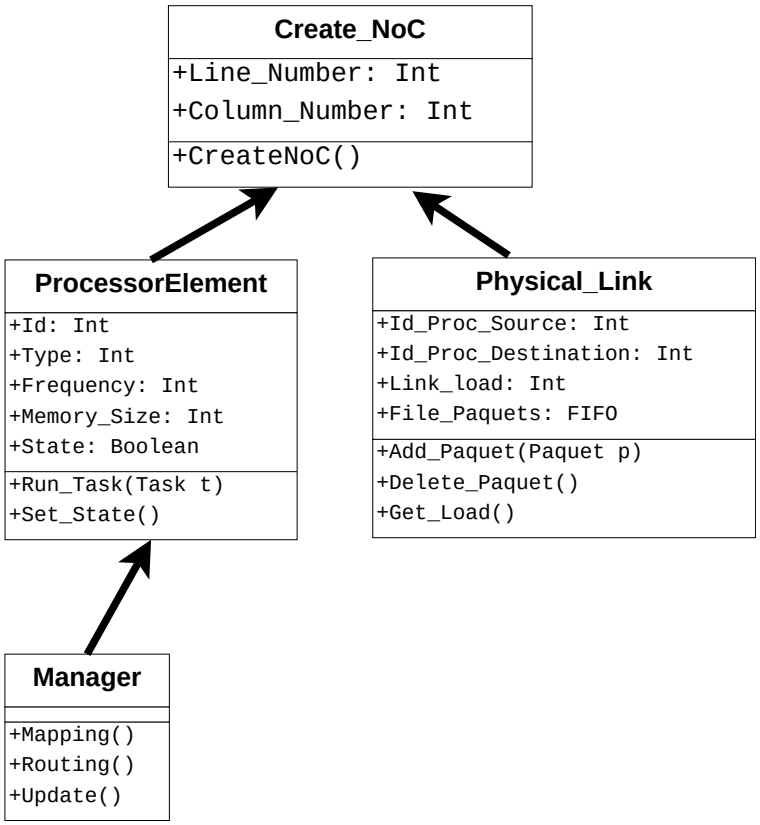


FIGURE 3.4 – Diagramme de classes du package Architecture

— **Id :** identifiant du lien,

— **Idprocsource :** Id du processeur source,

— **Idprocdestination :** Id du processeur destination,

— **Load :** représente la taille des données qui doivent transiter par ce lien.

3.4.4 Création d'applications

Une application est représentée par un graphe de tâches orienté TG = (T,L), où T est l'ensemble des tâches d'une application et L est l'ensemble des liens qui relient les tâches de cette application (figure 3.5.(a)). Chaque application a une seule tâche initiale et plusieurs tâches softwares et hardwares. La connexion entre deux tâches est une connexion maître-esclave (3.5.(b)). La tâche début de l'application est la tâche initiale, cette dernière n'a pas de maître. Les liens contiennent respectivement les données partagées entre chaque maître-esclave.

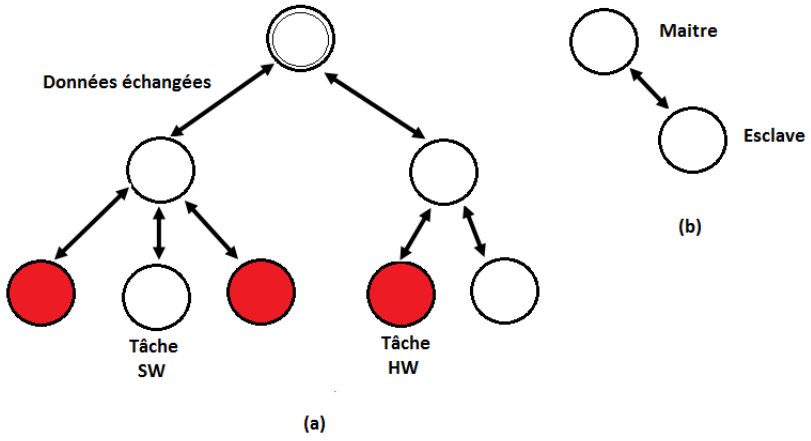


FIGURE 3.5 – Applications maitre/esclave

Représentation des applications :

La liste des applications est représentée par un fichier XML donné en entré. Les figures 3.6 et 3.7 montrent un exemple d'une application et sa représentation en fichier XML, respectivement.

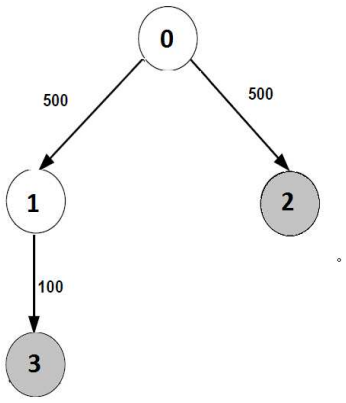


FIGURE 3.6 – Exemple d'une application

Diagramme de classes

Le diagramme suivant 3.8 montre les différentes classes du package application.

Application : Les principaux attributs de l'application sont :

— TaskList : la liste des tâches qui composent l'application,

— ExecutionTime : le temps d'exécution global de l'application,

— EnergyConsumption : l'énergie consommée lors de son exécution.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<list_applicatio>
                // la list d'application
      <application>   // Debut d'application

            <tache>    // Tache

            <id>0</id>  // Id de la tâche
            <Type_Taille>
            // Taille et Nombre de cycles selon le type de la tâche

           <taille_par_type Type="1"  Taille="300" Cycles="12000" />

            <taille_par_type Type="3"  Taille="180" Cycles="8000" />

            </Type_Taille>

            <succ>  //List des fils

            <id_succ id="1" data="500" /> // Id et données partagées

            <id_succ id="2" data="500" />

            </succ>

            </tache>

            <tache>  //Tache
                  <id>1</id>

                  <Type_Taille>

            <taille_par_type Type="1"  Taille="120" Cycles="9000" />

                 </Type_Taille>
                  <succ>
                  </succ>
            </tache>
            <tache>
                  <id>2</id>
                  <Type_Taille>

            <taille_par_type Type="2"  Taille="120" Cycles="7000" />
            <taille_par_type Type="3"  Taille="200" Cycles="10000" />

                  </Type_Taille>
                  <succ>
                  </succ>
            </tache>
            <tache>
                  <id>3</id>
                  <Type_Taille>

            <taille_par_type Type="3"  Taille="150" Cycles="7500" />
                  </Type_Taille>
                  <succ>
                  </succ>
            </tache>
                  </application>
</list_applicatio>
```

FIGURE 3.7 – Representation d'une application en XML

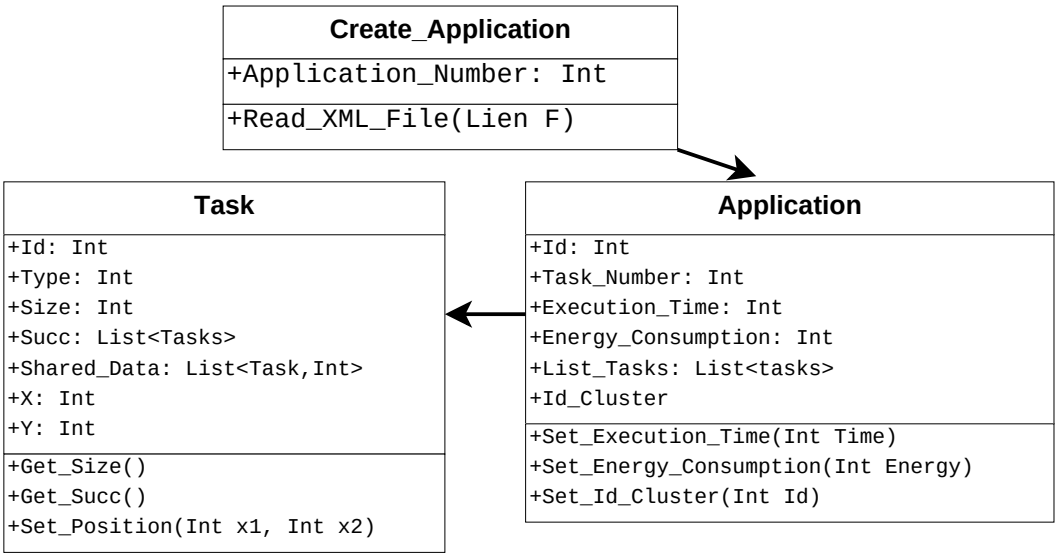


FIGURE 3.8 – Diagramme de classes du package Application

Task : Chaque tâche est associée aux attributs suivants :

- Id : Identifiant de la tâche,
- Type : le type de processeur sur lequel la tâche peut s'exécuter(GP, FPGA, etc.),
- Size : la taille de la tâche selon le type (en nombre d'unité de donnée "octet, bit, etc."),
- cycles : le nombre de cycles nécessaire à l'exécution de la tâche selon le type,
- SharedData : la taille de donnée partagée avec chaque fils (en unités de données),
- x,y : les coordonnées du processeur sur lequel la tâche sera placée.

Diagramme de séquence :

Le diagramme de séquence 3.9 illustre l'interaction entre la tâche maître et sa tâche esclave.

3.4.5 Modèle de simulation

Dans cette section nous expliquons l'intérêt et les concepts de la simulation.

La simulation est par définition une imitation ou une représentation d'un système par un autre. En tant que telle, plusieurs aspects de la simulation sont des

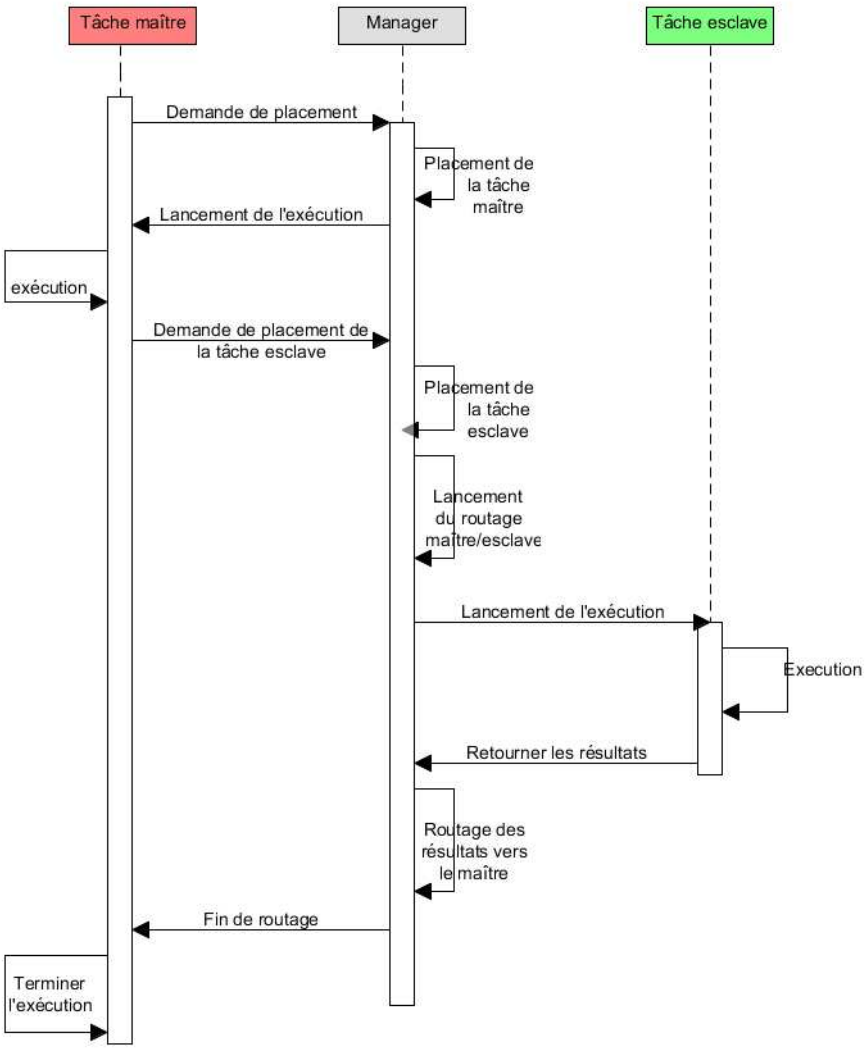


FIGURE 3.9 – Diagramme de séquence illustrant l'interaction entre maître/esclave

simplifications du système réel ou proviennent de déduction faites à partir d'études sur des systèmes similaires.

Quand un système est simulé, il est représenté comme un ensemble de modules qui sont interconnectés les uns aux autres. La manière dont ces modules s'exécutent et interagissent les uns avec les autres est appelée un modèle de simulation. Parmi les modèles de simulation on distingue la simulation à événement discret et la simulation à pas fixe.

Le simulateur que nous avons conçu utilise l'approche par événement discret. La prochaine section décrit cette approche.

La simulation à évènement discret :

En simulation à événement discret, seuls les points où l'état du système change dans le temps sont représentés. En d'autres termes, le système est modélisé comme une suite d'événements, c'est-à-dire les instants où les changements d'état du système se produisent. Des événements comme les arrivées des entités, le début de service d'une entité, une reprise de service d'une ressource où chacun se produit à un instant[119]. Dans la simulation à événements discrets, il existe une file d'attente globale dans laquelle est stockée la liste des événements qui vont se produire. L'avantage de cette technique se résume au fait que le système est vu comme étant composé d'une collection d'événements, chacun de ces événements s'exécutant instantanément vis-à-vis de l'horloge de simulation, changeant l'état de l'objet modélisé, sélectionnant et programmant l'événement successeur qui se réalisera dans le futur. L'inconvénient majeur de cette approche est que le programmeur doit explicitement partitionner un modèle en événements, ce qui rend la structure du modèle difficile à comprendre.

Les composants de base du modèle de simulation

Quelque soit le système simulé il doit comporter trois composants principaux :

- **entités :** sont des objets comportant des attributs propres qui influent dans le changement d'état du système,
- **files d'attente :** les entités attendent généralement dans une file d'attente jusqu'à être servies,
- **ressources :** traitent ou servent les entités en file d'attente. Ces ressources sont libres quand elles sont disponibles pour le traitement ou occupées quand elles traitent des entités.

  Par analogie dans notre système, ces composants sont représentés respectivement par :

- les applications,
- file d'attente des applications,
- l'ensemble des PEs et liens physiques.

La liste des événements du simulateur :

Le tableau 3.2 cite quelques événements, changement d'état ainsi que les événements programmés qui sont gérés par le simulateur.

Evénemnt	Changement d'état	Evénement programmé
Arrivée des Applications	Ajouter les applications à la file d'attente des applications	Placement des tâches initiales
Placements des tâches initiales	Rendre le cluster ocupée Rendre le PE du milieu du cluster occupée	Début exécution
Début exécution	Rendre le PE occupée	Fin exécution
Fin exécution	Libérer le PE	Placement des tâches fils
Placement dynamiques des tâches fils	Assigner une tâche à un PE	Début de routage
Début de routage	Modifier la charge des liens	Fin de routage
Fin de routage	Modifier la charge des liens	Début exécution

TABLE 3.2 – Liste des événements

Afin de mieux comprendre le déroulement des événements lors de la simulation ; le tableau 3.3 présente un exemple de la simulation du placement de 2 applications en parallèle. Chacunes de ces deux applications est composée de 2 tâches (une tâche initiale et sa tâche esclave).

Temps	Evénemnt
0	Arrivée de l'application 1 et 2 Placement statique des tâches initiales de l'application 1 et 2 Début exécution des deux tâches initiales
400	Demande de placement de la tâche esclave de l'application 1 Placement dynamique de la tâche esclave
500	Placement de la tâche esclave (application 1) terminé Début de routage (du maître vers l'esclave)
600	Demande de placement de la tâche esclave de l'application 2 Placement dynamique de la tâche esclave
700	Placement de la tâche esclave terminé de l'application 2 Début de routage (du maître vers l'esclave)
3100	Fin de routage entre la tâche maître et son esclave (application 1) Début exécution de la tâche esclave
3300	Fin de routage entre la tâche maître et son esclave (application 2) Début exécution de la tâche esclave
3600	Fin d'exécution de la tâche esclave (application 1) Lancement du routage (de l'esclave vers le maître)
3700	Fin d'exécution de la tâche esclave (application 2) Lancement du routage (de l'esclave vers le maître)
4100	Fin de routage entre la tâche esclave et son maître (application 1) Reprise d'exécution de la tâche maître
4300	Fin de routage entre la tâche esclave et son maître (application 2) Reprise d'exécution de la tâche maître
5000	Fin d'exécution de la tâche initiale (maître) (application 1) Fin d'exécution de l'application 1
5300	Fin d'exécution de la tâche initiale (maître) (application 2) Fin d'exécution de l'application 2 Fin de simulation

TABLE 3.3 – Exemple d'un déroulement de la simulation

Diagramme d'activité :

La figure 3.10 présente le fonctionnement global du simulateur.

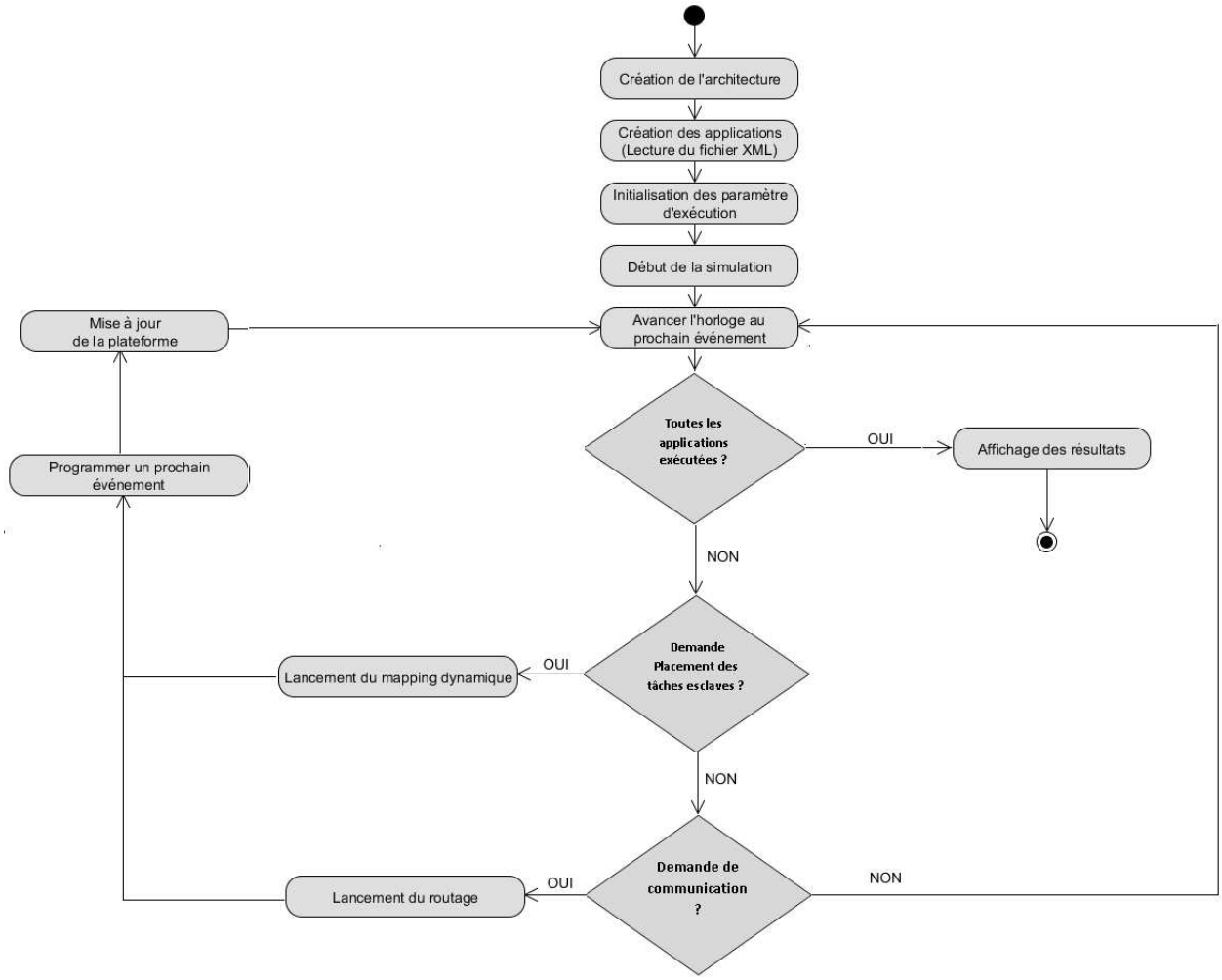


FIGURE 3.10 – Diagramme d'activité

Initialisations des paramètres d'exécution

Cette initialisation se fait à travers un fichier texte qui contient les paramètres suivants :

Nombre-Application	le nombre d'applications qui seront exécutées.
Type-de-Plateforme	détermine si les processeurs peuvent exécutés du MONO ou MULTI tâches.
Algorithme-de-placement	la liste des algorithmes de placement dynamique qui seront executés
Algorithme-de-routage	la liste des algorithmes de routages "XY" ou "MORA".
Temps-envoi	le temps que prend l'envoie d'un paquet entre 2 neouds(en Cycle d'horloge).
Energie-Consomee-Envoi	l'energie consommée lors de l'envoi d'un paquet.
Mode	détermine le mode d'exécution de chaque type de PE.
Architecure	Le chemin de l'architecture crée.
Application	Le chemin du fichier XML de l'application crée.

3.4.6 Mesures de performances

Le simulateur que nous avons mis en œuvre permet de simuler plusieurs techniques de placement dynamique d'applications sur une architecture NoC afin de mesurer et comparer les performances de ces techniques en termes de temps d'exécution et de consommation d'énergie.

3.5 Conclusion

Le manque d'outils qui permettent d'effectuer le placement dynamique d'applications sur des plateformes MPSoCs hétérogène basée NoC, de simuler le trafic et d'évaluer les performances des systèmes basés sur cette structure nous ont poussés à concevoir un nouveau simulateur. Ce chapitre a été entièrement consacré aux détails de la conception des différentes fonctionnalités de ce nouveau simulateur et à quelques tests qui ont été présentés à la fin du chapitre. Néanmoins de par sa conception orientée objet, cet outil est facilement extensible et peut être enrichi et complété par d'autres fonctionnalités s'intégrant dans la phase de conception des MPSoCs basée NoCs.

Chapitre 4

Heuristiques pour le routage et le placement dynamique de tâches en spirale sur un MPSoC basée NoC

Sommaire

4.1 Introduction

Les systèmes Multiprocesseurs (MPSoCs) sont une solution qui implémente un nombre multiple d'éléments de traitements (PEs) sur une même puce. L'avancée de la nanotechnologie prédit que les MPSoCs futurs contiendront des milliers d'unités de traitement (PEs) sur une seule puce d'ici 2015 [120]. Les MPSoCs sont de plus

en plus utilisés dans les applications embarquées complexes. Le Network-On-Chip (NoC) a été introduit comme un mécanisme d'interconnections entre les différentes unités de traitements qui se trouvent sur une même puce, permettant une énergie efficace, une connectique évolutive (scalable) [14, 121]. Le placement (Mapping) est une très importante phase dans l'exploration architecturale dans les systèmes MPSoCs basés NoC. Les plateformes d'architecture et d'application sont représentées par des graphes. Considérant le moment où le placement des tâches est exécuté, les approches peuvent être statiques ou dynamiques. Le placement statique définit le placement des tâches lors de la conception, qui dispose d'une vue globale sur les ressources du MPSoCs et comme il est exécuté lors de la conception, il utilise des algorithmes complexes pour mieux explorer les ressources d'un MPSoC, résultant d'une solution optimisée [122, 123, 124, 125, 126, 127, 128]. Cependant, le placement statique n'est pas capable de gérer la charge de travail dynamique, qui se présente sous forme de nouvelles tâches ou applications chargées lors de l'exécution. Pour faire face aux exigences de charge de travail dynamique sur les MPSoCS actuels, les techniques de placement dynamique sont nécessaires pour le placement sur les ressources de la plateforme [129, 130, 131, 98, 95, 100, 132]. Le but principal de ce chapitre est de présenter une nouvelle heuristique de placement dynamique de tâches d'applications en spirale. L'heuristique présentée est appliquée sur une plateforme MPSoC hétérogène. Deux types d'unités de traitements (PEs) sont considérées "instruction set processors" (ISPs) et "Reconfigurable Areas" (RA). "Instructions set processors" sont utilisés pour exécuter les tâches logicielles et "Reconfigurable Areas" pour les matérielles. L'heuristique essaye aussi de placer les tâches d'application en des régions de clusterisation pour réduire les surcharges des communications des tâches communicantes. L'heuristique proposée dans ce chapitre tente de placer les tâches d'une application les plus communicantes en spirale, dans le but de minimiser les coûts de communication. Un algorithme de routage basé sur l'algorithme de Dijkstra est également présenté dans ce chapitre. L'objective de placement des communications (routage) est de trouver la meilleure charge possible du chemin qui minimise les couts de communication. La nouvelle heuristique présentée montre une

amélioration de performances significative par rapport aux dernières heuristiques de placement rapportées dans la littérature. Les mesures de la performance incluent le temps d'exécution et la consommation d'énergie.

Le reste de ce chapitre est organisé comme suit :

La **Section 2** présente une classification des techniques de placement statique et dynamique à partir d'un état de l'art.

La **Section 3** présente l'architecture du MPSoC.

Dans la **Section 4**, la stratégie de placement des tâches en spirale et la stratégie de placement des communications à base de l'algorithme dijkstra sont présentées.

L'expérimental et les résultats sont présentés dans la **Section 5**.

La **Section 6** conclut ce chapitre.

4.2 Classification des placements statique et dynamique

Le placement des tâches sur la plateforme MPSoC consiste à trouver l'affectation des tâches sur les éléments de la plateforme qui satisfait certains critères d'optimisation comme la réduction de la consommation d'énergie, la réduction du temps total d'exécution et l'optimisation de l'occupation des canaux. Si la plateforme est hétérogène, alors le processus de "task binding" (liaison de tâches) est nécessaire avant de trouver le placement d'une tâche. Cela consiste à définir les ressources de la plateforme pour chaque type de tâche telle que « instruction set processors » pour les tâches logicielles et les « FPGA tiles » pour les tâches matérielles. Le placement des tâches est réalisé soit par placement statique (design-time) ou dynamique (run-time) [120].

4.2.1 Techniques de placement statique

La plupart des travaux existant dans la littérature qui traitent le problème de placement sur des architectures MPSoC basées NoC, sont du genre statique, ce qui définit le placement des tâches lors de la conception (design time). Cette démarche

consiste à avoir une vue globale sur les ressources de la plateforme MPSoC et les tâches d'applications (graphes de tâches). Etant donné, qu'elles sont exécutées lors de la conception, elles utilisent des algorithmes complexes pour mieux explorer les ressources de l'MPSoC, résultant de solutions optimisées. Des heuristiques comme l'approche génétique ou PSO et des méthodes exactes comme la recherche Tabou ou le recuit simulé, sont présentées dans [122, 124, 125, 127, 128]. Dans [123, 126] des algorithmes de placement pour une énergie efficace, sont présents. Ces techniques trouvent des placements fixes des tâches lors de la conception avec un comportement de calcul et de communications bien connu. Un état de l'art classifie pour les techniques de placements statique est montré dans la Table 4.1. Cependant, les placements statiques ne sont pas capables de gérer une charge de travail dynamique, où on a de nouvelles tâches ou applications chargées à l'exécution. Pour faire face à cette caractéristique des MPSoCs actuelle et future, les techniques de placement dynamique (run-time) s'avèrent nécessaires pour palier aux besoins de charge dynamique, et ainsi prendre en charge les placements dynamiques sur les ressources de la plateforme.

4.2.2 Techniques de placement Dynamique

Le défi dans les derniers travaux pour la résolution du problème de placement sur les MPSoCs hétérogènes basés NoC, est de présenter des techniques de placement dynamique pour le placement efficace des tâches ou d'applications qui demandent à être exécutées lors de l'execution.

Wildermann et al. [133] ont évalué les avantages de l'utilisation d'une heuristique de placement dynamique (fonctions de coûts de communication et de voisinage), qui permet de diminuer la surcharge de communication. Holzenspies et al. [103] ont étudié une autre technique de placement dynamique spatiale, et considèrent des applications de streaming pour le placement sur des MPSoCs hétérogènes, visant à réduire la consommation d'énergie imposée par de tels comportements d'application.

Schranzhofer et al. [134] suggèrent une stratégie dynamique basée sur les placements pré-calculés (modèles définis lors de la conception) utilisés pour placer les

REF	Tasks Mapping	Communications Mapping	Tasks Scheduling	Communications Scheduling	Voltage selection link	Voltage Selection tasks	Objectives	Guarantee hard deadlines
Y.Zhang and al 2002	List Scheduling	–	List Scheduling	–	–	ILP	Min Energy consumption	No
S.Kumar and al 2003	Genetic Algorithm	–	ASAP/ ALAP	–	–	Max	Performance time	No
R.Marculescu and al 2004	List Scheduling	Deterministic	List Scheduling	Deterministic	–	–	-Max performance time -Min Energy consumption	Yes
D. Shin and al 2004	Genetic Algorithm	Genetic Algorithm	List Scheduling	–	List Scheduling	–	Min Energy consumption	Yes
M.Armin and al 2007	List Priority (TPL, PPL)	–	–	–	–	–	Min Energy consumption	No
C.Chou and al 2009	Heuristic	Heuristic	–	Deterministic	–	–	Max Performance time	Yes
F.Vardi and al 2009	List Priority (TPL1, TPL2, RTPL, PPL)	–	–	–	–	–	- Max performance time -Min Energy consumption	No
X.Wang and al 2009	Heuristic	Deterministic	–	–	–	–	Min Energy consumption	Yes

TABLE 4.1 – Etat de l'art classé pour les techniques de placement statique

tâches nouvellement acquises sur les PEs au moment de l'exécution.

Carvalho et al. [135] évaluent les avantages et les inconvénients de l'utilisation des heuristiques de placement dynamique (par exemple la charge de chemin et le meilleur voisin), par rapport à ceux des heuristiques statiques (par exemple le recuit simulé et la recherche Taboo). L'approche de Carvalho a été prolongée par Singh et al. [120, 136], en utilisant une stratégie de packing qui va être détaillée en vue d'une comparaison avec nos propositions, ce qui réduit le coût de communication

dans la même plateforme MPSoC basée NoC. En outre, l'approche de Singh a été améliorée pour permettre un placement dynamique Muli-tâches des applications sur le même PE. Différentes heuristiques de placement ont été utilisées pour évaluer la performance. Selon les auteurs, le coût de communication de l'ensemble du système est réduit, ce qui diminue la consommation d'énergie.

Faruque et al. [105] proposent une approche décentralisée de placement dynamique à base d'agents, ciblant les MPSoCs hétérogènes à grandes échelles à base de NoC (un MPSoC de 32x64 est utilisé comme étude de cas). Les travaux les plus récents proposés dans la littérature, sont classés et montrés dans la Table 4.2. Les heuristiques de placement dynamique du plus proche voisin (Nearest Neighbor (NN)) et meilleur voisin (Best Neighbor (BN)) présentées dans les travaux de Carvalho et Moraes [108] et les deux démarches de placement dynamique sont aussi présentées dans le travail de Singh et al. [120] elles sont en outre utilisées pour l'évaluation et la comparaison de performance avec nos heuristiques proposées de placement dynamique.

4.3 Architecture MPSoC Hétérogène

L'architecture MPSoC utilisée dans ce travail contient un ensemble d'éléments de traitement différents, qui interagissent via un réseau de communication [14]. Les tâches logicielles s'exécutent sur "instruction set processors" (ISPs) et les tâches matérielles sur "Reconfigurable Area" (RA) ou sur des IPs dédiées. L'une des unités de traitement (PE) est utilisée comme processeur Manager (M) responsable de l'ordonnancement des tâches et applications (scheduling), la liaison des tâches (Task binding), le placement de tâche (Mapping), l'acheminement des communications, le contrôle de ressources et de reconfiguration. Le M ne connaît que les tâches initiales des applications. La première tâche de chaque application est lancée par le M et de nouvelles demandes de communications avec d'autres tâches sont demandées au fur et à mesure par les tâches déjà placées sur la plateforme du MPSoC au moment de l'exécution. Les tâches demandées sont chargées de la mémoire quand une communication est nécessaire.

Authors	Mono / Multi Tasks	Type of Architecture	Type of control	Optimizations Goals
Smit 2005	Mono	heterogeneous	Centralised	Energie Consumption
Ngouanga 2006	Mono	homogeneous	Centralised	Communication volume , Computational Load
Hölzenspies 2007/2008	Mono	heterogeneous	Centralised	Energie Consumption
Chou 2007/2008	Mono	homogeneous	Centralised	Energie Consumption
Al Faruque 2008	Mono	heterogeneous	Distributed	Execution time , Mapping time
Mehran 2008	Mono	homogeneous	Centralised	Mapping time , Energie Consumption
Wildermann 2009	Mono	homogeneous	Centralised	communication, latency Energie Consumption
Schranzhofer 2010	Mono	homogeneous	Centralised	Energie Consumption
Carvalho 2010	Mono	heterogeneous	Centralised	Communication volume Network contention
Singh 2009, 2010	Multi	heterogeneous	Centralised	Energie Consumption, Communication volume, Network contention
Marcelo 2011	Multi	homogeneous	Centralised	Tasks interdependency evaluation , energy consumption
Proposed Work	Mono	heterogeneous	Centralised	Communication volume, Network contention, Execution time , energie consumption

TABLE 4.2 – Etat de l'art classé pour les techniques de placement dynamique

4.4 Approches proposées

Cette section décrit nos approches. Tout d'abord, nous décrivons notre heuristique de placement dynamique des tâches d'applications avec une stratégie de packing en spirale. En second lieu, nous décrivons l'heuristique de placement des communications avec un algorithme de routage de Dijkstra modifié. D'abord, nous introduisons quelques définitions pour la bonne compréhension de l'approche propo-

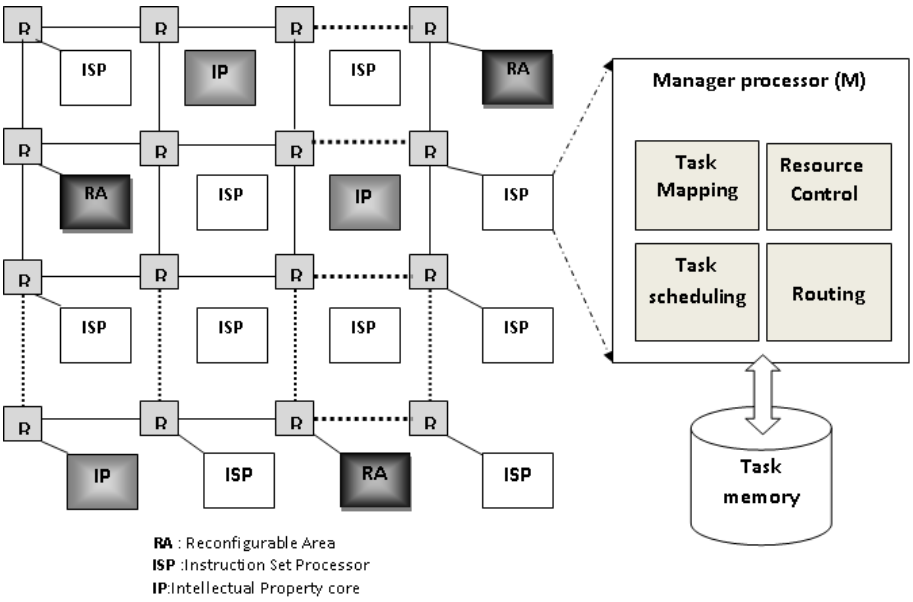


FIGURE 4.1 – Architecture conceptuelle MPSoC

sée. La première définit le modèle d'application, la deuxième le modèle d'architecture utilisé et la troisième la fonction de placement.

4.4.1 Définitions

Modèle de calcul d'application

Un graphe de tâches d'application est représenté par un graphe directe acyclique $TG = (T, E)$, où T est l'ensemble des tâches d'une application et E l'ensemble des liens de communications entre les tâches d'une application. La figure 4.2(a) décrit une application qui contient une tâche initiale, des tâches logicielles et des tâches matérielles avec tous les liens de communications les connectant (E) et la figure 4.2(b) montre une paire maître-esclave (communicating tasks). La tâche de départ d'une application est la tâche initiale qui n'a pas de tâche maître. E contient toutes les paires de tâches communicantes et est représenté par $(mt_{id}, st_{id}, (V_{ms}, V_{sm}))$, où mt_{id} représente l'identifiant de la tâche maître, st_{id} représente l'identifiant de la tâche esclave, V_{ms} est le volume de données du maître vers l'esclave, V_{sm} est le volume de données de l'esclave vers le maître.

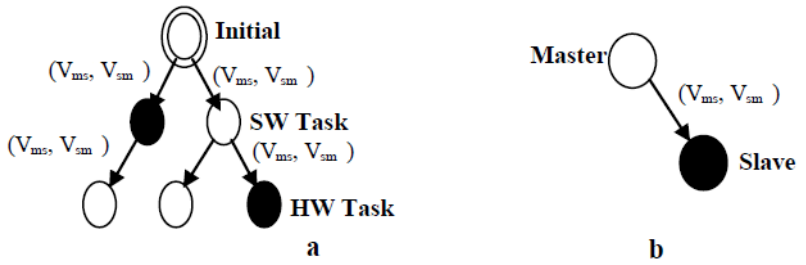


FIGURE 4.2 – Maître-Esclave et la modélisation de l'application

Modèle de calcul d'architecture

L'architecture MPSoC basée NoC est un graphe directe $AG = (P, V)$, où P est un ensembles de tuiles p_i. $v_{i,j}$, représente le canal physique entre deux tuiles p_i et p_j. Une tuile p_i, est constituée d'un routeur, une interface de réseau, un élément de traitement hétérogène, une mémoire locale et une mémoire cache.

Fonction de placement

Le placement des tâches d'application sur une plateforme MPSoC basée NoC est représenté par la fonction $mpg : t_i\ (\in T) \Rightarrow p_i\ (\in P)$.

4.4.2 Heuristiques de placement dynamique de référence

The First Free (FF)heuristic

Le FF consiste à trouver la première ressource libre en suivant un parcours des ressources colonne par colonne et de haut en bas, en commençant à chaque fois à partir de la première ressource (PE_{00}).

Minimum Maximum Channel load (MMC) heuristic

Cela considère tous les placements possibles pour une tâche donnée et choisit celui qui augmente le moins la charge maximale des canaux du NoC.

Algorithm 1 ALGORITHME FF

Entrées : tâche−à−Placer. **Sortie :** PE.x,PE.y ; /** Position du processeur sur le quel la tâche sera placé */

```
1: non-placé ← true ;
2: /*rechercher une ressource libre */
3: for i=0 to Nombre−PE−Total do
4:    /* Si le type de la tâche est le même que celui du PE */
5:    if (tâche−à−Placer.type = PE_i[X][Y].type) then
6:       /* Si PE est libre */
7:       if (PE_i[X][Y].isfree()) then
8:          non-placé ← false ;
9:          tâche-à-Placer.x← PE_i.x ;
10:         tâche-à-Placer.y← PE_i.y ;
11:      end if
12:   end if
13: end for
14: if non-placé then
15:   /* Mettre la tâche dans la liste des tâches non placé */
16:   List−tâche−non−palcé ← tâche−à−Placer ;
17: end if
```

Minimum Average Channel load (MAC)heuristic

Cela concerne tous les placements possibles pour une tâche donnée en choisissant celui qui augmente le moins la charge moyenne des canaux du NoC.

The Nearest Neighbor (NN) heuristic

Le NN recherche la première ressource libre parmi ses voisines. Pour mapper une tâche qui demande à être placée, l'algorithme essaye en premier lieu, de placer la tâche-fils sur un PE autour de la ressource sur laquelle la tâche-père a été placée avec un saut égal à 1. Les PEs sont recherchés avec la séquence gauche, bas, haut, droit. Si aucune ressource adéquate n'est trouvée, l'algorithme reprend la même séquence à partir des processeurs déjà parcourus jusqu'à atteindre la limite du NoC.

The Path Load (PL) heuristic

Calcule la charge dans chaque canal utilisé dans la voie de communication. PL calcule le coût de la voie de communication entre la tâche source et chacune des ressources disponibles. Le placement sélectionné équivaut à un coût minimal.

Algorithm 2 ALGORITHME DU NN

Entrées : tâche−à−Placer, X,Y **Sortie :** PE.x,PE.y ; /** Position du processeur sur le quel la tâche sera placé */

```
1: non-placé ← true ; Saut ← 1 ; i ← 0 ;
2: /*rechercher une ressource libre au Saut=1 suivant la séquence Gauche, Bas, Haut,Droit*/
3: rechercher−gauche−bas−haut−droit(tâche−à−Placer, X,Y) ;
4: /* A partir du saut=2 jusqu'à la limite du NOC, répéter la même séquence de recherche à
   partir des ressource déja consulté avant */
5: while (non-placé and (Saut ≤ NoC-limit )) do
6:   Saut++ ;
7:   rechercher−gauche−bas−haut−droit(tâche−à−Placer, X,Y-Saut) ;
8:   if (non-placé) then
9:     rechercher−gauche−bas−haut−droit(tâche−à−Placer, X+Saut,Y) ;
10:  end if
11:  if (non-placé) then
12:    rechercher−gauche−bas−haut−droit(tâche−à−Placer, X-Saut,Y) ;
13:  end if
14:  if (non-placé) then
15:    rechercher−gauche−bas−haut−droit(tâche−à−Placer, X,Y+Saut) ;
16:  end if
17: end while
18: if (non-placé) then
19:   List−tâche−non−palcé ← tâche−à−Placer ;
20: end if
```

Algorithm 3 RECHERCHER−GAUCHE−BAS−HAUT−DROIT

Entrées : tâche−à−Placer, X,Y **Sortie :** PE.X,PE.Y ;

```
1: /* Si x,y ne dépasse pas la limite du NOC */
2: if (0≤x≤NOC−LIMIT) and (0≤y≤NOC−Limit) then
3:   /* Si le type de la tâche est le même que celui du PE */
4:   if (tâche−à−Placer.type=PE[X][Y].type) then
5:     /* Si PE est libre */
6:     if (PE[X][Y].isfree()) then
7:       non-placé ← false ;
8:       tâche-à-Placer.x← PE.x ;
9:       tâche-à-Placer.y← PE.y ;
10:    end if
11:  end if
12: end if
```

The Best Neighbor (BN) heuristic

Le BN utilise une stratégie de recherche présentée dans le travail de Singh ; cet algorithme n'affecte pas la tâche à la première ressource trouvée (NN) mais celle qui a le lien le moins chargé.

4.4.3 Heuristique proposée basée sur la stratégie de packing en spirale et le routage dynamique

Stratégie de packing en spirale

Pour placer les applications, on procède d'abord par le placement des tâches initiales de manière à les éloigner le plus possible lors du placement en choisissant en priorité les milieux des clusters, en utilisant une stratégie de clustérisation comme montré dans la figure 4.3. Ce qui permet que le placement des tâches d'une même application dans une même région, réduisant ainsi les coûts de communication. Les frontières des clusters sont virtuelles et les régions communes pourraient être partagées par les tâches des différentes applications. Après le placement des tâches initiales d'applications, les tâches communicatives demandent à être placées. Pour cela, le processeur maître (M) essaye de placer la tâche communicante qui demande à être exécutée autour du processeur exécutant la tâche appelante et cela à partir d'une distance égale à 1 (hop) jusqu'à la limite du NoC. La ressource (le processeur en fonction du type de la tâche) est recherché avec une stratégie de packing en spirale selon le séquencement 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , comme le montre la Figure 4.3. On explore les voisins en spirale en effectuant le placement de telle façon qu'il empêche le calcul du coût de toutes les solutions possibles, comme dans le PL et le calcul du meilleur voisin comme dans l' heuristique BN. Ce qui réduit le temps d'exécution pour le placement.

Algorithme de routage dijkstra modifié

Après avoir placé les tâches communicantes, nous avons besoin d'un placement des communications. Notre nouvelle heuristique de placement des communications proposée tente de rechercher un meilleur chemin avec une bande passante la moins éle-

Algorithm 4 SPIRAL HEURISTIC

Entrées : NoFreeResources , CurrentProcessor, PE, FreeResources, HopDistance, NocLimit **Sortie :** FreeElement

```
1: HopDistance ← 0
2: HopX ← 0
3: HopY ← 0
4: while (FreeElement = NULL) AND HopX != NocLimit AND HopY != NocLimit do
5:   if (CurrentProcessor.left.isFree()) then
6:     FreeElement ← CurrentProcessor.left ;
7:   else
8:     HopX- - ;
9:   end if
10:  if CurrentProcessor.top.isFree() then
11:    FreeElement ← CurrentProcessor.top ;
12:  else
13:    HopY- - ;
14:  end if
15:  if CurrentProcessor.right.isFree() then
16:    FreeElement ← CurrentProcessor.right ;
17:  else
18:    HopX+ + ;
19:  end if
20:  if CurrentProcessor.bottom.isFree() then
21:    FreeElement ← CurrentProcessor.bottom ;
22:  else
23:    HopY+ + ;
24:  end if
25: end while
```

vée. L'heuristique proposée réduit le temps d'exécution et de consommation d'énergie.

Algorithm 5 MODIFIED DIJKSTRA ROUTING ALGORITHM

Entrées : IdSource , IdDestination **Sortie :** BestTraject

```
   /*While we did not reach the destination */
2: while (stop=false) do
     /*Got back the neighbor who has the least used link*/
4:   min ← minWeight() ;
     if min <> idDest then
6:     /*Add the link in the list of the path*/
       Listpath.add(min)
8:   else
       stop ← true
10:  end if
   end while
12: BestTraject ← Listpath
```

4.5 Configurations expérimentales et Résultats

Pour l'expérimentation nous avons utilisé un langage à haut niveau d'abstraction à savoir JAVA qui nous a permis rapidement de comparer différentes heuristiques

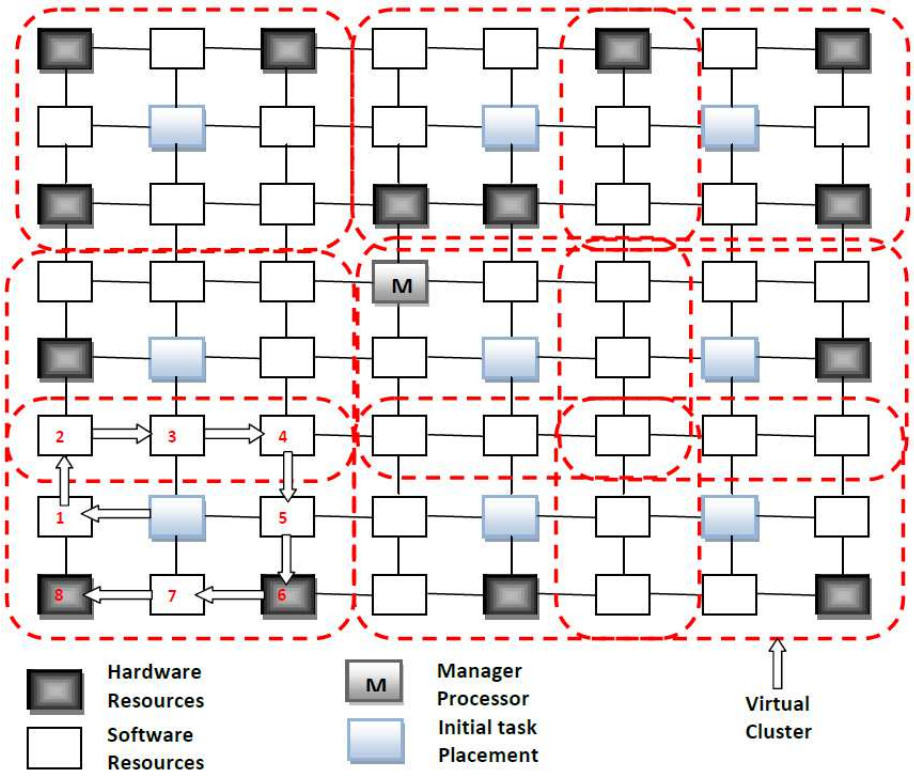


FIGURE 4.3 – Placement des tâches initiale et la strategie de packing en spirale.

de placement dynamique sur une architecture MPSoC basée NoC.

4.5.1 Configurations expérimentales

Nous avons réalisé une plateforme hétérogène simulée de 64 unités de traitements (8x8) dont 14 de type matériel (RA), 49 de type software et une unité de traitement utilisée comme Manager responsable du placement des tâches d'applications. La configuration, la mise à jour de la plateforme, Le placement des communications (communications routing). Cette plateforme utilise un réseau sur puce pour la communication. Nous avons utilisé un fichier XML pour la description des graphes de tâches utilisées qui sont les mêmes que dans le travail de A.K.Singh, tâches (initiale, software et hardware). Leur temps d'exécution dépend de la spécificité et de la capacité du processeur. Nous avons fixé des paramètres, pour un processeur logiciel dont l'exécution d'une instruction nécessite 40 cycles. Par contre un processeur matériel est plus rapide et nécessite 20 cycles seulement,mai il consomme plus d'énergie que le processeur logiciel (on les a fixé a 20 et 10 cycles respectivement). La taille

des tâches est représentée en nombre d'instructions. La taille des données échangées est fixée en 100 paquets qui peuvent être changés. Le scénario utilisé exécute une, trois, sept et dix applications où chaque application contient de 7 à 9 tâches. La plateforme est divisée en 09 clusters qui permettent de lancer neuf (09) applications en parallèle. Ceci n'est pas une contrainte car n'importe quel nombre d'application peut être exécutée sur la plateforme en utilisant une fille d'attente quand le nombre est supérieur à 9. Pour le placement des tâches d'applications, nous avons implémenté notre algorithme de placement dynamique de tâches basées sur la stratégie de packing en spirale et la méthode de routing basée sur le dijkstra modifié. La technique en spirale essaye de placer les tâches les plus communicante avec un minimum d'exploration de l'espace du NoC. L'implémentation de notre méthode de placement des communications (modified Dijkstra routing) a minimisé considérablement le temps d'exécution et la consommation d'énergie du système. Pour une comparaison avec nos propositions nous avons implémenté aussi les heuristiques de placements de référence dans la littérature NN et BN.

4.5.2 Résultats expérimentaux

Nous avons simulé l'exécution des méthodes de placement dynamique Nearest Neighbor (NN) et Best Neighbor (BN) pour le scenario de placement d'une, trois, sept, et dix applications en parallèle sur la plateforme simulée de 64 unités de traitements (8x8) dont 14 unités de traitements hardware, 49 unités software et une unité de traitement Manager (M). Nous avons aussi simulé l'exécution de nos heuristiques, proposé des placements dynamiques en spirale des tâches et des communications. Pour les mesures de performances nous avons calculé le temps d'exécution et la consommation d'énergie. La Figure 4.4 montre l'optimisation apportée par notre proposition en terme de temps d'exécution par rapport à l'utilisation des deux autres méthodes de référence. La figure 4.5 montre la réduction d'énergie apportée par l'utilisation de notre approche qui donne de meilleurs résultats par rapport aux deux autres méthodes implémentées.

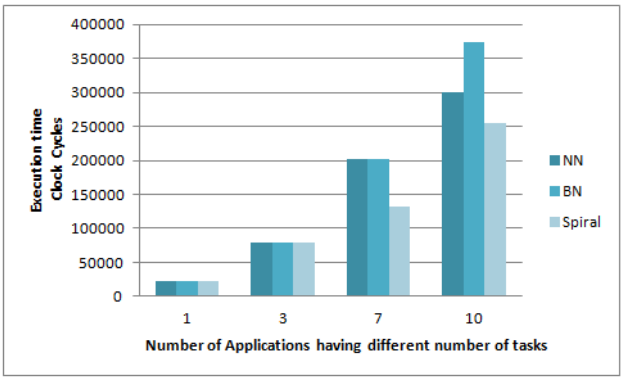


FIGURE 4.4 – Comparaison du temps d'execution de l'approche proposée avec NN et BN respectivement

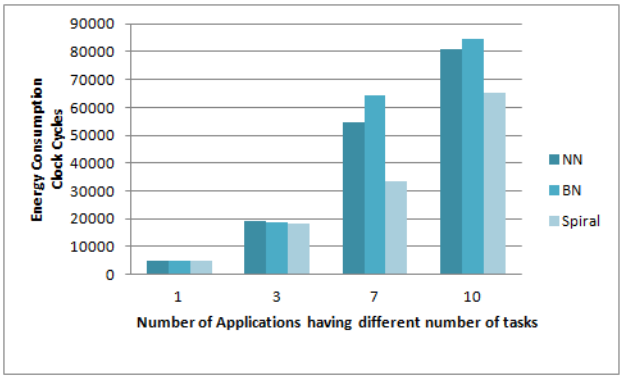


FIGURE 4.5 – Comparaison de la consommation d'énergie de l'approche proposée avec NN et BN respectivement

4.6 conclusion

Une nouvelle heuristique de placement dynamique de tâches, qui essaye de placer les tâches d'applications les plus communicantes, de façon à réduire le coût des communications et minimiser l'espace de recherche dans le NoC. Pour réduire au maximum le coût des communications, un routage Dijkstra modifié est également proposé dans le présent travail. Grâce à l'environnement de simulation que nous avons réalisé, nous avons fait une étude comparative entre le plus proche voisin (NN), le meilleur voisin (BN) et l'heuristique de placement dynamique proposée basée sur une stratégie de packing en spirale. Avec le placement des communications avec un algorithme de Dijkstra modifié, nous avons montré l'optimisation offerte par notre approche. Le défit du prochain travail présenté dans le chapitre suivant est la proposition de nouvelles stratégies de recherches et approches de routage dynamique.

Chapitre 5

Heuristiques pour le placement dynamique de tâches et communications sur un MPSoC basée NoC

Sommaire

5.1 Introduction

Les systèmes embarqués intensif utilisent des Systèmes Multiprocesseurs sur puce (MPSoCs), qui offrent un parallélisme accru pour atteindre la haute performance [6]. Pour faire face aux limites de processeur à usage général unique, la demande croissante de calcul et les exigences de performance. Un MPSoC contient plusieurs éléments de traitement (PEs) dans la même puce. Il existe deux types de MPSoCs : homogènes et hétérogènes. Un MPSoC homogène contient des PE identiques [86, 23, 66, 137, 16], alors que différents types de PEs sont intégrés dans un MPSoC hétérogène [138, 4, 139, 140]. Le réseau sur puce (NoC) a été introduit comme une interconnexion évolutive et à énergie efficace pour soutenir la communication entre les PEs [14, 15, 46].

Le concepteur doit placer les tâches de l'application sur les différentes ressources de traitement du MPSoC. Les techniques de placement statique définit le placement de la tâche au moment du design, d'avoir une vue globale des ressources MPSoC. Ces techniques de placement peuvent utiliser des algorithmes complexes afin de mieux explorer les ressources MPSoC en vue d'obtenir des solutions optimisées [122, 123, 126]. Les techniques de placement dynamique (run-time) sont nécessaires pour gérer ces charges de travail variables (dynamiques) [98, 95, 100, 132, 108, 133]. Ces techniques trouve le placement des tâches sur les ressources MPSoC au moment de l'exécution. Les dernières approches de placement dynamique, essayent de placer les tâches communicantes sur les PEs voisins disponibles, c'est à dire le plus près les un des autres afin de réduire la surcharge de communication [135, 120]. Cependant, ces approches ne fonctionnent pas bien lorsque les applications contiennent un grand nombre de tâches. En outre, la plupart des travaux de placement rapportés dans la littérature utilisent un algorithme de routage déterministe à savoir l'algorithme de routage XY [120, 100, 132, 133, 103, 135, 105, 141]. Cependant, pour un système qui doit gérer un flux de travail dynamique, l'utilisation d'une méthode de placement dynamique des communications peut conduire à de meilleurs résultats.

Dans ce chapitre nous présentons une stratégie de packing Manhattan qui permet d'explorer et de placer les tâches d'application plus rapidement que les der-

nières stratégies de placement existantes, entraînant des coûts optimisés de placement des tâches. Nous présentons également un algorithme de routage dynamique multi-objectifs (MORA) qui réduit les coûts de communications par rapport aux approches de routage souvent employées. Le modèle utilisé pour la représentation des applications est le modèle maître-esclave. Ce type de modèle est utilisé pour représenter les applications qui ont des tâches communicantes parallèles. La plateforme MPSoC hétérogène considérée contient deux types de PEs : "Instruction Set Processeurs" (ISPs) et "Reconfigurable Areas" (RAs), qui exécutent des tâches matérielles et logicielles, respectivement. Toutefois, les PEs peuvent être configurés pour exécuter divers types de tâches. Pour le placement dynamique des tâches, les techniques de placement de l'état de l'art réduisent les coûts de communication par le placement des tâches communicantes sur les plus proches PEs disponibles [100, 135, 120]. Les derniers travaux utilisent une stratégie de packing pour réaliser cet objectif. Cependant, la plupart d'entre eux ne mettent pas l'accent sur la minimisation du temps de recherche (placement). Dans notre stratégie de packing Manhattan proposée, nous cherchons non seulement de placer les tâches communicantes sur les plus proches PEs disponibles, mais aussi de minimiser le temps de recherche. En outre, les approches existantes utilisent des techniques de routage déterministes pour faciliter la communication. En revanche, le MORA dynamique tente de trouver le chemin point a point de la communication qui a la charge la plus faible (la plus large bande passante), résultant à un temps d'exécution et consommation d'énergie optimisés. La stratégie de packing Manhattan montre des améliorations significatives de performance par rapport aux dernières approches de placement. Les approches de placement dynamique utilisant la méthode de routage MORA conduisent aussi à des améliorations de performances par rapport aux approches employant d'autres méthodes de routage tel que XY. Le reste du chapitre est organisé comme suit. Section 5.2 donne un aperçu de l'état de l'art. La section 5.3 décrit le modèle de l'architecture MPSoC considérée. Dans la section 5.5, la stratégie de packing Manhattan proposée et l'algorithme de routage multi-objectif (MORA) ont été présentés. La configuration expérimentale et les résultats sont présentés dans la section 5.6. La section 5.7

conclut le chapitre.

5.2 État de l'Art

Le placement des tâches dans une plateforme MPSoC nécessite de trouver le placement des tâches sur les ressources de la plateforme en vue de certains critères d'optimisation comme la réduction de la consommation d'énergie, le temps total d'exécution et l'optimisation de l'occupation des canaux de communications. Le placement peut être réalisé soit par des techniques de placement statiques (au moment du design) ou dynamique (au moment de l'exécution).

5.2.1 Techniques de placement statique

La plupart des travaux existants dans la littérature pour résoudre le problème de placement sur une plateforme MPSoC sont des techniques de placement statique [122, 123, 125, 127, 128, 142, 143]. Méta-heuristiques comme l'approche génétique [83, 84] et des méthodes comme la recherche Tabou [85, 81] et le recuit stimulés [80, 87] sont présentées. Ces techniques trouvent un placement fixe des tâches au moment du design avec un comportement d'exécution et de communication bien connu. Cependant, le placement statique n'est pas capable de gérer la charge de travail dynamique des tâches ou des applications qui ont besoin d'être chargé dans le MPSoC à l'exécution. Les techniques de placement dynamique (run-time) sont nécessaires pour gérer le placement de ces charges de travail sur les ressources de la plateforme.

5.2.2 Techniques de placement dynamique

Les derniers travaux rapportés dans la littérature traitent le problème de placement dynamique des tâches des applications sur les architectures MPSoCs en optimisant les différentes métriques de performance.

Wildermann et al. [133] évaluent l'avantage d'utiliser une heuristique de placement dynamique, qui diminue le coût de communication. Une fonction de coût de voisinage a été utilisé pour réduire les coûts de communication.

Holzenspies et al. [103] étudient une autre technique de placement dynamique spatiale pour le placement d'applications de streaming sur MPSoC hétérogène, Visant à réduire la consommation d'énergie.

Schranzhofer et al. [134] proposent une stratégie de placement dynamique basée sur les placements pré-calculées(modèle définis au moment de la conception), qui sont utilisés pour définir le placement des tâches nouvellement arrivés aux PEs lors de l'exécution.

Chou et al. [98] utilisent une plateforme NoC avec multiples niveaux de voltage. leur technique de placement est basée sur un algorithme de sélection de région qui minimise la consommation d'énergie de communication. L'energie de la communication est diminué de 50%. La technique de placement est appliquée a une plateforme NoC homogène.

In [95], les auteurs intègrent les informations de comportement de l'utilisateur dans le processus d'allocation des ressources. Cela permet au système de mieux répondre aux changements en temps réel et à s'adapter aux besoins des utilisateurs de façon dynamique.

Ost et al. [144] proposent une technique de placement dynamique a énergie efficace qui minimise la consommation d'énergie de 16% dans le meilleur cas.

Mehran et al [100] proposent une technique de placement dynamique en spirale (DSM) pour le placement de tâches durant l'exécution. le placement d'une tâche est recherché dans un chemin en spirale du centre vers la limite du NoC, de sorte que les tâches communicantes peuvent être placées à proximités les unes des autres. ils tentent également de réduire le temps d'exécution en réduisant le temps de placement dynamique, temps de re-configuration et le temps de migration de tâche.

Faruque et al. [105] proposent une approche de placement décentralisée a base d'agent ciblant les grandes MPSoCs hétérogènes à base de NoC tels que les systèmes 32x64. L'heuristique proposée place les applications d'une manière décentralisée en utilisant une approche basée sur les agents. Plusieurs agents sont utilisés pour réaliser la gestion de ressources. Il existe deux types d'agents : Global Agent (GA) et Cluster Agents (CAs). Toute la plateforme est divisée en petits clusters et chaque CA mis à

jour ces connaissances de ses ressources de cluster. Le GA conserve des informations globales sur tous les clusters. Les agents négocient les uns avec les autres pour trouver un PE appropriés pour le placement d'une tâche. Le placement a base d'agents réduit l'effort de calcul et d'observation du trafic pour le processus de placement, par rapport aux approches centralisées.

Carvalho et al. [108] présentent des heuristiques pour le placement dynamique de tâches en deux phases. La première phase trouve le placement des tâches initiales des différentes applications sur l'architecture MPSoC, tandis que la deuxième phase utilise des mèthodes différentes (par exemple, First Free (FF), Nearest Neighbor (NN), Minimum Maximum Channel Load (MAC) et Path Load (PL)) pour trouver le placement des autres tâches a la volée en fonction des demandes de communications et les charges dans les liens des NoCs. La charge canal NoC, la congestion et la latence paquet se réduit lorsque l'on utilise des méthodes différentes. l'architecture MPSoC basée NoC ciblé contient des PEs supportant chacun mono-tâche.

In [135], Les auteurs évaluent les heuristiques de placement dynamique et les compare avec les techniques de placement statique comme le recuit simulé et la recherche tabou.

Singh et al. [120] ciblent l'architecture MPSoC hétérogène contenant des PEs logiciel et matériel. Dans l'architecture, parmis les unités de traitement disponible, une seul unité de traitement agit comme un processeur manager qui est responsable de la liaison de tâche (Task binding), placement de tâche, la migration de tâche, contrôle de ressources et le contrôles de re-configuration. L'état de la ressource est mis a jour lors de l'exécution et le processeur Manager assure le suivi de l'information sur l'occupation des ressources. Leurs heuristiques de placement place les tâches communicantes d'une application a proximité les unes des autres de tel sorte à minimiser le coût de communication en vue d'améliorer la performance globale.

L'heuristique [120] examines les ressources disponibles avant de recommander les tâches adjacentes sur le même PE. Le processus de placement est effectuée en deux phases. Tout d'abord, les tâches initiales sont placées au centre des clusters qui sont obtenus en divisant le NoC en régions. Par la suite, les tâches communicantes sont

demandées et placées par leurs approches de placement proposées.

en général, les travaux proposés dans [108] sont étendues dans [120] en employant une stratégie de packing qui minimise le surcoût de la communication dans la plateforme MPSoC basée NoC.

Une heuristique d'énergie efficace pour le placement dynamique de tâche, nommée consommation d'energie basse basée sur les dépendances des voisinages (LEC-DN) a été présenté dans [145, 132]. La principale fonction de coût ici n'est pas seulement la distance en sauts entre les tâches communicantes, mais également la proximité du nombre de sauts et le volume de communication entre les tâches, puisque le nombre de flits transmises définit l'énergie de communication. Lorsque la tâche cible a une seule tâche communicante qui a déja été placé, LEC-DN utilise la recherche du plus proche voisin (NN) d'une manière en spirale. D'un autre côté, s'il y'a plus qu'une tâches communicantes qui sont déjà placés, il recherche un PE à l'intérieur de la zone de délimitation défini par la position d'une telle tâche en fonction du volume de communication.

In [146], un placement dynamique decentralisée axée application et ressources efficace a été proposé, où les tâches peuvent être intégrés progressivement à une tâche prédécesseur déjà placé. Il s'agit d'une approche auto-embarqué qui est entièrement décentralisé et autonome.

Les approches de placement dynamique existantes consomme un grand temps d'exploration pour trouver le placement des tâches. La stratégie de Manhattan proposée effectue l'exploration plus rapide avec objectifs de placer les tâches communicantes sur des PEs plus proche entre eux. Les heuristiques de placement du plus proche voisin (NN) et meilleur voisin (BN) présentées dans [135] ainsi que la stratégie de packing présenté dans [120] sont pris pour l'évaluation et la comparaison de performance par rapport a notre stratégie de packing Manhattan. La plupart des travaux existants utilisent un algorithme de routage déterministe, a savoir XY [120, 100, 132, 133, 103, 135, 105].

Toutefois, pour un système qui a une charge de travail dynamique, l'utilisation d'une méthode de routage dynamique peut conduire a des améliorations significa-

tives. Le NN et BN emploient l'algorithme de routage XY, alors que notre approche de placement emploie l'algorithme de routage dynamique multi-objectif (MORA)et donne de meilleurs résultats.

5.3 Modélisation du problème de placement et heuristiques de référence

5.3.1 Graphe de tâches d'application

Un graphe de tâches d'application est représenté comme un graphe acyclique orienté$TG = (T,E)$, où T est l'ensemble de toutes les tâches d'une application et E est l'ensemble des liens entres les tâches d'une application. Figure 5.1(a) montre une application qui a des tâches initiale, logiciel et matériel avec les arêtes (E) connectant ces tâches. Figure 5.1(b) montre la paire Maître-Esclave (communicating tasks). La tâche de départ d'une application est la tâche initiale qui n'a pas de maître. Chaque tâche est associée avec les attributs suivants : l'identifiant de la tâche t_{id}, le type de la tâche t_{type} (hardware, software, initial), et le temps d'exécution de la tâche t_{exec} sur chaque type de PE. L'ensemble de liens E contient toutes les arêtes reliant les tâches communicantes. Chaque lien (5.1 (b)) à les attributs suivant : l'identifiant de la tâche maître mt_{id} représentant la tâche maître connecté, l'identifiant de la tâche esclave st_{id} représentant la tâche esclave connecté, le volume de données envoyé du maître à l'esclave (V_{ms}), et le volume de données envoyé de l'esclave au maître (V_{sm}). S'il ya de multiples tâches esclave communiquant avec la tâche maître , les tâches esclaves sont placés dans l'ordre de leur numéro d'identification assignée. Ces identifiants de tâches esclave sont attribués au moment du design pour optimiser les performances, basée sur les arêtes et les couts de communications entres les tâches maître-esclave. Par exemple, soit l'identificateur de la tâche maître est de 0 et les identifions des ces deux tâches communicantes sont 1 et 2, puis premièrement la tâche avec l'identifiant 1 demande a être placé. Pour transmettre et recevoir des messages par une tâche, notre algorithme de routage (MORA) choisit le chemin le plus court et la charge des liens point a point qui dispose d'une faible charge dans

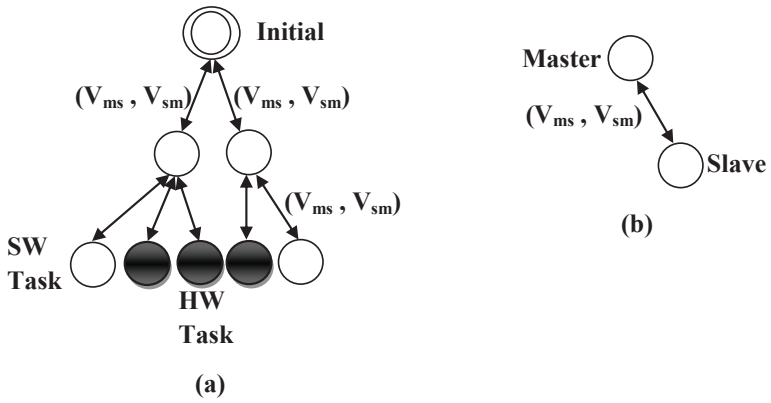


FIGURE 5.1 – Modélisation du graphe de tâches d'application et la pair Maître-Esclave

l'architecture MPSoC.

5.3.2 Graphe d'architecture MPSoC hétérogène basée NoC

Fig. 5.2 montre le modèle de l'architecture MPSoC hétérogène utilisé dans ce travail. L'architecture comprend un ensemble de différents éléments de traitement (PEs) qui interagissent via un réseau de communication [14]. Les PEs peuvent être de divers types, tels que instruction set processors (ISPs), reconfigurable logics (reconfigurable area-RA), dedicated intellectual properties (IPs), etc. Tâches à exécuter sur les PEs sont classés en tant que logiciel et matériel, qui mettent en œuvre normalement des fonctions simple et calcules intensives, respectivement. Les tâches logicielles s'exécutent sur les ISPs et les tâches matérielles s'exécutent sur RAs ou IPs dédiée. ISPs exécute les tâches logiciels efficacement. L'induction des RAs dans la plateforme offre une flexibilité au matériel à un niveau similaire à la programmabilité ISPs. Cependant, la hausse des overheads de reconfiguration des RAs doivent être prises en compte.

Le réseau de communication nécessaire pour faciliter la communication entre les PEs est disposés dans une topologie 2D maillé [108], comme illustré dans Fig. 5.2. Protocole de communication réseau utilise la commutation de paquets wormhole, le contrôle de flux handshake, les buffers d'entrée et l'algorithme de routage XY. Dans le routage XY, les paquets sont transférés en premier dans la direction X, puis dans la direction Y affin de les transférer à partir de PE source au PE de destination. En plus

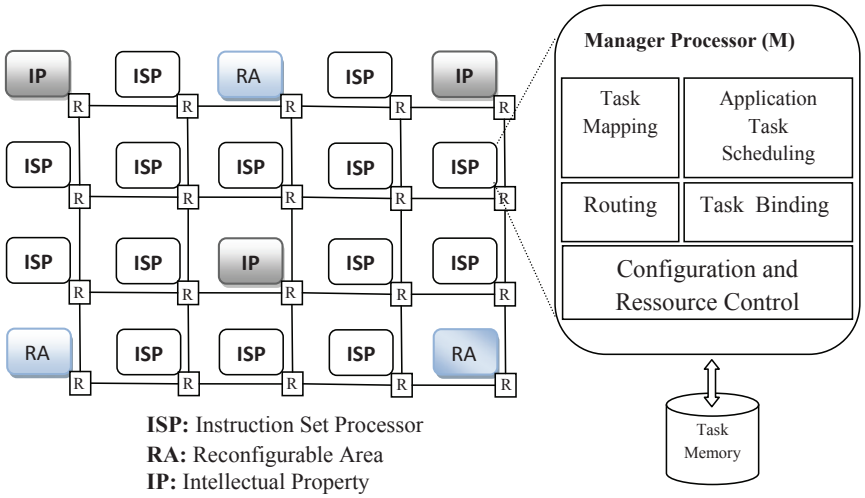


FIGURE 5.2 – Architecture MPSoC hétérogène

du routage XY, notre algorithme de routage dynamique multi-objectif (MORA) a également été incorporé, qui sera détaillé dans la section suivante. La communication inter-tâches est supporté par un mécanisme de passages de messages similaires à celui utilisé dans [108].

L'architecture MPSoC hétérogène basée NoC est un graphe orienté $AG = (P, V)$, où P est l'ensemble de tuiles et V représente les canaux physique entre les tuiles. Chaque tuile (in P) à les attributs suivants : l'identifiant de la tuile p_{id}, l'adresse de la tuile p_{add} qui est utilisée pour recevoir les paquets transmis a partir d'autres tuiles, type de la tuile p_{type} (hardware, software, initial). Chaque canal physique conserve les informations en paquets et le pourcentage d'utilisation de la bande passante disponible en vue de faciliter la transmission efficace des données.

5.3.3 Placement

Dans l'architecture MPSoC, un PE est utilisé comme processeur manager (MP) qui est responsable de la liaison des tâches(task binding), placement des tâches(placement), l'ordonnancement des tâches d'application, routage des communications, re-configuration et contrôle de ressources. *Task binding* est nécessaire avant le placement de tâches dans le cas des MPSoCs hétérogène. Pour chaque tâche, le binding process définit le type des PEs sur les quelles la tâche peut être placé et exécuté. Par exemple, tâches logiciels seront placés et exécutés dur les ISPs, alors que les tâches matérielles sur

les RAs et les IPs. *placement de tâche* identifie la position d'un PE dans l'architecture en vue d'attribuer une tâche. *l'ordonnancement des tâches d'application* définit l'ordre d'exécution des tâches initiales des applications sur les clusters. *routage des communications* définit le mécanisme à utiliser pour le routage de données d'un PE a l'autre. *Contrôle de ressource* est maintenue par le mise à jour de l'état des ressources au moment de l'exécution en vue de fournir au MP des informations précises sur l'occupation des ressources. L'information actualisée des ressources aide le MP a prendre de meilleures décisions de placement au moment de l'exécution, qui est basée sur l'utilisation des liens du Noc et les PEs lors de l'exécution. Les résultats de configurations généraux sont utilisés pour simuler le processus de *contrôle de configuration* [147].

Le MP connais que les tâches initiales des applications. La tâche initiale de chaque application est lancé par le MP et les nouvelles tâches communicantes sont chargées dans la plateforme MPSoC lors de l'exécution a partir de la *mémoire des tâches* lorsque une communication entre eux est nécessaire et ils ne sont pas déjà placé.

Le placement de la tâche est représenté par $mpg(t_i \Rightarrow p_i)$, où la tâche t_i d'une application est placé sur la tuile p_i dans l'architecture MPSoC. Le placement de l'application consiste aux placement de toutes les tâches de l'application sur différents PEs de l'architecture tout en optimisant certains mesures de performance tels que le temps d'exécution, la consommation d'énergie, la latence, la congestion, etc. Le placement multiples d'applications implique le placement de tâches à partir de différentes applications en parallèle tout en optimisant la performance de chacune des applications. Ce travail considère le placement simultanée de plusieurs applications sur l'architecture MPSoC.

Le placement de la tâche initiale a un impact significatif sur la performance du placement dynamique. Les tâches initiales sont considérés comme des tâches logicielles et donc sont placés sur des ressources logicielles (ISPs). Les tâches initiales d'applications sont placés de manière distribuée sur toute l'architecture. Pour cela, l'architecture est divisée en plusieurs clusters répartis et les tâches initiales sont placés au centre des clusters, comme représenté sur la Fig. 5.3. Un tel placement

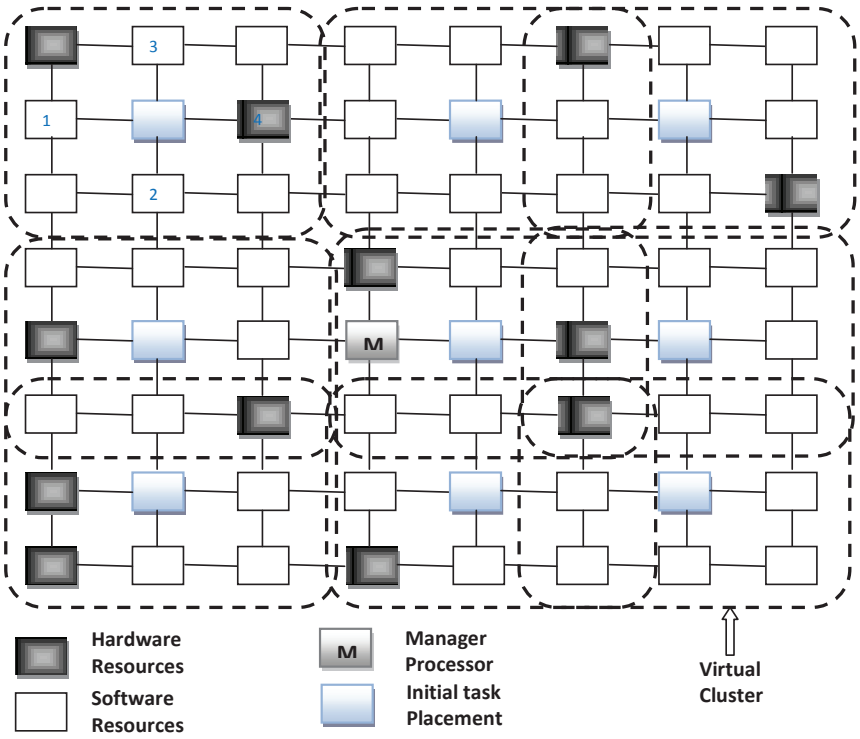


FIGURE 5.3 – Placement des tâches initiales pour le placement de 9 applications simultanément.

des tâches initiales des différentes applications facilite le placement des tâches de chaque application proches les unes des autres à l'intérieur de régions particulière (clusters).

Cela réduit les couts de communications puisque les tâches communicantes de chaque applications sont placés a proximité. Les frontières des clusters sont virtuels et ainsi des régions communes pourraient être partagés par les tâches d'applications différentes. après que les tâches initiales de chaque application sont placés, les demandes de communications sont envoyés aux tâches communicantes afin de trouver leur placement.

5.4 Heuristiques de placement de référence

Nous utiliserons les heuristiques de placement Packing-based Nearest Neighbor (PNN) et Packing-based Best Neighbor (PBN) proposées dans [120] comme des heuristiques de référence à utiliser pour l'étude comparative.

5.4.1 Packing-based Nearest Neighbor (PNN)

Afin de trouver le placement pour une tâche, PNN commence à rechercher un PE libre capable d'exécuter (supporté) la tâche à proximité du PE contenant la tâche maître. L'heuristique PNN a été introduit dans l'algorithme 6. L'algorithme prend en entrée : NoC_limit, requestedTask, position (X, Y) du PE contenant la tâche maître et fournit la position X', Y' de la ressource libre qui peu exécuter la tâche. Premièrement, les ressources les plus proches au PE sur lequel s'exécute la tâche maître sont recherchées pour leurs disponibilités et leurs supports (de même type). Ces PEs voisins sont situés à une distance de saut égale a 1 depuis le PE de la tâche maître de la manière *Gauche*, *Bas*, *Haut*, *Droite*. Si aucun des PEs à la distance hop 1 est disponible pour supporté la tâche alors PEs à distance de saut de 2 sont recherchées. Le processus de recherche se poursuit jusqu'à la limite NoC. Pour rechercher les PEs à un saut particulier (>1), la recherche commence a partir de chaque ressource visitée au saut précédent et la même séquence de recherche du saut 1 est suivie. Le processus de recherche similaire est répété en augmentant la distance de saut jusqu'à ce qu'un PE libre et qui permet d'exécuter la tâche demandée est trouvé. Le processus s'arrête dès qu'un PE libre et qui permet de supporter la tâche est trouvé. Si toutes les ressources dans le NoC_limit ne sont pas capable de supporter la tâche demandée, alors la tâche est ajoutée dans la liste de List_Task_not_mapped et son placement sera recherché dès qu'une ressource PE sera libre. L'algorithme retourne la position de la ressource capable de supporter la tâche demandée (X' = PE.X et Y' = PE.Y).

L'algorithm 7 montre le fonctionnement de la fonction de recherche, qui est suivie pour évaluer le PE. Les tests de la fonction est si la ressource à une position donnée est libre et si elle est en mesure de supporter la tâche demandée. La même fonction est appelée pour évaluer les ressources aux positions *Gauche*, *Bas*, *Haut* et *Droite* par rapport à la position de la tâche maître où elle est placée. La ressource peut supporter la tâche demandée si la ressource est libre (PE[X][Y].isFree) et le type de la ressource est le même que le type de la tâche demandée (requestedTask.type = PE[X][Y].type). La fonction change la valeur de la variable *mapped* à *true* pour

Algorithm 6 PNN Algorithm

```
Require: NoC_limit, requestedTask, X, Y
Ensure: X', Y'
 1: mapped←False ; hop←1 ; i=0 ;
 2: search_Left_Down_Up_Right(requestedTask,X,Y)
 3: while ((mapped=false)and(i<NoC_limit)) do
 4:    i++ ;
 5:    search_Left_Down_Up_Right(requestedTask,X,Y-i)
 6:    if (mapped=false) then
 7:       search_Left_Down_Up_Right(requestedTask,X+i,Y)
 8:    end if
 9:    if (mapped=false) then
10:       search_Left_Down_Up_Right(requestedTask,X-i,Y)
11:    end if
12:    if (mapped=false) then
13:       search_Left_Down_Up_Right(requestedTask,X,Y+i)
14:    end if
15: end while
16: if (mapped=false) then
17:    List_Task_not_mapped←requestedTask
18: else
19:    X'←PE.X ;
20:    Y'←PE.Y ;
21: end if
```

indiquer que la ressource est capable de supporter la tâche demandée et la position de la ressource est marquée comme X′ et Y′. Si la ressource n'est pas en mesure de supporter la tâche alors l'algorithme 6 continue avec les PEs restants dans le même ordre de sequence.

Par conséquent, le processus de recherche évalue tous les voisins à une distance de n-hop, où n varie de 1 à NoC_limit. Dans le cas où un PE libre capable de supporter la tâche demandée n'est pas trouvé à des sauts antérieures. La disponibilité et le support doivent être vérifiés comme un PE pourrait être libre mais ne peut pas supporter la tâche si les deux ne sont pas du même type (par exemple, matériel). Le NoC_limit est la distance de saut maximale jusqu'à laquelle le réseau peut être analysée pour trouver un placement. La distance du hop maximale est la distance de Manhattan à partir du point de départ de la recherche jusqu'au coin le plus éloigné de la frontière NoC. L'heuristique se termine dès qu'un PE libre et qui permet de supporter la tâche est trouvé. A chaque saut (hop), les PEs sont recherchés avec la séquence gauche, bas, haut et droite. Par exemple, la séquence de recherche est montré par 1 : gauche, 2 : bas, 3 : haut, et 4 : droite, sur la Fig. 5.3. La même stratégie est suivie pour placer toutes les tâches demandées pour chaque application.

Algorithm 7 Search Algorithm

Require: *requestedTask, NoC_limit, X, Y*
Ensure: *X', Y'*
1: **if** $((0 \leq X \leq NoC\_limit) \&\& (0 \leq Y \leq NoC\_limit))$ **then**
2: **if** ((PE[X][Y].isFree)&&(requestedTask.type = PE[X][Y].type)) **then**
3: mapped←true ;
4: *X'*←*PE.X* ;
5: *Y'*←*PE.Y* ;
6: **end if**
7: **end if**

5.4.2 Packing-based Best Neighbor (PBN)

L'heuristique PNN ne considère que la proximité d'une ressource disponible pour exécuter une tâche donnée. L'heuristique PBN combine la stratégie de recherche de PNN avec l'approche de calcul de la charge du chemin (PL). Contrairement à PNN qui s'arrête une fois qu'une ressource libre et capable de supporté une tâche est trouvée, PBN évalue tous les PEs qui sont libres et capables de supporter une tâche donnée sur une même distance. Pour tous les PEs libres et qui peuvent supporter une tâche donnée leurs charges de chemin est calculées et le PE avec le PL minimum est choisie pour le placement finale afin d'obtenir le meilleur voisin des voisins disponibles. Le lien qui doit transférer un nombre minimum de paquets est celui qui a le PL minimum. Si le placement est trouvée avec les PEs dans la distance hop actuel, le processus d'évaluation est arrêté pour d'autres distances hop plus élevées. Les autres étapes de l'heuristique PBN sont semblables à celles de l'heuristique PNN.

5.5 Stratégies de packing et de routage proposées

Cette section décrit nos approches de placement dynamique proposées. L'approche proposée cherche à minimiser le temps de recherche pour trouver une ressource libre en mesure d'exécuter une tâche donnée afin d'optimiser le processus de placement de tâche. La stratégie de packing Manhattan proposée réduit le temps d'exécution globale et la consommation d'énergie pour le placement dynamique des tâches de manière significative. Les communications entre les tâches sont établies en utilisant un nouveau algorithme proposé de routage multi-objectif pour une communication efficace en vue de réduire les coûts de communication en termes de temps et d'énergie de communication. Les tâches communicantes qui sont placées

sur deux PEs différents dans l'architecture MPSoC ont besoin de communiquer à travers le NoC, ce qui pourrait nécessiter une énergie considérable. Cela nous a motivé à proposer un algorithme de routage multi-objectifs afin de réduire le coût de la communication.

5.5.1 Manhattan Packing Strategies

Avant de décrire les stratégies de packing Manhattan proposées, nous montrons un exemple de placement par la stratégie PNN pour mettre en évidence ses limites. PNN a été considérée comme l'une des heuristiques de placement de référence.

Pour placer une tâche demandée, d'abord, la tâche est tentée d'être placée sur les PEs autour du PE sur lequel s'exécute la tâche maître à une distance de saut égale à 1. Les PEs sont recherchés dans la séquence de *gauche*, *bas*, *haut* et *droite*, notée 1, 2, 3 et 4, respectivement en Fig. 5.4 (a). La même stratégie est suivie du saut 1 jusqu'à ce qu'un PE libre et qui peut supporter la tâche est trouvé. Chaque application suit la même stratégie pour le placement des tâches demandées sur les ressources de la plateforme MPSoC. Le reste des tâches est placé comme représenté sur la Fig. 5.4 (a). Les limitations de l'approche PNN sont observées lorsque le nombre des tâches dans les applications considérées est très important. La plupart des travaux existants et références ne considèrent pas les applications à grand nombre de tâches. Dans le cas d'un grand nombre de tâches dans l'application, la PNN n'est pas très performante en termes de temps de placement. En outre, pour la position appropriée de la ressource matérielle dans la plateforme, PNN encourt un temps de placement élevé car il doit chercher pour 20 PEs avant d'atteindre le PE matériel, comme représenté sur la Fig. 5.4 (a). La séquence de recherche est mentionnée comme étant de 1 à 20. L'approche proposée permet de réduire le temps de placement pour les applications à grand nombre de tâches.

Manhattan Packing-based Nearest Neighbor (MPNN)

Pour placer une tâche demandée, la stratégie MPNN tente d'abord de trouver un placement sur une distance de saut de 1 d'une manière similaire à celle de PNN.

La séquence de recherche pour les PEs est 1, 2, 3 et 4 comme montré dans la Fig. 5.4 (b). Pour le deuxième saut, les PEs sont recherchés dans la séquence *gauche*, *gauche-bas*, *bas*, *droite-bas*, *droite*, *haut-droite*, *haut* and *haut-gauche*, désignée par les différents numéros à une distance PEs de deux sauts à partir de 5. La même stratégie de recherche est utilisée en augmentant la distance du saut. Pour placer une tâche matérielle demandée, La stratégie PNN a besoin d'évaluer 20 PEs(nombre de recherche) jusqu'à ce qu'un PE matériel libre est trouvé, comme montré dans la Fig 5.4 (a). En revanche, notre stratégie MPNN évalue seulement 9 PEs(nombre de recherche).

L'heuristique MPNN a été introduite dans l'algorithme 8. Pour le placement d'une tâche demandée, l'algorithme prend en entrée *requestedTask*, les coordonnées X & Y du PE qui exécute la tâche maître et la taille de la plateforme (NoC_limit) et fournit en sortie les coordonnées (X′ & Y′) du PE qui est en mesure d'exécuter la tâche. Une variable *mapped* est utilisée pour tester si la tâche a été placée ou pas, et elle est initialement affecté à faux. Hop (distance ou pas) est utilisé pour déterminer la distance en nombre de saut (allant de 1 jusqu'à NoC_limit) entre le PE qui exécute la tâche maître et le PE qui exécute la tâche demandées. Le placement de la tâche est recherché a partir de la distance hop 1 jusqu'à NoC_limit et il est arrêté dès qu'un PE libre et en mesure de supporter la tâche est trouvé.

Pour le hop=1, le MPNN évalues les PEs en suivant la sequence *gauche*, *bas*, *droite* and *haut*. Pour le hop supérieur ou égal à 2, la séquence d'évaluation est *gauche*, *gauche-bas*, *bas*, *droite-bas*, *droite*, *haut-droite*, *haut* et *haut-gauche*, comme décrit dans l'algorithme. Pour chaque évaluation, le MPNN fait appel a la fonction de recherche présentée dans l'algorithme 7, à savoir, recherche similaire à celle appliqué par PNN. Cette fonction teste si la position du PE est dans le NoC_limit et si il peux supporter la tâche. La fonction retourne la position du PE si 1) le PE est dans le NoC_limit, 2) le type de la tâche est le même que celui du PE et 3) le PE est libre. En outre, la fonction attribue la variable *mapped* à *true* pour indiquer qu'une ressource libre et en mesure d'exécuter la tâche est trouvée.

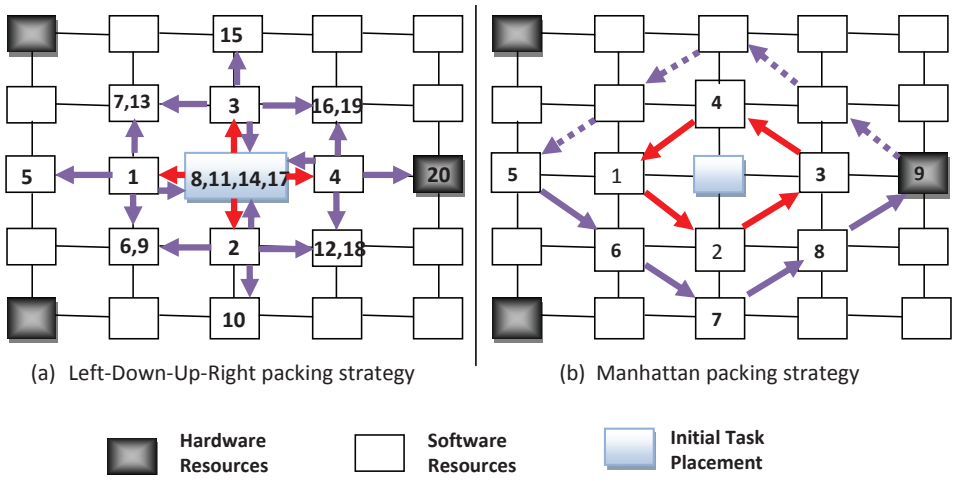


FIGURE 5.4 – placement par les stratégies MPNN et PNN.

Manhattan Packing-based Best Neighbor (MPBN)

L'espace de recherche de la stratégie MPBN est similaire à MPNN. Contrairement à MPNN qui place la tâche sur le premier PE libre et qui permet de supporter la tâche, le MPBN évalue tous les PEs sur une même distance hop et sélectionne celui qui a la charge du lien minimum. Si un PE libre et qui supporte la tâche est trouvé sur la distance hop courante, après évaluations pour les autres distances hop est annulé et l'algorithme se termine.

5.5.2 Multi-Objective Routing Algorithm (MORA)

La plus part des heuristiques de référence qui traitent les placements dynamiques (par exemple, [120, 135, 108]) utilisent l'algorithme de routage déterministe XY pour faciliter les communications entre les tâches communicantes une fois qu'elles sont placés sur les PEs. Exemple d'un tel routage est montré dans la Fig. 5.5 (a). La figure montre un exemple d'un NoC où deux tâches communicantes sont placées sur deux PEs source et destination, elles ont besoin de communiquer entre elles. Les valeurs mentionnées prés des liens représente le volume de données présent dans le lien , c.à.d le nombre de paquets à transmettre sur le lien. Fig. 5.5 (a) montre que pour transférer un paquet d'un PE source à un PE destination, le paquet est transféré d'abord sur une distance de deux hop dans la direction X, puis sur une distance de deux hop dans la direction Y en suivant le mécanisme de routage XY.

Algorithm 8 MPNN Heuristic

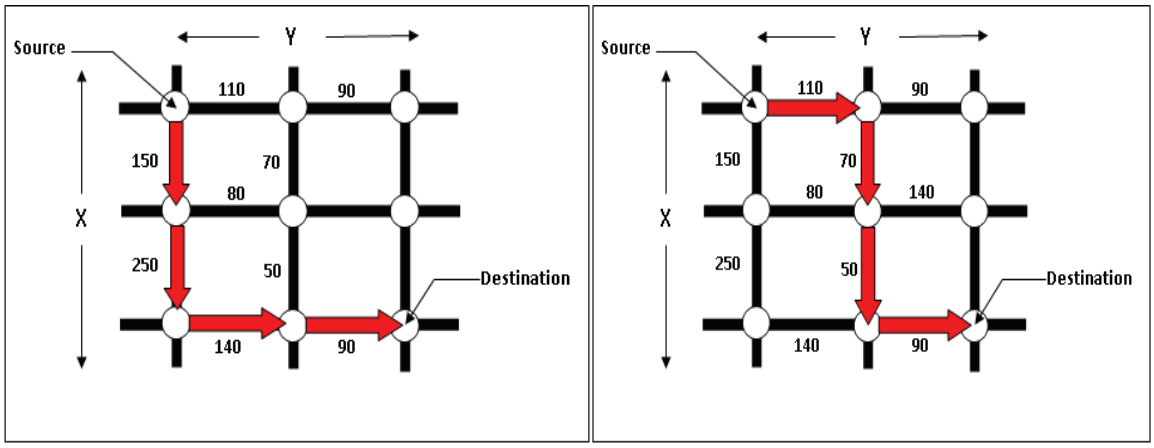
```
Require: NoC_limit, requestedTask, X, Y
Ensure: X', Y'
 1: mapped←False ; hop←1 ;
 2: while ((hop < NoC_limit)&&(mapped=false)) do
 3:    search(requestedTask, X, Y-hop) {search Left}
 4:    for (i = 1 ; i < hop ; i++) do
 5:       if (mapped=false) then
 6:          search(requestedTask, X+i, Y-(hop-i)){search Left Down}
 7:       end if
 8:    end for
 9:    if (mapped=false) then
10:       search(requestedTask, X+hop, Y){search Down}
11:    end if
12:    for (i = 1 ; i < hop ; i++) do
13:       if (mapped =false) then
14:          search(requestedTask, X+(hop-i), Y+i){search Right Down}
15:       end if
16:    end for
17:    if (mapped=false) then
18:       search(requestedTask, X, Y+hop){search Right}
19:    end if
20:    for (i = 1 ; i < hop ; i++) do
21:       if (mapped=false) then
22:          search(requestedTask, X-i, Y+(hop-i)){search Top Right}
23:       end if
24:    end for
25:    if (mapped=false) then
26:       search(requestedTask, X-hop, Y){search Top}
27:    end if
28:    for (i = 1 ; i < hop ; i++) do
29:       if (mapped=false) then
30:          search(requestedTask, X-(hop-i), Y-i){search Top Left}
31:       end if
32:    end for
33:    hop++ ;
34: end while
```

Les paquets sont envoyés un par un sur le même chemin crée par le premier paquet. Dans la Fig. 5.5 (a), le premier lien choisi dans la direction de X a un volume de (150) qui est plus important que celui du volume présent dans la direction Y qui est de (110). De même, le deuxième lien choisis dans la direction de X a un plus grand volume que celui présent dans la direction de Y (250 vs. 80). Ce mécanisme route les paquets à travers un chemin qui entraîne des coûts de communication élevés dus à des volumes élevés présents dans les liens choisis pour la communication. Ce qui entraîne des coûts élevés de communication. Ainsi, le choix de ces voies de communication peut entraîner des temps de communication et de consommation d'énergie plus élevés.

Afin de fournir une communication efficace entre les PEs source et destination, une stratégie de routage efficace doit être développée. La stratégie de routage doit être en mesure de choisir les liens qui disposent de la moindre charge au moment de l'exécution. Fig. 5.5 (b) décrit un exemple pour le fonctionnement de l'algorithme

de routage dynamique proposé, présenté dans l'algorithme 9. Contrairement à l'algorithme XY, le MORA choisit un chemin de routage efficace où les paquets sont transférés par les liens ayant les charges les plus faibles. La direction à prendre à partir du PE source à PE de destination suit des chemins différents en fonction de l'emplacement des PEs et des charges des liens. Si la coordonnée X du PE source (X_{source}) est inférieure à la coordonnée x du PE de destination, alors la trajectoire (path) sera du *haut* en *bas*, sinon du *bas* en *haut*. Pour le *bas* en *haut*, si les coordonnée Y du PE source (Y_{source}) est inférieure à la coordonnée Y du PE de destination (Y_{dest}), alors le chemin sera du *gauche* à *droite*, Sinon de la *droite* à la *gauche*. Pour tous les différents liens, l'algorithme choisit la direction du lien qui a la moindre charge. Par exemple, l' Algorithme 10 montre comment le lien qui dispose de la moindre charge est trouvé dans le cas de *haut_ en_ bas-gauche_ à_ droite*. En fonction des valeurs de charges des liens, l'algorithme choisit la direction du lien de *gauche* à *droite* (X'=X_{source},Y'=$Y_{source}+1$) ou du *haut* en *down*, qui a la moindre charge. Une approche similaire à celle de l'algorithme 10 est suivie pour les autres cas où *haut*, *bas*, *gauche* et *droite* sont contenues dans la fonction appelante. Dans le cas ou Y_{source} et Y_{dest} sont les mêmes c'est à dire dans la même colonne, la direction est de *haut* en *bas* ou de *bas* en *haut* et il n'y a pas d'évaluation pour l'obtention de la charge du lien. La direction est automatiquement pris dans l'une des deux directions. De même, si X_{source} et X_{dest} sont les mêmes, C'est à dire dans la même ligne alors la direction du lien choisi est de *gauche* à *droite* ou de *droite* à *gauche*. Ce genre de sélection de liens vers le PE de destination facilite de choisir les liens les moins chargés. Une fois un lien choisi devient plus chargé, un autre lien moins chargé est choisi pour la transmission de paquet si le PE source et le PE de destination ne sont pas dans la même ligne ou la même colonne. Sinon, le même lien est utilisé. Pour toutes les tâches communicantes, les paquets à transférer utilisent la même stratégie.

Toutes les fonctions ont le même principe.



(a) XY Routing (b) MORA Routing

FIGURE 5.5 – XY et MORA.

Algorithm 9 Multi-Objective Routing Algorithm

Require: X_{source}, Y_{source}, X_{dest}, Y_{dest}
Ensure: *X'*, *Y'*
{Up to Down}
if $X_{source} < X_{dest}$ **then**
  if $Y_{source} < Y_{dest}$ **then**
    Up_to_Down-Left_to_Right(X_{source}, Y_{source})
  else
    Up_to_Down-Right_to_Left(X_{source}, Y_{source})
  end if
else
  {Down to Up}
  if $Y_{source} < Y_{dest}$ **then**
    Down_to_Up-Left_to_Right(X_{source}, Y_{source})
  else
    Down_to_Up-Right_to_Left(X_{source}, Y_{source})
  end if
end if
if X_{source}=X_{dest} **then**
  Right_to_Left-Left_to_Right(X_{source}, Y_{source}){in the same row}
end if
if Y_{source}=Y_{dest} **then**
  Up_to_Down-Down_to_Up(X_{source}, Y_{source}){in the same column}
end if

5.5.3 Calcul du Temps d'exécution globale et la consommation d'énergie

Modélisation et Formulation mathématique

$T_{exe}^{ti(s/h)}$: temps d'exécution de la tâche t_i sur la ressource (la tâche logicielle sur la ressource logicielle et la tâche matérielle sur la ressource matérielle).

Q_j^i : le volume de données échangés entre t_i and t_j.

R : débit (nombre de paquets envoyés sur un seul envoi).

T_{map}^{ti} : le temps de placement de la tâche t_i.

$T_{map}^{com_{master}^{ti}}$: temps de placement de communication de la tâche maître à la tâche

Algorithm 10 Up_to_Down-Left_to_Right

Require: X_{source}, Y_{source}
Ensure: X', Y'
1: **if** *get_value_ Link*(X_{source},$Y_{source+1}$) < *get_value_ Link*($X_{source+1}$,Y_{source}) **then**
2: $X' \leftarrow X_{source}$
3: $Y' \leftarrow Y_{source+1}$ {Left to Right}
4: **else**
5: $X' \leftarrow X_{source+1}$ {Up to Down}
6: $Y' \leftarrow Y_{source}$
7: **end if**

esclave.

$T_{map}^{com_{ti}^{master}}$: temps de placement de communication de la tâche esclave à la tâche maître.

T_{upload}^{ti} : temps de chargement de la tâche (configuration) t_i.

EC_{ti}^{s} : consommation d'énergie logicielle.

EC_{ti}^{h} : consommation d'énergie matérielle.

EC_{s}^{R} : consommation d'énergie d'un seul paquet avec le débit R.

EC_{w}^{d} : consommation d'énergie pour l'attente de données d.

EC_{inst}^{s} : consommation d'énergie d'une instruction sur une ressource logicielle.

EC_{inst}^{h} : consommation d'énergie d'une instruction sur une ressource matérielle.

$EC_{s_{ti}^{tj}}^{R}$: consommation d'énergie d'un paquet envoyé de t_i vers t_j avec un débit R.

EC_{w} : consommation d'énergie pour l'attente de données sur le lien pour l'envoi de données de t_i vers t_j.

EC_{Total} : consommation d'énergie totale.

T_{exe}^{app} : temps d'exécution d'application.

$T_{Totalexec}$: temps d'exécution totale.

Calcul de la consommation d'énergie : La façon de calcul des différentes valeurs de consommation d'énergie sont introduits par la suite.

La consommation d'énergie pour la tâche logiciel :

$$EC_{t_i}^{s} = EC_{inst}^{s} * T_{st}^{inst} \tag{5.1}$$

La consommation d'énergie pour la tâche du matériel :

$$EC_{t_i}^{h} = EC_{inst}^{h} * T_{ht}^{inst} \tag{5.2}$$

La consommation d'énergie pour l'envoi d'un paquet de t_i à t_j avec un débit R :

$$EC^{R}_{s^{t_j}_{t_i}} = \quad EC^{R}_{s} * \frac{Q^{i}_{j}}{R} \tag{5.3}$$

La consommation d'énergie pour l'attente de données dur le lien lors de l'envoi de données de t_i à t_j :

$$EC_{w} = EC^{d}_{w} * \frac{Q^{i}_{j}}{R} \tag{5.4}$$

Enfin, la consommation d'énergie totale est calculée comme suit :

$$EC_{Total} = \sum EC_{w} + \sum EC^{R}_{s^{t_j}_{t_i}} + \sum EC^{s}_{t_i} + \sum EC^{h}_{t_i} \tag{5.5}$$

Calcul du temps d'exécution : Le temps d'exécution d'une tâche t_i (software ou hardware), $T^{t_i}_{exe}$, est la somme des temps nécessaires pour trouver un placement pour la tâche ($T^{t_i}_{map}$) et ses communications , temps de configuration pour la tâche t_i ($T^{t_i}_{upload}$) et le temps d'exécution de la tâche ($T^{t_i(s/h)}_{exe}$) dans la ressource configurée (ressource logicielle ou matérielle). Le temps de placement des communications se compose du temps pris pour le placement des communications de la tâche maître t_i à la tâche esclave t_j et de la tâche esclave t_j à la tâche maître t_i. La tâche maître t_i ne termine pas jusqu'à ce que toutes ces tâches esclaves soient terminées. Comme les esclaves s'exécutent en parallèle, celle qui prend le temps maximum contribue au temps d'exécution de la tâche maître t_i. Le temps d'exécution de l'application se compose du temps des communications en plus de ceux mentionnés ci-dessus.
Le temps d'exécution pour chaque tâche est calculé de façon récursive de la façon suivante :

$$T^{t_i}_{exe} = T^{t_i}_{upload} + T^{t_i(s/h)}_{exe} + T^{t_i}_{map} + T^{com^{t_i}_{master}}_{map} + T^{com^{master}_{t_i}}_{map} + \sum_{n=1}^{slaves-number} Max\ T^{tn}_{exe} \tag{5.6}$$

Le temps d'exécution global est calculé comme le temps d'exécution maximum parmi toutes les applications en cours d'exécution en parallèle.

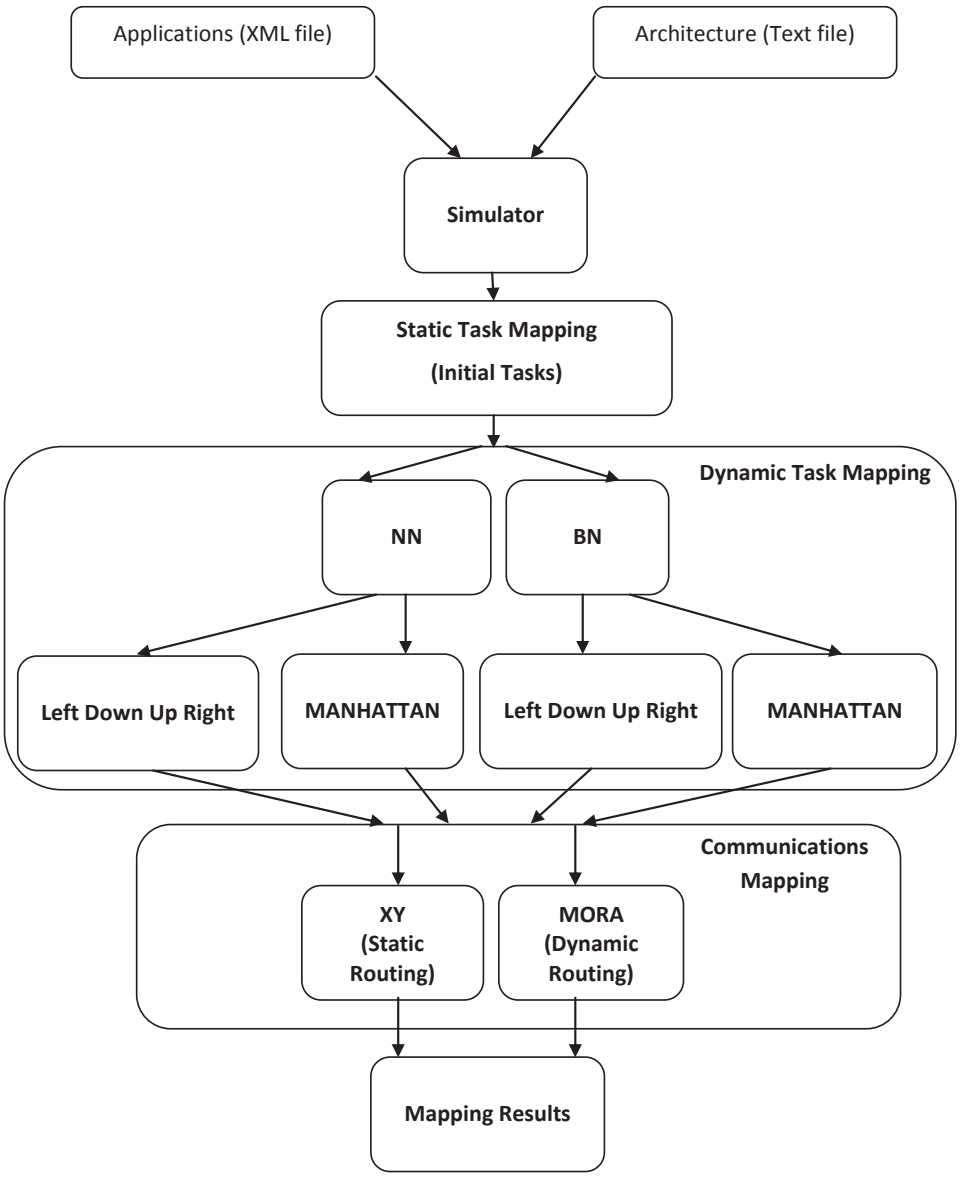


FIGURE 5.6 – Flux d'exécution de la simulation.

$$T_{TotalExec} = \max_{i=0 \ to \ appl-numb} T_{exe}^{app} \tag{5.7}$$

5.6 Validation par Simulation

Pour comparer les heuristiques de placement, nous avons développé un simulateur de haut niveau écrit en Java. Fig. 5.6 montre le flux pour l'exécution de la simulation.

5.6.1 Configuration de Simulations

Cette section décrit la configuration expérimentale utilisée. Toutes les applications sont modélisées comme illustré sur la Fig. 5.1 (a), avec des tâches initiales, matérielles et logicielles. Les valeurs présentes sur les liens correspondent au volume des données qui doivent être envoyées et reçues par le maître comme il est expliqué dans la définition 5.3.1. Le NoC est modélisé comme dans la Fig. 5.3 avec les tâches initiales placées au milieu de chaque cluster. Nous avons réalisé une plateforme hétérogène qui contient 64 éléments de traitement (PEs) : 12 hardware, 51 software, et un processeur Manager. Le Manager est responsable de trouver le placement des tâches d'applications, la liaison des tâches (task binding),la configuration des tâches(upload), la mise à jour des ressources de la plateforme et le routage des communications. La plateforme utilise un réseau sur puce (Network-on-Chip (NoC)) en tant que support de communication, qui est responsable de la transmission des données entre les tâches. Le processeur Manager ne connait que les tâches initiales. Lorsque les tâches initiales commencent leurs exécutions, les tâches esclaves sont placées dynamiquement, en fonction des demandes de communications. Le temps d'exécution des tâches dépend du type et de la capacité du processeur (PE). Nous pouvons varier plusieurs paramètres via un fichier de configuration d'entrée (fichier de paramètres) qui contient tous les paramètres tels que la configuration de la plateforme, le choix des heuristiques de placement dynamique, la méthode de routage, etc. Les simulations sont effectuées sur différents scénarios :

— **Scenario 1 :** Applications générées par l'outil *Task Graph For Free* (TGFF [75]) comme illustré dans la Fig. 5.7 (3-4 Niveau, 1-3 fils). Applications contiennent un maximum de 9 tâches.
— **Scenario 2 :** Applications *Multi-Window Display* (MWD), *Video Object Plane Decoder* (VOPD), *Pecture-In Picture* (PIP) comme illustré dans la Fig. 5.8. Les applications MWD, VOPD et PIP contiennent 12, 15 et 8 tâches, respectivement.
— **Scenario 3 :** Plusieurs applications MPEG-4. L'application contient 13 tâches

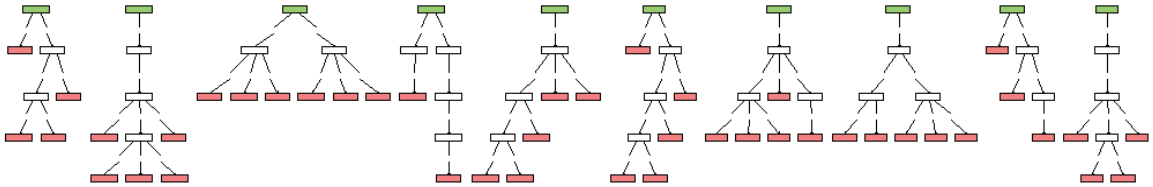


FIGURE 5.7 – Applications générer par l'outil TGFF (3-4 Niveau, 1-3 fils).

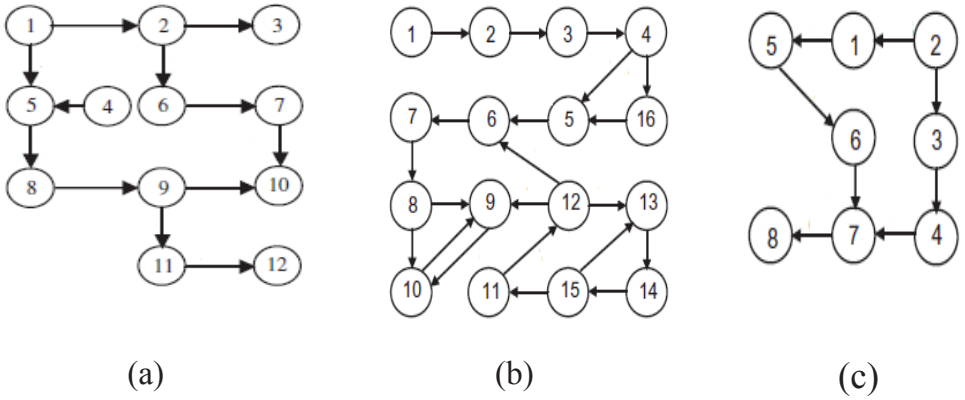


FIGURE 5.8 – Applications (a) MWD, (b) VOPD, (c) PIP.

(Fig. 5.9).

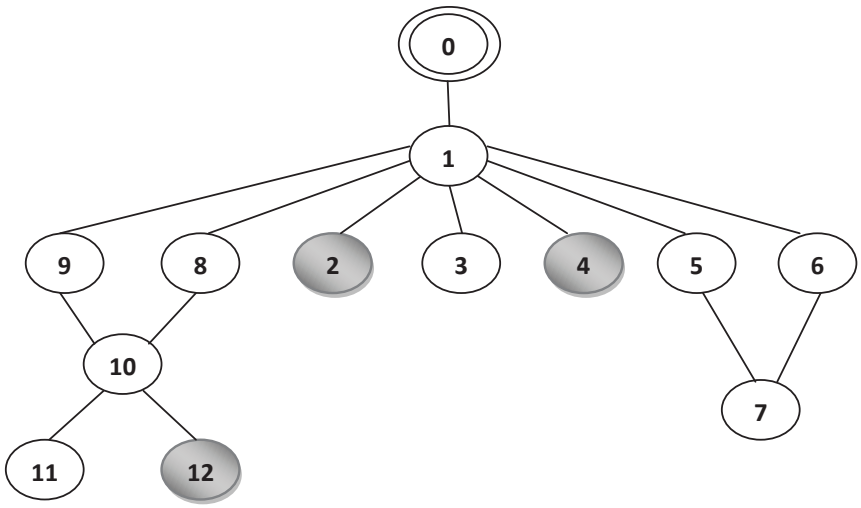


FIGURE 5.9 – Application MPEG-4.

— **Scenario 4 :** quatre séries d'applications : 1^{er} série - chaque application ayant 5 tâches, 2^{eme} série - chaque application ayant 10 tâches, 3^{eme} série - chaque application ayant 15 tâches, et 4^{eme} série - chaque application ayant 20 tâches.

Pour chaque scénario, nous essayons de placer et d'exécuter un total de dix (10) applications, alors que n'importe quel nombre d'applications peut être envisagés.

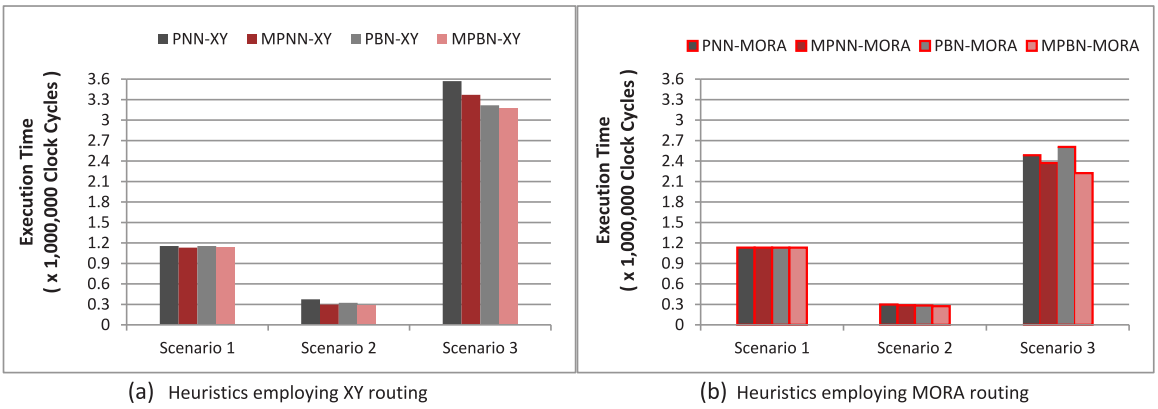


FIGURE 5.10 – Comparaison du temps d'exécution de PNN et PBN avec MPNN et MPBN respectivement lors de l'utilisation du routage XY et MORA pour trois scénarios.

La plateforme est divisée en neuf (9) clusters et donc neuf applications peuvent être placées et exécutées initialement. Toute application supplémentaire doit attendre son tour d'exécution dans une fille d'attente tant que l'un des neuf premières application n'a pas fini. Plusieurs instances de la même application sont considérés pour prendre un total de dix (10) applications dans chaque scénario. Le volume de données dans les différents scénarios a été varié. Les applications avec un nombre variable de tâches sont considérées pour voir la distance en termes de nombre de sauts sur le quel les tâches de la même application peuvent êtres placés. Pour une application à grande taille, les tâches sur les feuilles du graphe d'application peuvent être placées loin de la tâche initiale. Le temps de placement pour une tâche augmente avec le nombre de sauts puisque un grand nombre de PEs a besoin d'être évalué. Il faut pour cela une technique efficace qui peut réduire le temps de placement. En outre, il faut disposer d'une méthode de routage adaptative de façon à minimiser les coûts des communications. Dans le travail actuel, les ressources matérielles et logicielles sont utilisées en mode Mono-tâche c.à.d que chaque ressource n'exécute qu'une seule tâche. Notre plateforme permet aussi d'exécuter le multi-tâches mais leurs résultats seront discutés lors de nos future publications.

5.6.2 Résultats de Simulations

Les résultats obtenus pour nos heuristiques Manhattan proposées MPNN et MPBN sont comparées avec les heuristiques de placement dynamique existantes PNN et PBN. Nos heuristiques montrent un temps de placement (rcherche) inférieure. L'approche de routage proposée MORA a été utilisée avec les heuristiques de placement existantes et proposées afin d'optimiser les coûts de communications. L'adaptation de MORA dans les heuristiques de placement montre des améliorations de performance.

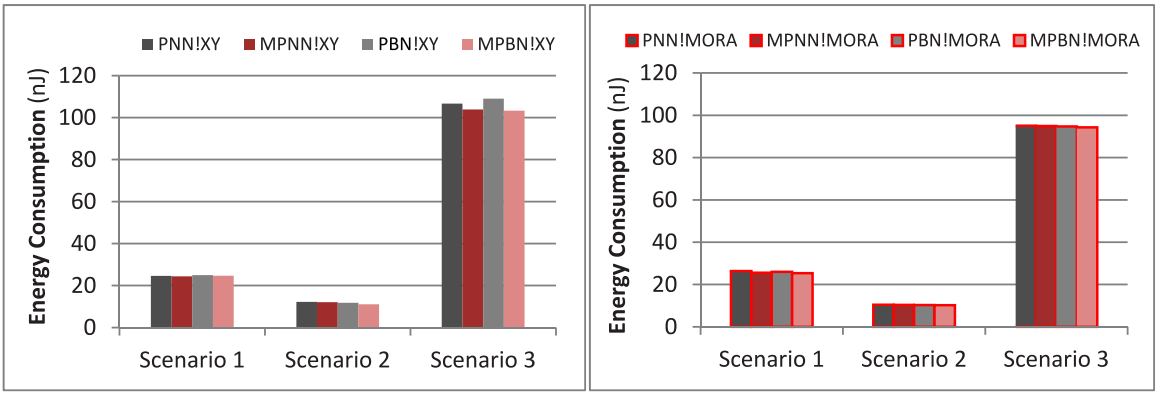
Temps d'exécution total

Temps d'exécution total comprend le temps de placement (le temps nécessaire pour trouver un placement), temps de configuration, temps de communication, temps de calcul et temps d'attente lorsque aucune ressource libre n'est disponible sur la plateforme. L'adaptation de la stratégie de packing dans le processus de placement facilite le placement des tâches communicantes à proximité et tout en réduisant le temps de communication. Il a été également observé que le temps de mapping contribuant au temps d'exécution total se réduit lorsque l'on utilise les heuristiques Manhattan parce que l'espace de recherche pour trouver le placement d'une tâche est minimisé. L'utilisation d'un routage déterministe comme XY peut influer sur les coûts de communications. En utilisant le routage dynamique MORA proposé, nous pouvons réduire considérablement le temps de communication. Dans MORA, les paquets peuvent prendre plus d'une trajectoire (chemin) pour faciliter une communication plus rapide, résultant en une réduction des coûts de communication.

Les graphes de la Fig. 5.10 montrent le temps d'exécution totale pris pour le placement des 10 applications considérées pour les différents scénarios où différentes heuristiques sont appliquées. Fig. 5.10 (a) et (b) montre les temps d'exécution lorsque les heuristiques utilisent les approches de routage XY et MORA, respectivement. Un couple d'observations peuvent être faits à partir des figures. Tout d'abord, les approches proposées MPNN et MPBN réduisent le temps d'exécution par rapport à PNN et PBN, respectivement. Ceci peut être observé lorsque les deux approches de

routage XY et MORA sont utilisées. Deuxièmement, les approches existantes PNN et PBN ont un temps d'exécution qui est réduit lors de l'utilisation de l'approche de routage MORA par rapport à l'utilisation du routage XY. Cela est dû à l'adaptation dynamique des trajets par MORA. La différence des temps d'exécution peut être observée en regardant la Fig. 5.10 (a) et (b).Les résultats similaires peuvent être observés pour les heuristiques proposées MPNN et MPBN lorsque l'approche de routage est modifiée de XY à MORA. Ces observations montrent que nos approches réduisent le temps d'exécution par rapport aux approches existantes, même si l'approche de routage existante XY est employée. Une réduction supplémentaire est obtenue lorsque l'approche de routage proposé MORA est utilisée.

Dans le scénario 1, nos approches proposées MPNN et MPBN employant MORA réduisent le temps d'exécution total par 1% et 2% par raport à PNN et PBN employant le routage MORA. Pour le scénario 1, la réduction du temps d'exécution n'est pas importante puisque le nombre de tâches dans les applications sont est inférieur à dix (10) et il ya au maximum trois esclaves par maître, ce qui limite les avantages de nos approches. Dans le scénario 2, nos approches proposées MPNN et MPBN employant MORA réduisent le temps d'exécution total par 4% and 5% par raport à PNN et PBN employant le routage MORA. La réduction meilleur que celle obtenue avec le scénario 1, puisque le nombre de tâches dans les applications envisagées a été augmenté mais le nombre d'esclaves par maître reste constant. Dans le scénario 3, Nos approches proposées MPNN et MPBN employant MORA réduisent le temps d'exécution total par 6% et 15% par raport à PNN et PBN employant le routage MORA. La réduction est plus grande que le scénarios 1 et 2 puisque l'application MPEG-4 contient 13 tâches et une des tâches maître contiennent sept (7) tâches esclaves. Les approches proposées fourniront de meilleurs résultats lorsque les applications contiennent un grand nombre de tâches tout en ayant un grand nombre de tâches esclave par maître.



(a) Heuristics employing XY routing

(b) Heuristics employing MORA routing

FIGURE 5.11 – Comparaison de la consommation d'énergie de PNN et PBN avec MPNN et MPBN respectivement lors de l'utilisation du routage XY et MORA pour trois scénarios.

Consommation d'énergie

Une consommation d'énergie est nécessaire lorsqu'un paquet doit être transmis d'un PE source à un PE destination, puis le traitement du paquet par le PE de destination une fois qu'il est reçu. Les énergies nécessaires à la transmission et au traitement sont normalement dénommées énergie de communication et de calcul, respectivement.

L'énergie de communication dépend du nombre de paquets à transmettre, le nombre de liens a parcourir entre le PE source et PE de destination et l'énergie nécessaires pour la transmission d'un paquet par le lien. L'énergie de la communication est calculée en multipliant le nombre de paquets à transmettre et la consommation d'énergie de chaque paquet à transmettre sur les liens utilisées. La consommation d'énergie totale inclus l'énergie d'attente des données dur les liens, l'énergie nécessaire pour la recherche d'une ressource libre et l'énergie nécessaire pour le calcul (tâches hard et soft).

Les graphes dans Fig. 5.11 montre la consommation d'énergie pour l'exécution de dix (10) applications considérées sur différents scénarios lorsque différentes heuristiques sont employées. Fig. 5.11 (a) et (b) montre la consommation d'énergie lorsque les heuristiques emploient les approches de routages XY et MORA, respectivement. Les résultats montrent que la consommation d'énergie totale par les heuristiques MPNN et MPBN employant le routage MORA est réduit significativement compa-

rée à PNN et PBN, respectivement. Pour le scénario 1, nos approches proposées MPNN et MPBN réduisent la consommation d'énergie de 3 % et 4 % par rapport à PNN et PBN utilisant le routage MORA. Pour le scénario 2, nos approches proposées MPNN et MPBN réduisent la consommation d'énergie par 2 % et 3 % par rapport à PNN et PBN utilisant le routage MORA, considérant que l'amélioration de 2 % a été obtenu pour le scénario 3.

Manhattan vs Existant

Table 5.1 résume en pourcentage la réduction obtenu par l'utilisation des stratégies de placement proposées MPNN et MPBN comparer à PNN et PBN respectivement lors de l'utilisation des approches de routage XY et MORA pour différents scénarios. On peut observer que les stratégies proposées surpassent les stratégies existantes.

	Scenarios	XY Routing		MORA Routing	
		MPNN over PNN	MPBN over PBN	MPNN over PNN	MPBN over PBN
Reduction (%) in Execution Time	Scenario 1	4%	4%	1%	2%
	Scenario 2	20%	13%	4%	5%
	Scenario 3	6%	2%	6%	15%
Reduction (%) in Energy Consumption	Scenario 1	2%	3%	3%	4%
	Scenario 2	2%	7%	2%	3%
	Scenario 3	3%	2%	2%	2%

TABLE 5.1 – Réduction en (%) du temps d'exécution et consommation d'énergie avec l'utilisation des stratégies de placement proposées utilisant XY et MORA comparées à PNN et PBN pour trois scénarios.

MORA vs XY

Table 5.2 montre la réduction du temps d'exécution et la consommation d'énergie lors de l'utilisation des différentes heuristiques avec le routage MORA par rapport au routage XY. L'approche proposée MORA montre des réductions plus importantes avec l'augmentation du nombre de tâches dans les applications considérées (Scénario

	Scenarios	PNN	PBN	MPNN	MPBN
		MORA over XY	MORA over XY	MORA over XY	MORA over XY
Reduction (%) in Execution Time	Scenario 1	3%	0%	4%	1%
	Scenario 2	20%	5%	13%	4%
	Scenario 3	30%	29%	19%	31%
Reduction (%) in Energy Consumption	Scenario 1	0%	0%	0%	0%
	Scenario 2	15%	14%	12%	8%
	Scenario 3	12%	9%	14%	13%

TABLE 5.2 – Déduction du temps d'exécution et la consommation d'énergie par MORA sur XY lors de l'utilisation des heuristiques PNN, PBN, MPNN et MPBN pour trois scénarios.

1 à Scénario 3).

L'évaluation des performances pour des applications à grande taille

Nous avons évalué la performance pour des applications à grande taille considérées dans le scénario 4. Quatre séries d'applications sont utilisées, chaque série contiennent dix (10) applications avec 5, 10, 15 et 20 tâches. Figure 5.12 montre les temps d'exécution des quatre séries d'applications pris en compte dans le scénario 4. Un couple d'observations peut être fait à partir de la figure 5.12. Tout d'abord, le temps d'exécution se réduit quand le routage MORA est utilisé comparer au routage XY pour toutes les heuristiques. Deuxièmement, les approches Manhattan MPNN et MPBN montrent une autre réduction. Troisièmement, la réduction de l'exécution de notre approche par rapport au approches existantes augmente à mesure que le nombre de tâches dans les applications envisagées est augmenté. Cela est dû au fait que les approches existantes rencontre un grand temps de recherche pour trouver les placements des tâches, alors que notre approche trouve les placements en moins de temps. La différence du temps de recherche entre nos approches et les approches existantes augmente avec le nombre de tâches dans les applications considérées. Par conséquent, notre approche fournit plus d'optimisation dans le temps total d'exécution pour des applications à grande taille.

Figure 5.13 montre la consommation d'énergie pour quatre séries d'applications

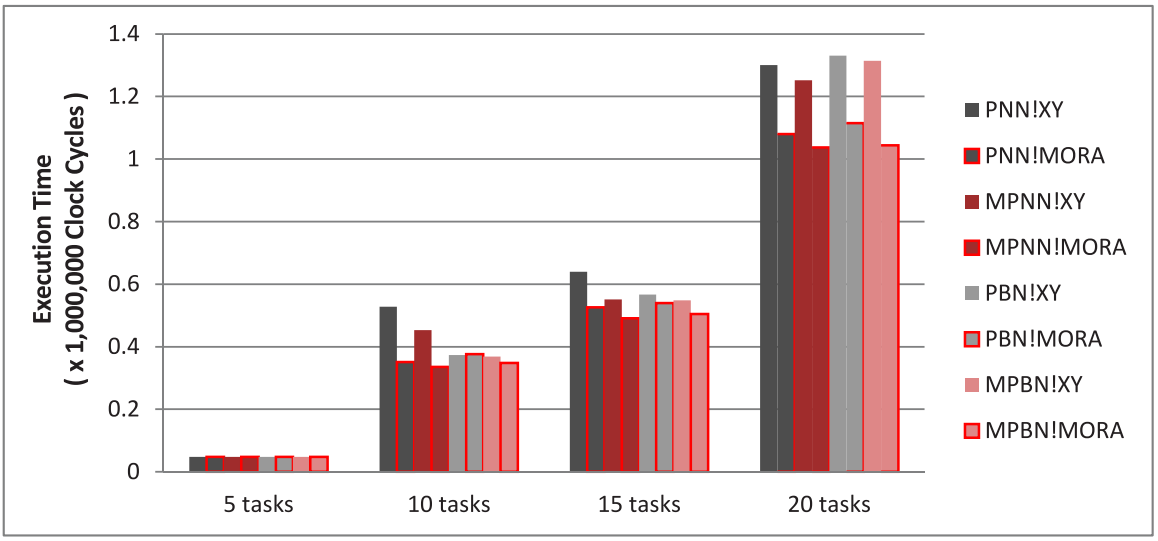


FIGURE 5.12 – Temps d'exécution de 10 applications pour quatre séries d'applications (scénario 4), où chaque application contient 5, 10, 15 et 20 tâches.

considérées dans le scénario 4. On peut observer que la réduction de la consommation d'énergie par notre approche par rapport aux approches existantes augmente de la même manière que le nombre de tâches dans les applications envisagées est augmentée. Par conséquent, notre approche donne de meilleurs optimisations pour les applications à grandes tailles.

Comparaison de complexité : Nombre de recherches

Nous avons calculé la complexité des différentes heuristiques en termes de nombre de recherches à effectuer pour le placement de toutes les tâches dans une série d'applications. Il a déjà été démontré que le nombre de recherches de la stratégie proposée Manhattan est inférieure aux stratégies existantes, par exemple 9 par rapport à 20 comme représenté sur la Fig. 5.4. Table 5.3 montre le nombre de recherches par différentes heuristiques pour quatre séries d'applications, où chaque série contient 10 applications. Chaque application dans les quatre séries contient 5, 10, 15 et 20 tâches, respectivement. Un couple d'observations peuvent être faites à partir de la Table5.3. Tout d'abord, le nombre de recherches est fortement réduit par les stratégies Manhattan MPNN et MPBN par rapport à PNN et PBN, respectivement. Deuxièmement, les stratégies Manhattan proposées montrent une réduction plus

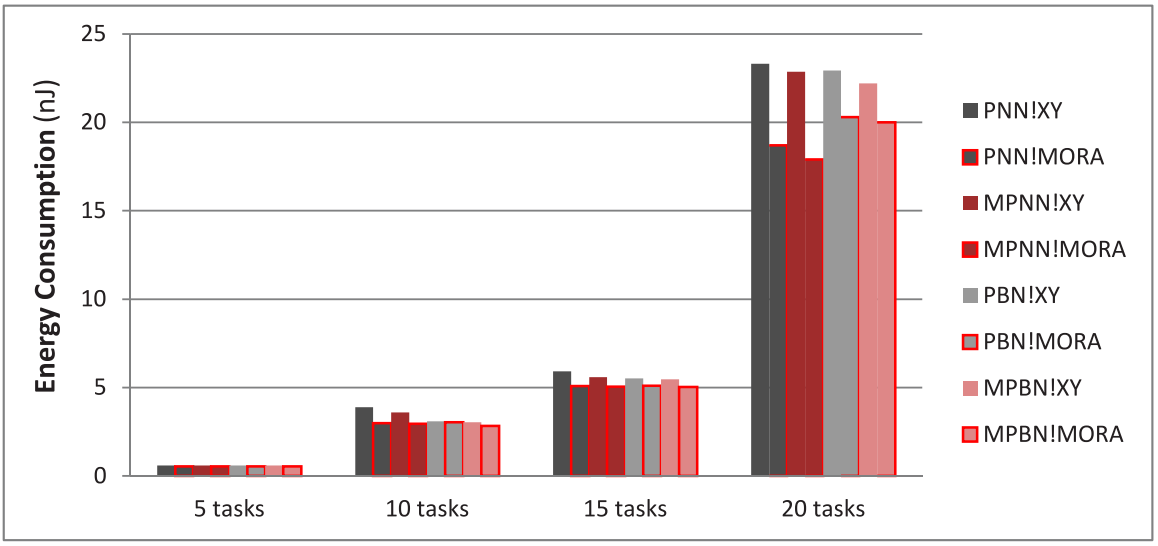


FIGURE 5.13 – Consommation d'énergie de 10 applications pour quatre séries d'applications (scénario 4), où chaque application contient 5, 10, 15 et 20 tâches.

	Apps-5tasks	Apps-10tasks	Apps-15tasks	Apps-20tasks
PNN	1690	10370	16360	46650
MPNN	1540	7950	12470	43210
PBN	2280	10800	17200	51130
MPBN	2260	9800	15130	48150

TABLE 5.3 – Nombre de recherches pour toutes les tâches dans différentes séries d'applications pour le scénario 4 on utilisant des stratégies de Manhattan et existantes.

élevée du nombre de recherches pour les applications à grand nombre de tâches. Il a également été observé que le nombre de recherches ne dépend pas de l'approche de routage appliquée. L'approche de routage est destinée aux placements des communications, mais pas les tâches. Ainsi, les stratégies de Manhattan réduisent la complexité et une autre réduction pour les applications à grand nombre de tâches. La comparaison de la complexité des différents algorithmes en termes de temps d'exécution montre que nos approches ont réduit la complexité.

5.7 Conclusion

Nous avons présenté dans ce chapitre, une approche de placement qui effectue le placement dynamique des applications maître-esclave sur des MPSoCs hétérogènes dans deux phases différentes. La première phase utilise de nouvelles stratégies de

placement proposées sur la base de la distance de Manhattan pour placer les tâches de l'application à proximité afin de réduire les coûts de communication. Autre avantage, est que les stratégies de Manhattan réduisent le temps de placement de chaque tâche. La deuxième phase place les communications entre les tâches sur un réseau sur puce. Pour réduire les coûts de communications, un algorithme de routage multi-objectifs (MORA) a été proposé . Les simulations ont montré que les stratégies proposées de placement Manhattan ainsi que le MORA montrent une réduction significative dans le temps total d'exécution et la consommation d'énergie par rapport aux approches existantes.

Chapitre 6

Conclusion générale et perspectives

Dans ce chapitre, nous résumons les contributions présentées dans cette thèse et nous concluons sur les possibles orientations futures de la recherche sur cette thématique.

6.1 Conclusion

Dans cette thèse nous avons proposé un certain nombre de nouvelles techniques de placement dynamique d'applications sur des architectures hétérogènes MPSoC à base de NoC. Un état de l'art approfondi nous a permis de bien se positionner et de pouvoir extraire et d'identifier les limites et les insuffisances des propositions existantes. Il a également été établi que la nécessité de passer aux MPSoCs s'avère indispensables vu la complexité croissante des applications actuelles et futurs. De même le passage aux MPSoCs hétérogènes garantit de meilleures performances. Le passage aux réseaux sur puce (NoCs) répond aux besoins de communication croissants entre les diverses unités de traitement sur une même puce. Les goulots d'étranglement de communication continuent d'être un sujet de préoccupation, ce qui limite les gains de performance dans les grandes réalisations de grande envergure. Dans cette thèse, nous avons proposé une nouvelle heuristique de placement dynamique en spirale pour le placement efficace des tâches d'applications sur une plateforme MPSoC hétérogène basée NoC contenant des éléments de traitement (PE) qui ne supportent qu'une seule tâche. Les techniques de placement existantes ne peuvent pas fournir de meilleurs résultats en comparaison avec notre proposition. La technique proposée place les tâches d'application sur les éléments de traitement MPSoC de manière à ce que toutes les tâches esclaves par maître soient placées le plus proche possible de la

tâche maître (les tâches esclaves sont placées autour de la tâche maître), conduisant à un placement des tâches efficaces. Nous avons ainsi proposé une deuxième nouvelle heuristique de placement des communications, afin de réduire les coûts des communications. Cette technique est basée sur l'algorithme Dijkstra modifié afin de trouver le chemin le plus court et le moins chargé afin d'optimiser les coûts des communications. Ces techniques montrent une amélioration des performances par rapport aux techniques existantes. Une troisième nouvelle heuristique de placement dynamique pour le placement efficace d'applications sur une plateforme MPSoC hétérogène basée sur un NoC contenant des éléments de traitement (PE) qui ne supportent qu'une seule tâche a été aussi proposée. On constate que les techniques de placement existantes ne peuvent pas fournir de meilleurs résultats dans le cas des applications plus larges (applications qui contiennent un nombre de tâches important). La technique proposée place les tâches d'applications sur les éléments de traitement MPSoC de manière très systématique. L'approche proposée utilise une stratégie de packing basée sur la distance de Manhattan et met le point sur le temps de mapping (le temps de recherche d'une ressource libre) qui peut influencer les performances du système, ce temps est minimisé d'une façon considérable pendant que les performances sont optimisées. La plupart des travaux existants traitant la même problématique, utilisent pour le placement des communications, le routage statique à savoir le XY, pour sa simplicité d'implémentation. Une quatrième nouvelle heuristique de routage multi-objectif (MORA) pour le placement dynamique des communications a aussi été proposée dans ce travail. Le but du MORA est d'acheminer les paquets échangés par le lien le moins chargé sur les liaisons point a point et vers la destination de façon à réduire les coûts des communications. L'approche MORA montre une amélioration des performances par rapport aux techniques existantes à savoir le routage statique XY. Ces techniques montrent une amélioration des performances par rapport aux techniques existantes. Comme dernière contribution dans cette thèse est la réalisation d'une plateforme de simulation qui permet de simuler n'importe quelle plateforme multi-cœurs interconnectés par un réseau sur puce homogène ou hétérogène de n'importe quelle taille. Exécuter et tester les propositions algorithmiques

de placement dynamique d'applications sur une plateforme simulé. En conclusion, cette thèse a présenté de nouvelles techniques de placement dynamique pour gérer le dynamisme dans les applications qui sont lancées au moment de l'exécution. Les techniques ont montrées un grand potentiel pour divers type d'applications multi-médias et ont un potentiel de données de bons résultats pour autre domaine d'applications qui présentent un dynamisme. Il est à noter que les techniques proposées minimisent les temps d'exécutions sans compromettre à la performance globale de calcul.

6.2 Perspectives

Cette section présente brièvement quelques orientations de travaux de recherches futurs qui peuvent étendre ou compléter le travail de cette thèse.

- L'extension du simulateur afin de permettre aux ressources (éléments de traitements) de supporter du multi-tâches c.à.d que chaque ressource peut supporter plusieurs tâches. le simulateur actuel ne permet pour chaque ressource de supporter qu'une seule tâche (Mono-tâche).
- L'augmentation de l'hétérogénéité dans les éléments de traitement : Les techniques de placement proposées dans cette thèse tiennent compte essentiellement de deux types d'éléments de traitement (PEs), processeur à usage général (GPP) et le matériel reconfigurable (RA). Cependant, il serait intéressant d'étendre ce travail à prendre en charge l'augmentation du degré d'hétérogénéité et d'intégrer différents types de PEs. Ceci doit être adressé avec précaution en raison de l'explosion potentielle du nombre de permutations qui doivent être prises en compte à chaque étape. Bien que les techniques proposées puissent être étendues pour l'hétérogénéité accrue dans les PEs, ils auront besoin pour établir une limite supérieure d'hétérogénéité dans le but de maintenir la faible complexité des techniques.
- Réaliser le placement dynamique sur une plateforme matérielle à base de FPGA : Il serait intéressant d'évaluer les techniques de placement dynamique proposées sur une telle plateforme. Celle ci prévoit l'évaluation détaillée de la

communication et de calcul des coûts pour chacune des méthodes proposées. En outre, les ressources supplémentaires nécessaires et leurs conséquences sur les performances de calcul peuvent être examinées en détail. Il est utile d'explorer la possibilité d'accélérer le processus de placement dynamique dans le matériel.

- Architecture Dynamique reconfigurable pour l'amélioration de la performance : Les techniques de placement visent une architecture fixe actuellement. Cependant, avec la possibilité de mettre en œuvre une plateforme hétérogène MPSoC à base de NoC sur un FPGA, les techniques de placement pourraient être encore étendues pour supporter la personnalisation dynamique de l'architecture MPSoC de manière à optimiser l'utilisation des PEs. Il est envisagé que l'architecture de MPSoC pourrait être modifiée pour supporter des scénarios de placement alternatifs pour satisfaire à la fois les exigences fonctionnelles et non fonctionnelles. La personnalisation dynamique comprendra la reconfiguration des blocs FPGA pour différents type de PEs afin d'atteindre une performance maximale. La performance devrait s'améliorer avec l'hétérogénéité accrue dans l'architecture, mais avec le sûrcout d'une reconfiguration. Par conséquent, les techniques étendues devront maximiser la performance tout en minimisant le temps de configuration.
- Migration et équilibrage de charge : Pour permettre un redéploiement d'applications pour des besoins de performances ou de nouvelles applications qui demandes a être placée des propositions de décision de migration doivent être prise en charge. De même pour garantir un redéploiement en cas de panne d'un ou plusieurs cœurs de calcul dans la plateforme. Finalement garantir un équilibrage de charge dans la plateforme.
- Solveur Dynamique pour la clusterisation : Le travail actuel considère une clusterisation statique prédéfinie au départ pour une bonne répartition des tâches initiales d'applications différentes affin de garantir une bonne répartition des tâches d'applications sur la plateforme. Pour des besoins de performances travailler sur des propositions de clusterisations dynamique pour des architectures

a grandes échelles ne peu qu'augmenter les performances.

Bibliographie

[1] *INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS (ITRS)*, 2008. http ://www.itrs.net/Links/2008ITRS/Home2008.htm.

[2] *Amdahls Law Demonstration.* http ://en.wikipedia.org/wiki/Amdahlslaw.

[3] *IMEC MPSoC Mapping Tools*, 2008. http ://www.imec.be/ScientificReport/SR2008/HTML/1225004.html.

[4] L. Smit, G. Smit, J. Hurink, H. Broersma, D. Paulusma, and P. Wolkotte. Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture. In *Proc. of International Conference on Field-Programmable Technology*, pages 421–424, December 2004.

[5] E. Gordon. Cramming more components onto integrated circuits. *Electronics Magazine*, 4 :114–117, 1965.

[6] Jerraya Ahmed Amine, Tenhunen Hannu, and Wolf Wayne. Guest editors' introduction : Multiprocessor systems-on-chips. *IEEE Computer*, 38(7) :36–40, 2005.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.

[8] *The keys to success in multicore application development*, 2009. http ://www.embedded.com/design/operating-systems/4008333/2/The-keys-to-success-in-multicore-application-development.

[9] P. Ross. *Why CPU Frequency Stalled*, April 2008. http ://spectrum.ieee.org/computing/hardware/why-cpu-frequency-stalled.

[10] *MPSoC Forum.* http ://www.mpsoc-forum.org/.

[11] *Mapping Applications to MPSoCs*, 2009. http ://www.artistembedded.org/artist/Overview1614.html.

[12] S. Borkar. Thousand core chips : a technology perspective. In *Proc. of Design Automation Conference*, pages 746–749, 2007.

[13] Asanovic Krste, Bodik Ras, Catanzaro Bryan Christopher, Gebis Joseph James, Husbands Parry, Keutzer Kurt, Patterson David, Plishker William Lester, Shalf John, Williams Samuel Webb, and Yelick Katherine. The landscape of parallel computing research : A view from berkeley. Technical report, EECS Department, University of California, Berkeley, Dec 2006.

[14] L. Benini and G. De Micheli. Networks on chips : a new soc paradigm. *IEEE Computer*, 35(1) :70 – 78, 2002.

[15] J. Henkel, W. Wolf, and S. Chakradhar. On-chip networks : A scalable, communication-centric embedded system design paradigm. In *Proc. of VLSI Design*, pages 845–851, 2004.

[16] D. Bertozzi and L. Benini. Xpipes : a network-on-chip architecture for gigascale systems-on-chip. *Cir. and Sys. Magazine, IEEE*, 4(2) :18–31, 2004.

[17] M. Hill and M. Marty. Amdahls law in the multicore era. *Computer*, 41(7) :33–38, July 2008.

[18] John L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31(5) :532–533, may 1988.

[19] *Raw Architecture Workstation (RAW)*, 2003. http ://groups.csail.mit.edu/cag/raw.

[20] *Asynchronous Array of Simple Processors (AsAP).* http ://www.ece.ucdavis.edu/ vcl/asap/.

[21] *Tera-op, Reliable, Intelligently adaptive Processing System (TRIPS).* http ://www.cs.utexas.edu/ trips/index.html.

[22] *WaveScalar processor*, 2006. http ://wavescalar.cs.washington.edu/index.html.

[23] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. Hs-scale : a hardware-software scalable mp-soc architecture for embedded systems. In *Proc. of ISVLSI*, pages 21–28, 2007.

[24] *First 100-core Processor with the New TILE-Gx Family*, 2009. http ://www.tilera.com/.

[25] *Multi-core chips by academia and industry.* http ://en.wikipedia.org/wiki/Multicore.

[26] S. Richardson. Mpoc : A chip multiprocessor for embedded systems. Technical report, HP Laboratories Technical Report HPL, 186, Palo Alto, CA, USA, 2002.

[27] *MPPA 256.* http ://www.kalray.eu/products/mppa-manycore/mppa-256/.

[28] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal. Run-time management of a mpsoc containing fpga fabric tiles. *IEEE Trans. Very Large Scale Integr. Syst.*, 16 :24–33, January 2008.

[29] G. J. Smit, A. B. Kokkeler, P. T. Wolkotte, and M. D. van de Burgwal. Multi-core architectures and streaming applications. In *Proc. of international workshop on System level interconnect prediction*, pages 35–42, 2008.

[30] *4S - Smart ChipS for Smart Surroundings*, 2007. http ://caes.ewi.utwente.nl/research/8-projects/8.

[31] T. Arpinen, P. Kukkala, E. Salminen, M. Hiken, and T. D. Hlen. Configurable multiprocessor platform with rtos for distributed execution of uml 2.0 designed applications. In *Proc. of Design, automation and test in Europe*, pages 1324 –1329, 2006.

[32] P. Heysters and G. Smit. Mapping of dsp algorithms on the montium architecture. In *proc. of International Parallel and Distributed Processing Symposium*, pages 22 – 26, April 2003.

[33] *Altera FPGAs.* http ://www.altera.com/.

[34] P. G. Paulin, C. Pilkington, E. Bensoudane, M. Langevin, and D. Lyonnard. Application of a multi-processor soc platform to high-speed packet forwarding. In *Proc. of conference on Design, automation and test in Europe*, pages 58–63., 2004.

[35] J. Leijten, J. van Meerbergen, A. Timmer, and J. Jess. Prophid : a heterogeneous multi-processor architecture for multimedia. In *Proc. of International Conference on Computer Design*, pages 164–169, 1997.

[36] M. J. Rutten, J. T. J. van Eijndhoven, E. G. T. Jaspers, P. van der Wolf, E.-J. D. Pol, O. P. Gangwal, and A. Timmer. A heterogeneous multiprocessor architecture for flexible media processing. *IEEE Des. Test*, 19 :39–50, 2002.

[37] E. Salminen, A. Kulmala, and T. D. Hamalainen. On network-on-chip comparison. In *Proc. of Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 503–510, 2007.

[38] R. Marculescu, U. Ogras, L.-S. Peh, N. Jerger, and Y. Hoskote. Outstanding research problems in noc design : System, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(1) :3–21, January 2009.

[39] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino. Spin : A scalable, packet switched, on-chip micro-network. In *Proc. of Design, Automation and Test in Europe*, pages 70 –73, 2003.

[40] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip : Concepts, architectures, and implementations. *IEEE Des. Test*, 22(5) :414–421, 2005.

[41] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Qnoc : Qos architecture and design process for network on chip. *J. Syst. Archit*, 50 :105–128, February 2004.

[42] C. Hilton and B. Nelson. Pnoc : a flexible circuit-switched noc for fpga-based systems. In *Proc. of IEEE Proceedings on Computers and Digital Techniques*, volume 153, pages 181 –188, May 2006.

[43] D. Castells-Rufas, J. Joven, and J. Carrabina. A validation and performance evaluation tool for protonoc. In *Proc. of International Symposium on System-on-Chip*, pages 1 – 4, November 2006.

[44] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *Proc. of Design, automation and test in Europe*, pages 890 – 895, 2004.

[45] T. Bjerregaard and S. Mahadevan. A survey of research and practices of networkon-chip. *ACM Comput. Surv.*, 38(1), 2006.

[46] F. Moraes, N. Calazans, A. Mello, L. Mller, , and L. Ost. Hermes : an infrastructure for low area overhead packet-switching networks on chip. *Integr. VLSI J.*, 38(1) :69–93, 2004.

[47] U. Y. Ogras and R. Marculescu. Energy- and performance-driven noc communication architecture synthesis using a decomposition approach. In *Proc. of Design, Automation and Test in Europe*, pages 352–357, 2005.

[48] A. Chagoya-Garzon, X. Guerin, F. Rousseau, F. Petrot, D. Rossetti, A. Lonardo, P. Vicini, and P. S. Paolucci. Synthesis of communication mechanisms for multitile systems based on heterogeneous multi-processor system-on-chips. In *Proc. of IEEE/IFIP International Symposium on Rapid System Prototyping*, pages 48–54, 2009.

[49] Z. Yang, A. Kumar, and Y. Ha. An area-efficient dynamically reconfigurable spatial division multiplexing network-on-chip with static throughput guarantee. In *Proc. of International Conference on Field-Programmable Technology*, pages 389 –392, December 2010.

[50] *Sonics Inc : The Nettwork-on-Chip Company.* http ://www.sonicsinc.com/.

[51] *Arteris : The Nettwork-on-Chip Company.* http ://www.arteris.com/.

[52] J. Hu and R. Marculescu. Dyad : smart routing for networks-on-chip. In *Proc. of Design Automation Conference*, pages 260 – 263, 2004.

[53] S. Murali, D. Atienz, L. Benini, and G. De Michel. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proc. of 43rd annual Design Automation Conference*, pages 845–848, 2006.

[54] G. Martin. Overview of the mpsoc design challenge. In *Proc. of Design Automation Conference*, pages 274–279, 2006.

[55] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. Mparm : Exploring the multi-processor soc design space with systemc. *J. VLSI Signal Process. Syst*, 41(2) :169–182, 2005.

[56] M. Monchiero, G. Palermo, C. Silvano, and O. Villa. A modular approach to model heterogeneous mpsoc at cycle level. In *Proc. of EUROMICRO Conference on Digital System Design Architectures*, pages 158–164, 2008.

[57] J. Cong, K. Gururaj, G. Han, A. Kaplan, M. Naik, , and G. Reinmanin. Mcsim : an efficient simulation tool for mpsoc designs. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, pages 364–371, 2008.

[58] Y. Atat and N.-E. Zergainoh. Simulink-based mpsoc design : New approach to bridge the gap between algorithm and architecture design. In *Proc. of IEEE Computer Society Annual Symposium on VLSI*, pages 9–14, 2007.

[59] *SystemC*. http ://www.accellera.org/home/.

[60] P. Paulin, C. Pilkington, and E. Bensoudane. Stepnp : A system-level exploration platform for network processors. *IEEE Des. Test*, 19 :17– 26, 2002.

[61] G. Beltrame, D. Sciuto, C. Silvano, P. Paulin, and E. Bensoudane. An application mapping methodology and case study for multi-processor on-chip architectures. In *Proc. of IFIP International Conference on Very Large Scale Integration*, pages 146–151, October 2006.

[62] H. Nikolov, T. Stefanov, and E. Deprettere. Systematic and automated multiprocessor system design, programming, and implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27 :542–555, March 2008.

[63] M. D. Nava, P. Blouet, P. Teninge, M. Coppola, T. Ben-Ismail, S. Picchiottino, and R. Wilson. An open platform for developing multiprocessor socs. *Computer*, 38(7) :60–67, 2005.

[64] D. Atienza, P. G. Del Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, and J. M. Mendias. A fast hw/sw fpga-based thermal emulation framework for

multiprocessor system-on-chip. In *Proc. of 43rd annual Design Automation Conference*, pages 618–623, 2006.

[65] F. Sun, S. Ravi, A. Raghunathan, and N. Jha. Application-specific heterogeneous multiprocessor synthesis using extensible processors. *Proc. of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(9) :1589–1602, September 2006.

[66] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal. Multiprocessor systems synthesis for multiple use-cases of multiple applications on fpga. *ACM Trans. Des. Autom. Electron. Syst.*, 13(3) :1–27, 2008.

[67] *Tensilica Inc.* http ://www.tensilica.com/.

[68] S. Lukovic and L. Fiorin. An automated design flow for noc-based mpsocs on fpga. In *Proc. of IEEE/IFIP International Symposium on Rapid System Prototyping*, pages 58–64, 2008.

[69] S. V. Tota, M. R. Casu, M. R. Roch, L. Macchiarulo, and M. Zamboni. A case study for noc-based homogeneous mpsoc architectures. *IEEE Trans. Very Large Scale Integr. Syst.*, 17 :384–388, March 2009.

[70] A. Kumar, A. Hansson, J. Huisken, and H. Corporaal. Interactive presentation : An fpga design flow for reconfigurable network-based multi-processor systems on chip. In *Proc. of Design, automation and test in Europe*, pages 117–122, 2007.

[71] T. Dorta, J. Jimenez, J. Martin, U. Bidarte, and A. Astarloa. Overview of fpga based multiprocessor systems. In *Proc. of International Conference on Reconfigurable Computing and FPGAs*, pages 273–278, December 2009.

[72] G.-G. Mplemenos and I. Papaefstathiou. Mplem : An 80-processor fpga based multiprocessor system. In *Proc. of International Symposium on Field-Programmable Custom Computing Machines*, pages 273–274, April 2008.

[73] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz. Ramp blue : A message-passing manycore system in fpgas. In *Proc. of Internatio-*

nal Conference on Field Programmable Logic and Applications, pages 54–61, August 2007.

[74] J. Ceng, J. Castrillon, W. Sheng, H. Scharwachter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda. Maps : an integrated framework for mpsoc application parallelization. In *Proc. of Design Automation Conference*, pages 754–759, 2008.

[75] R. P. Dick, D. L. Rhodes, and W. Wolf. Tgff : task graphs for free. In *Proc. of international workshop on Hardware/software codesign*, pages 97–101, 1998.

[76] S. Stuijk, M. Geilen, and T. Basten. Sdf3 : Sdf for free. In *Proc. of International Conference on Application of Concurrency to System Design*, pages 276–278, June 2006.

[77] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano. Communication aware allocation and scheduling framework for stream-oriented multiprocessor systems-on-chip. In *Proc. of conference on Design, automation and test in Europe*, pages 3–8, 2006.

[78] M. Ruggiero, A. Guerri, D. Bertozzi, M. Milano, and L. Benini. A fast and accurate technique for mapping parallel applications on stream-oriented mpsoc platforms with communication awareness. *Int. J. Parallel Program.*, 36(1) :3–36, 2008.

[79] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4) :551–562, April 2005.

[80] C. Marcon, A. Borin, A. Susin, L. Carro, and F.Wagner. Time and energy efficient mapping of embedded applications onto nocs. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 33–38, 2005.

[81] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli. A methodology for mapping multiple use-cases onto networks on chips. In *Proc. of conference on Design, automation and test in Europe*, pages 118–123, 2006.

[82] C.-E. Rhee, H.-Y. Jeong, and S. Ha. Many-to-many core-switch mapping in 2-d mesh noc architectures. In *Proc. of IEEE International Conference on Computer Design*, pages 438–443, 2004.

[83] Lei Tang and Kumar Shashi. Algorithms and tools for network on chip based system design. In *Proc. of Integrated circuits and systems design*, page 163, 2003.

[84] D. Wu, B. M. Al-Hashimi, and P. Eles. Scheduling and mapping of conditional task graphs for the synthesis of low power embedded systems. In *Proc. of conference on Design, Automation and Test in Europe*, page 10090, 2003.

[85] S. Manolache et al. Fault and energy-aware communication mapping with guaranteed latency for applications implemented on noc. In *Proc. of DAC*, pages 266–269, 2005.

[86] L. Lin et al. Communication-driven task binding for multiprocessor with latency insensitive network-on-chip. In *Proc. of ASP-DAC*, pages 39–44, 2005.

[87] H. Orsila et al. Automated memory-aware application distribution for multiprocessor system-on-chips. *J. Syst. Archit.*, 53(11) :795–815, 2007.

[88] M. Branca, L. Camerini, F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo. Evolutionary algorithms for the mapping of pipelined applications onto heterogeneous embedded systems. In *Proc. of Annual conference on Genetic and evolutionary computation*, pages 1435–1442, 2009.

[89] L. Thiele, I. Bacivarov, W. Haid, and K. Huang. Mapping applications to tiled multiprocessor embedded systems. In *Proc. of International Conference on Application of Concurrency to System Design*, pages 29–40, 2007.

[90] G. Chen, F. Li, S. Son, and M. Kandemir. Application mapping for chip multiprocessors. In *Proc. of ACM/IEEE Design Automation Conference*, pages 620–625, June 2008.

[91] N. Satish, K. Ravindran, and K. Keutzer. A decomposition-based constraint optimization approach for statically scheduling task graphs with communica-

tion delays to multiprocessors. In *Proc. of conference on Design, automation and test in Europe*, pages 57–62, 2007.

[92] *Abou El Hassan Benyamina, Ordonnancement Hierarchique Multi-Objectif DApplication Embarquees Intensives, decembre 2008, Oran*, 2008. http ://forge.lifl.fr/AASGaspard/wiki/DocumentsDeReference.

[93] *Aroui Abdelkader, Ordonnacement des taches repetitives, 2011, Oran*, 2011. http ://forge.lifl.fr/AASGaspard/wiki/Travaux2010-2011/MagistAroui.

[94] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang. Mapping of applications to mpsocs. In *Proc. of IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 109–118, 2011.

[95] Chou Chen-Ling and Marculescu Radu. User-aware dynamic task allocation in networks-on-chip. In *Proc. of Design, Automation and Test in Europe, DATE'08*, pages 1232–1237, 2008.

[96] Z. Peter, S. Gilles, U. Nurten, S. Nicolas, B. Pascal, and G. Manfred. A decentralised task mapping approach for homogeneous multiprocessor network-on-chips. *International Journal of Reconfigurable Computing*, 2009, 2009.

[97] E. W. Briao, D. Barcelos, and F. R. Wagner. Dynamic task allocation strategies in mpsoc for soft real-time applications. In *Proc. of conference on Design, automation and test in Europe*, pages 1386–1389, 2008.

[98] Chou Chen-Ling and Marculescu Radu. Incremental run-time application mapping for homogeneous nocs with multiple voltage levels. In *Proc. of 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS,07)*, pages 161–166, 2007.

[99] A. Ngouanga, G. Sassatelli, L. Torres, T. Gil, A. Soares, and A. Susin. A contextual resources use : a proof of concept through the apaches platform. In *Proc. of IEEE Design and Diagnostics of Electronic Circuits and systems*, pages 42–47, 2006.

[100] A. Mehran et al. Dsm : A heuristic dynamic spiral mapping algorithm for network on chip. *IEICE Electronics Express*, 5(13) :464–471, 2008.

[101] G. Sassatelli, N. Saint-Jean, P. Benoit, L. Torres, M. Robert, C. Woszezenki, I. A. Grehs, and F. Moraes. Run-time mapping and communication strategies for homogeneous noc-based mpsocs. In *Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines*, 2007.

[102] O. Moreira, J. J.-D. Mol, and M. Bekooij. Online resource management in a multiprocessor with a network-on-chip. In *Proc. of ACM symposium on Applied computing*, pages 1557–1564, 2007.

[103] P.Holzenspies, J.Hurink, J.Kuper, and G.Smit. Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (mpsoc). In *Proc. of Design, Automation and Test in Europe, DATE '08*, pages 212–217, 2008.

[104] T. D. ter Braak, P. K. F. Hlzenspies, J. Kuper, J. L. Hurink, , and G. J. M. Smit. Run-time spatial resource management for real-time applications on heterogeneous mpsocs. In *Proc. of Conference on Design, Automation and Test in Europe*, pages 357–362, 2010.

[105] M.Faruque, R.Krist, and J.Henkel. Adam : Run-time agent-based distributed application mapping for on-chip communication. In *Proc. of 45th ACM/IEEE Design Automation Conference, DAC' 08*, pages 760–765, 2008.

[106] A. Schranzhofer, J.-J. Chen, and L. Thiele. Power-aware mapping of probabilistic applications onto heterogeneous mpsoc platforms. In *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2009.

[107] Tang. Lei and Shashi. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proc. of the Euromicro Symposium on DSD*, page 180, 2003.

[108] E.Carvalho and F.Moraes. Congestion-aware task mapping in heterogeneous mpsocs. In *Proc. of International Symposium on System-on-Chip, SOC '08*, pages 1–4, 2008.

[109] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *Proc. of Conference on Design, Automation and Test in Europe*, pages 196–201, 2010.

[110] S. Stuijk, M. Geilen, and T. Basten. A predictable multiprocessor design flow for streaming applications with dynamic behaviour. In *Proc. of Euromicro Conference on Digital System Design : Architectures, Methods and Tools*, pages 548–555, 2010.

[111] B. Giovanni, L. Fossati, and D. Sciuto. Decision-theoretic design space exploration of multiprocessor platforms. *Trans. Comp. Aided Des. Integ. Cir. Sys.*, 29 :1083–1095, July 2010.

[112] G. Palermo, C. Silvano, and V. Zaccaria. Multi-objective design space exploration of embedded systems. *J. Embedded Comput.*, 1 :305–316, August 2005.

[113] Z. J. Jia, A. Pimentel, M. Thompson, T. Bautista, and A. Nunez. Nasa : A generic infrastructure for system-level mp-soc design space exploration. In *Proc. of IEEE Workshop on Embedded Systems for Real-Time Multimedia*, pages 41–50, October 2010.

[114] Hemayet Hossain, Mostak Ahmed, Abdullah Al-Nayeem, Tanzima Zerin Islam, and Md. Mostofa Akbar. gpnocsim - a general purpose simulator for network-on-chip. In *International Conference on Information and Communication Technology (ICICT)*, 2007.

[115] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James Balfour, Brian Towles, John Kim, and William J. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software*, 2013.

[116] *Noxim : the NoC Simulator.* http ://noxim.sourceforge.net/.

[117] *The Network Simulator ns-2.* http ://www.isi.edu/nsnam/ns/.

[118] *Darsim : A Parallel Cycle-Level NoC Simulator.* http ://dspace.mit.edu/handle/1721.1/59832.

[119] *Conception d'un simulateur de mapping dynamique sur une architecture heterogene MP-SOC basee NOC.* http ://forge.lifl.fr/AASGaspard/wiki/Travaux2012-2013/REKKABNABIL.

[120] A. K. Singh et al. Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms. *Journal of Systems Architecture*, 56(7) :242 – 255, 2010.

[121] D.Bertozzi and L.Benini. A network-on-chip architecture for gigascale systems-on-chip. *Circuits and Systems Magazine, IEEE*, 4(2) :18–31, 2004.

[122] Y. Zhang et al. Task scheduling and voltage selection for energy minimization. In *Proc. of Design Automation Conference*, pages 183 – 188, 2002.

[123] Shin Dongkun and Kim Jihong. Power-aware communication optimization for networks-on-chips with voltage scalable links. In *Proc. of CODES and ISSS*, pages 170–175, 2004.

[124] CL.Chou, C.Wu, and J.Lee. Integrated mapping and scheduling for circuit-switched networkon-chip architectures. In *proc. of 4th IEEE International Symposium on Electronic Design, Test and Applications, DELTA 2008.*, pages 415–420, 2008.

[125] F.Vardi, S.Saeidi, and A.Khademzadeh. Crinkle : A heuristic mapping algorithm for network on chip. *IEICE Electronics Express*, 6(24) :1737–1744, 2009.

[126] M.Armin, S.Saeidi, and A.Khademzadeh. Spiral : A heuristic mapping algorithm for network on chip. *IEICE Electronic Express*, 4(15) :478–484, 2007.

[127] X.Wang, M.Yang, Y.Jiang, and P.Liu. Power-aware mapping for network-on-chip architectures under bandwidth and latency constraints. *ACM Transactions on Architecture and Code Optimization (TACO)*, 7(1) :6, 2010.

[128] P.Ghosh, A.Sen, and A.Hall. Energy efficient application mapping to noc processing elements operating at multiple voltage levels. In *Proc. of 3rd*

ACM/IEEE International Symposium on Networks-on-Chip, NoCS'09, pages 80–85, 2009.

[129] E. Carvalho et al. Evaluation of static and dynamic task mapping algorithms in noc-based mpsocs. In *Proc. of 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, pages 87–90, 2009.

[130] Smit et al. Run-time mapping of applications to a heterogeneous soc. In *Proc. of Design, Automation and Test in Europe Conference and Exhibition*, 2004.

[131] P. Holzenspies et al. Mapping streaming applications on a reconfigurable mpsoc platform at run-time. In *Proc. of International Symposium on System-on-Chip, SOC'07*, pages 1–4, 2007.

[132] Marcelo Mandelli et al. Multi-task dynamic mapping onto noc-based mpsocs. In *Proceedings of the 24th symposium on Integrated circuits and systems design, SBCCI'11*, pages 191–196, 2011.

[133] S.Wildermann, T.Ziermann, and J.Teichet. Run time mapping of adaptive applications onto homogeneous noc-based reconfigurable architectures. In *Proc. of International Conference on Field-Programmable Technology, FPT'09*, pages 514 – 517, 2009.

[134] A.Schranzhofer, C.Jian-Jia, L.Santinelli, and L.Thiele. Dynamic and adaptive allocation of applications on mpsoc platforms. In *Proc. of 15th Asia and South Pacific Design Automation Conference, ASP-DAC '10*, pages 885 –890, 2010.

[135] E.Carvalho, N.Calazans, and F.Moraes. Dynamic task mapping for mpsocs. *IEEE Design Test of Computers*, 27(5) :26–35, 2010.

[136] A. K. Singh et al. Efficient heuristics for minimizing communication overhead in noc-based heterogeneous mpsoc platforms. In *Proc. of IEEE/IFIP International Symposium on Rapid System Prototyping, RSP '09*, pages 55 – 60, 2009.

[137] S. Vangal et al. An 80-tile 1.28tflops network-on-chip in 65nm cmos. In *Proc. of IEEE Int. Solid-State Circuits Conf.*, pages 98–589, Feb. 2007.

[138] V. Nollet et al. Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles. In *Proc. of DATE*, pages 234–239, 2005.

[139] Chip multiprocessor watch, 2008. http://view.eecs.berkeley.edu/wiki/Chip_Multi_Processor_Watch.

[140] M. Kistler et al. Cell multiprocessor communication network : Built for speed. *IEEE Micro*, 26(3) :10–23, 2006.

[141] A. K. Singh et al. Accelerating throughput-aware runtime mapping for heterogeneous mpsocs. *ACM Trans. Design Autom. Electr. Syst.*, 18(1) :9, 2012.

[142] A. Benyamina et al. Mapping real time applications on noc architecture with hybrid multi-objective algorithm. In *Proc. of META*, 2010.

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

[144] Luciano Ost et al. Power-aware dynamic mapping heuristics for noc-based mpsocs using a unified model-based approach. *ACM Trans. Embedded Comput. Syst.*, 12(3) :75, 2013.

[145] Marcelo Mandelli et al. Energy-aware dynamic task mapping for noc-based mpsocs. In *Proc. of ISCAS*, 2011.

[146] Andreas Weichslgartner et al. Dynamic decentralized mapping of tree-structured applications on noc architectures. In *Proc. of NOCS*, 2011.

[147] Leandro Möller, Ismael Grehs, Ewerson Carvalho, Rafael Soares, Ney Calazans, and Fernando Moraes. A noc-based infrastructure to enable dynamic self reconfigurable systems, 2007.