Université de Montréal

Approche efficace pour la conception des architectures multiprocesseurs sur puce électronique

par
Etienne Elie

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des arts et des sciences
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Decembre, 2010

Université de Montréal
Faculté des arts et des sciences

Cette thèse intitulée:

Approche efficace pour la conception des architectures multiprocesseurs sur puce électronique

présentée par:

Etienne Elie

a été évaluée par un jury composé des personnes suivantes:

El Mostapha Aboulhamid,	président-rapporteur
Abdelhakim Hafid,	directeur de recherche
Jacques A. Ferland,	codirecteur
Marcel Turcotte,	codirecteur
Fabian Bastin,	membre du jury
Mounir Boukadoum,	examinateur externe
Jean Saulnier,	représentant du doyen de la FAS

Thèse acceptée le: . . . . . .

RÉSUMÉ

Les systèmes multiprocesseurs sur puce électronique (On-Chip Multiprocessor [OCM]) sont considérés comme les meilleures structures pour occuper l'espace disponible sur les circuits intégrés actuels. Dans nos travaux, nous nous intéressons à un modèle architectural, appelé architecture isométrique de systèmes multiprocesseurs sur puce, qui permet d'évaluer, de prédire et d'optimiser les systèmes OCM en misant sur une organisation efficace des nœuds (processeurs et mémoires), et à des méthodologies qui permettent d'utiliser efficacement ces architectures.

Dans la première partie de la thèse, nous nous intéressons à la topologie du modèle et nous proposons une architecture qui permet d'utiliser efficacement et massivement les mémoires sur la puce. Les processeurs et les mémoires sont organisés selon une approche isométrique qui consiste à rapprocher les données des processus plutôt que d'optimiser les transferts entre les processeurs et les mémoires disposés de manière conventionnelle. L'architecture est un modèle maillé en trois dimensions. La disposition des unités sur ce modèle est inspirée de la structure cristalline du chlorure de sodium (NaCl), où chaque processeur peut accéder à six mémoires à la fois et où chaque mémoire peut communiquer avec autant de processeurs à la fois.

Dans la deuxième partie de notre travail, nous nous intéressons à une méthodologie de décomposition où le nombre de nœuds du modèle est idéal et peut être déterminé à partir d'une spécification matricielle de l'application qui est traitée par le modèle proposé. Sachant que la performance d'un modèle dépend de la quantité de flot de données échangées entre ses unités, en l'occurrence leur nombre, et notre but étant de garantir une bonne performance de calcul en fonction de l'application traitée, nous proposons de trouver le nombre idéal de processeurs et de mémoires du système à construire. Aussi, considérons-nous la décomposition de la spécification du modèle à construire ou de l'application à traiter en fonction de l'équilibre de charge des unités. Nous proposons ainsi une approche de décomposition sur trois points : la transformation de la spécification ou de l'application en une matrice d'incidence dont les éléments sont les flots de données entre les processus et les données, une nouvelle méthodologie basée sur le problème de

la formation des cellules (*Cell Formation Problem* [CFP]), et un équilibre de charge de processus dans les processeurs et de données dans les mémoires.

Dans la troisième partie, toujours dans le souci de concevoir un système efficace et performant, nous nous intéressons à l'affectation des processeurs et des mémoires par une méthodologie en deux étapes. Dans un premier temps, nous affectons des unités aux nœuds du système, considéré ici comme un graphe non orienté, et dans un deuxième temps, nous affectons des valeurs aux arcs de ce graphe. Pour l'affectation, nous proposons une modélisation des applications décomposées en utilisant une approche matricielle et l'utilisation du problème d'affectation quadratique (*Quadratic Assignment Problem* [QAP]). Pour l'affectation de valeurs aux arcs, nous proposons une approche de perturbation graduelle, afin de chercher la meilleure combinaison du coût de l'affectation, ceci en respectant certains paramètres comme la température, la dissipation de chaleur, la consommation d'énergie et la surface occupée par la puce.

Le but ultime de ce travail est de proposer aux architectes de systèmes multiprocesseurs sur puce une méthodologie non traditionnelle et un outil systématique et efficace d'aide à la conception dès la phase de la spécification fonctionnelle du système.

Mots clés : circuits intégrés, circuits intégrés en trois dimensions, graphe non orienté, optimalité, problème d'affectation quadratique (QAP), problème de formation de cellules (CFP), réseaux sur puce, système multiprocesseur sur puce (OCM System).

ABSTRACT

On-Chip Multiprocessor (OCM) systems are considered to be the best structures to occupy the abundant space available on today integrated circuits (IC). In our thesis, we are interested on an architectural model, called *Isometric on-Chip Multiprocessor Architecture* (ICMA), that optimizes the OCM systems by focusing on an effective organization of cores (processors and memories) and on methodologies that optimize the use of these architectures.

In the first part of this work, we study the topology of ICMA and propose an architecture that enables efficient and massive use of on-chip memories. ICMA organizes processors and memories in an isometric structure with the objective to get processed data close to the processors that use them rather than to optimize transfers between processors and memories, arranged in a conventional manner. ICMA is a mesh model in three dimensions. The organization of our architecture is inspired by the crystal structure of sodium chloride (*NaCl*), where each processor can access six different memories and where each memory can communicate with six processors at once.

In the second part of our work, we focus on a methodology of decomposition. This methodology is used to find the optimal number of nodes for a given application or specification. The approach we use is to transform an application or a specification into an incidence matrix, where the entries of this matrix are the interactions between processors and memories as entries. In other words, knowing that the performance of a model depends on the intensity of the data flow exchanged between its units, namely their number, we aim to guarantee a good computing performance by finding the optimal number of processors and memories that are suitable for the application computation. We also consider the load balancing of the units of ICMA during the specification phase of the design. Our proposed decomposition is on three points: the transformation of the specification or application into an incidence matrix, a new methodology based on the Cell Formation Problem (CFP), and load balancing processes in the processors and data in memories.

In the third part, we focus on the allocation of processor and memory by a two-step

methodology. Initially, we allocate units to the nodes of the system structure, considered here as an undirected graph, and subsequently we assign values to the arcs of this graph. For the assignment, we propose modeling of the decomposed application using a matrix approach and the Quadratic Assignment Problem (QAP). For the assignment of the values to the arcs, we propose an approach of gradual changes of these values in order to seek the best combination of cost allocation, this under certain metric constraints such as temperature, heat dissipation, power consumption and surface occupied by the chip.

The ultimate goal of this work is to propose a methodology for non-traditional, systematic and effective decision support design tools for multiprocessor system architects, from the phase of functional specification.

Keywords: Cell Formation Problem (CFP), Integrated Circuit (IC), On-Chip Multiprocessor (OCM), Network-on-Chip (NoC), Nonoriented Graph, Three Dimensions IC (3D-IC), Quadratic Assignment Problem (QAP).

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES

CFP	Cell Formation Problem
CN	Coordination Number
FPGA	Field-Programmable Gate Array
GIS	Gigascale Integrated System
HSC	Hardware/Software Codesign
IC	Integrated Circuit
ICMA	Isometric on-Chip Mutiprocessor Architecture
IP	Intellectual Property
MU	Memory Unit
MUI	Memory Unit Interface
OCM	On-Chip Multiprocessor
PE	Processing Element
PEI	Processing Element Interface
QAP	Quadratic Assignment Problem
XPS	Xilinx Platform Studio

NOTATION

m	Machines/processors quantity
n	Parts/memories quantity
I	Machines/processors set
J	Parts/memories set
A	Incidence matrix
IA	Interaction matrix
K	Number of cell
C_k	k^{th} machines/processors group
F_k	k^{th} parts/memories family
C	Machines/processors vector
F	Parts/memories vector
(C_k, F_k)	k^{th} processors and memories cell
(C, F)	Cell formation solution vector
(C^0, F_0)	Initial solution
F^*	Flow matrix
γ	Penalty factor
Eff	Efficacy of the cell formation
INA	Set of non assigned elements of A
$\mathcal{N}$	Graph representing OCM
$\bar{D}$	Connection time matrix
$L(e)$	Level of access going through link e
$cap(e)$	Minimum bus or link capacity
$\$(e)$	Cost of $cap(e)$ link capacity
CC	Total capacity cost

A Marie-Hélène

REMERCIEMENTS

L'une des richesses qu'on reçoit lors d'une thèse est la grâce de travailler avec des personnes merveilleuses. J'ai eu ce privilège d'avoir trois de ces personnes qui ont dirigé ma thèse. Je tiens à remercier premièrement mon directeur de recherche, Abdelhakim Hafid, pour sa direction et son enthousiasme contagieux. Durant toutes ces années, le professeur Hafid m'a donné la direction scientifique et personnelle, stimulé mon intelligence et à chaque fois, trouvé le mot qu'il faut pour m'encourager lorsque je suis arrivé dans cette phase de découragement que connaissent bien souvent beaucoup d'étudiants de doctorat. Je remercie Marcel Turcotte qui le premier a cru en moi lorsque je lui ai présenté mes idées de recherche. Durant la période que cette thèse a duré, le professeur Turcotte a toujours été là pour encourager, écouter et soutenir. Ses conseils ont été pour moi d'une valeur inestimable. Je tiens à témoigner toute ma gratitude au professeur Jacques Ferland, qui est devenu mon codirecteur durant cette thèse. J'ai bénéficié énormément de sa vaste connaissance en recherche opérationnelle et surtout de sa grande générosité. Sans ces incalculables nombres d'heures dans son bureau à chercher des solutions à nos problèmes, cette thèse n'aurait pas connu le dénouement qui est le sien aujourd'hui. Je tiens à le remercier davantage pour toutes les ressources qu'il a mises à ma disposition durant les deux ans de travail ensemble.

Je remercie particulièrement Jonathan Bellemare qui m'a soutenu fidèlement dans la programmation de nos idées. Il a toujours été disponible en tout temps pour aider.

Je tiens à remercier particulièrement les membres du jury, qui ont accepté d'évaluer ce travail.

Je remercie mes collègues du laboratoire LRC pour leur soutien ; l'ambiance chaleureuse qui régnait dans le groupe au début de ma thèse a été grandement appréciée. Je remercie mes connaissances de l'Université et en dehors de l'Université qui m'ont soutenu d'une manière ou d'une autre. Notamment Marie-Josée Boulay qui m'a donné

un bureau pour m'encourager à me concentrer sur le travail.

Je remercie mon ami et pasteur Calvin Wuntcha, et les membres de mon église, qui m'ont soutenu spécialement dans la prière durant toutes ces années.

Je remercie chaleureusement mes beaux-parents, Daniel et Diane Trempe, mon beau-frère Simon et mes deux belles-sœurs Martine et Sophie-Caroline, qui m'ont donné de l'amour et qui ont su m'encourager depuis que je suis entré dans la famille en 2007. Merci pour le bon café matinal de Daniel et les bons soupers de Diane chaque fois que je suis allé me réfugier dans le nord (Sainte-Thérèse-de-Blainville) pour chercher de l'inspiration.

Je remercie particulièrement mon père Émile Ogoubi, de qui j'ai appris à aimer et à chérir les livres. Merci pour toutes ces années où tu m'as initié à la littérature à travers la collection **Lagarde et Michard**. Je remercie également mes parents, mes frères et sœurs, et toute ma famille qui malgré la distance qui nous sépare ont su être présents durant cette thèse et m'ont toujours encouragé à persévérer.

Plus que tout, je remercie mon épouse Marie-Hélène Trempe. Les mots me manqueront pour exprimer la grâce et le privilège que j'ai d'être aussi bien marié et ce que cela a apporté à cette thèse. Je rends grâce à Dieu tous les jours de m'avoir donné une femme comme elle. Je lui dédie cette thèse.

CHAPITRE 1

INTRODUCTION

Dans ce chapitre, nous présentons le contexte, la motivations, les contributions et l'organisation de la thèse.

1.1 Contexte et motivations de la thèse

La conception d'un circuit intégré à plusieurs nœuds sur la même puce peut être considérée comme la résolution d'une fonction multi-objective impliquant la technologie de base (CMOS), les méthodes de spécification, d'architecture, de compilation, de vérification et d'affectation. Comme dans le cas de tout circuit intégré, trois composantes principales sont nécessaires pour une conception efficace des OCM : (1) la représentation de l'utilisateur (expression des besoins) ; (2) la technologie de base sur laquelle repose la fabrication physique ; et (3) l'architecture des systèmes OCM. Dans ce travail, nous nous attaquons à certains problèmes que posent la conception des circuits intégrés, en particulier les circuits intégrés à trois dimensions, et plus spécifiquement, les OCM avec leur technologie de communication, à savoir, les réseaux sur puce (*Network-on-Chip* (NoC)). Nous proposons une solution efficace qui prend en compte le degré de parallélisme dès la spécification du système à concevoir et au niveau architectural de la solution.

La suite de ce chapitre se divise en quatre parties, à savoir : (1) la méthodologie de la conception des circuits intégrés, et plus précisément, des systèmes OCM ; (2) quelques problématiques liées à la conception des systèmes sur circuits intégrés performants en trois dimensions ; (3) notre contribution à la résolution de certains de ces problèmes, et (4) le plan de la thèse.

1.2 La conception des systèmes multiprocesseurs sur puce

La conception d'un système OCM, comme tout autre système électronique, consiste à représenter des fonctions souhaitées par les utilisateurs sur une puce électronique. Cette

conception est basée sur une technologie qui permet de fabriquer un modèle physique du système souhaité. Ainsi, la faculté de créer ou de concevoir un système efficace relève de deux facteurs : l'efficacité de la technologie de base à implémenter adéquatement le modèle, et l'efficacité du modèle à représenter adéquatement les fonctions souhaitées. Toutefois, il y a un besoin de s'assurer que les fonctions souhaitées par les utilisateurs, le modèle et la technologie de base sont cohérents. En ce qui concerne la technologie de base, il y a eu beaucoup de progrès remarquables ces dernières années. Ces progrès permettent de créer des systèmes complexes à des vitesses de calcul impressionnantes. De ce fait, le travail de l'architecte système est devenu plus laborieux. Les contraintes imposées pour la conception des systèmes comme la taille, la fonctionnalité, la performance, la minimisation de la consommation d'énergie et la densité structurelle des puces sont devenues difficiles à intégrer. Aussi, lors de la conception d'un système sur puce, les ingénieurs doivent prendre en compte le peu de temps dont ils disposent, vu le fait que nous sommes dans un domaine concurrentiel.

Plusieurs méthodes et outils sont proposés pour réduire la complexité du processus de conception des OCM. Nous citons, en exemple, celles basées sur une approche hiérarchique.

1.2.1 Technologies de base

Durant les cinq dernières années, les besoins des utilisateurs, de plus en plus croissants dans l'industrie des nouvelles technologies, ont permis de découvrir de nouveaux matériaux comme le graphène, les circuits intégrés en trois dimensions (*Three Dimension Integrated Circuits* [3D-IC]) et l'intégration verticale des systèmes sur puce (*3D-IC Packaging* [3D-ICP]). Dans nos travaux, notre méthodologie s'appuie sur ces deux dernières technologies de base.

Les 3D-IC sont des puces électroniques dont les unités actives (processeurs, mémoires, DSP, SRAM, DRAM, FLASH, ...) sont organisées horizontalement et verticalement. Si la dimension horizontale est souvent composée de plusieurs unités électroniques comme dans la conception classique de circuits intégrés, la dimension verticale, quant à elle, est nouvelle et considérée comme une superposition verticale des

couches de circuits en deux dimensions, interconnectées par des canaux verticaux appelés « *Through-Silicon Via* (TSV) » [71]. Bien que cette thèse n'a pas pour mission de présenter en détail la technologie de la fabrication physique des 3D-IC, nous présentons dans cette section quelles sont les forces et les faiblesses de cette technologie et en quoi elle soutient notre démarche.

Les 3D-IC, vu leur aspect vertical, offrent une possibilité d'accroitre les performances que le structure 2D ne peuvent pas donner à ce jour. Leur avantage est que les composantes des circuits intégrés sont disposées de telle sorte que les voisins immedits d'une composante sont plus nombreux que dans le cas des circuits traditionnels. La disposition verticale des composante permet ainsi d'augmenter la performance grâce à une bande passante élevée pour les transferts de données [165]. D'autres avantages des 3D-IC sont l'augmentation de la densité fonctionnelle d'une part et la diminution de la taille des circuits d'autre part. Aussi, la disposition verticale entraine la réduction des canaux de connexion, ce qui entraine une augmentation de l'immunité aux bruits et, par conséquent, une augmentation de l'intégrité des données transmises d'une couche à une autre. Par ailleurs, la réduction des longueurs des canaux de transfert entraine une diminution de la capacitance nécessaire pour la propagation des signaux de données dans le circuit ; ceci entraine *de facto* une diminution de l'énergie consommée et de la chaleur dissipée par unité fonctionnelle [104].

Malgré les avantages prometteurs que nous venons de citer, les 3D-IC sont encore à la phase d'études et sont utilisés de façon marginale dans l'industrie des circuits intégrés. Plusieurs défis restent à relever. Citons en exemple le problème de dissipation de la chaleur. En effet, si les canaux d'interconnexion verticaux réduisent grandement la consommation d'énergie et par conséquent la chaleur dissipée au niveau individuel de chaque canal, il est évident que l'augmentation de la densité fonctionnelle et de la complexité des puces entrainera une augmentation substantielle de la chaleur entre les couches. Un autre problème que produisent les 3D-IC est l'augmentation de la complexité de conception de ces circuits. Les outils de conception dont nous disposons dans l'industrie étant surtout orientés vers la conception des circuits sur 2D, qui sont considérés comme les plus utilisés dans la conception des circuits intégrés [19], [89] de nouveaux outils de

conception assistée par ordinateur seront nécessaires pour prendre avantage de l'espace qu'offrent les 3D-IC. Par conséquent, un défi architectural se pose ; en effet, l'architecture dont dispose l'industrie de nos jours est celle de John von Neumann dont le *x*8086 est l'un des modèles les plus populaires. Une nouvelle architecture et organisation des composantes électroniques sont donc nécessaires. La Figure 1.1 illustre l'interaction entre les différents domaines de l'industrie des circuits intégrés qui permettent la conception des architectures des systèmes multiprocesseurs sur puce.

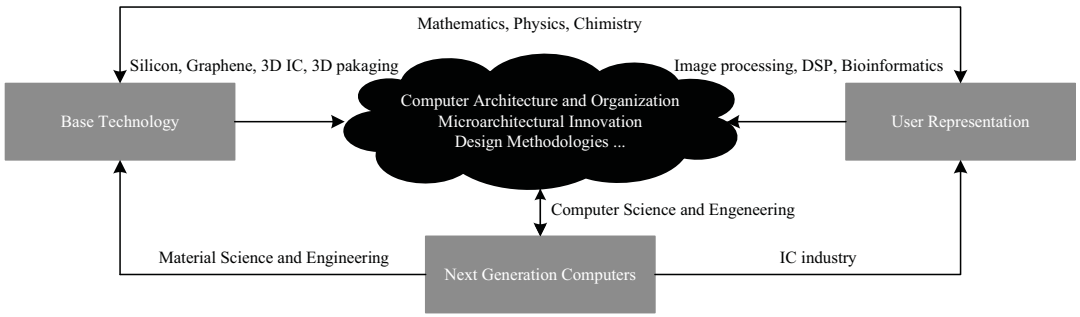

Figure 1.1 – Illustration de l'interaction entre la technologie de base, l'architecture et la conception des circuits intégrés

1.2.2 Approche hiérarchique de la conception des systèmes sur puce électronique

L'approche traditionnelle de la conception des circuits intégrés est basée sur une logique hiérarchique et séquentielle [68]. Elle a pour but de réduire la complexité à tous les niveaux de la conception et d'augmenter la fiabilité à chaque étape de cette hiérarchie. La Figure 1.1 présente une approche générale et récente du flot de la conception des systèmes OCM [112]. Suivant cette méthodologie, la conception du système est décrite en deux grands sous-ensembles : une partie matérielle et une partie logicielle. Dans cette thèse, nous nous intéressons particulièrement à la partie matérielle. Elle est composée des phases suivantes : la spécification du client ou de l'utilisateur, l'architecture du matériel, la séparation des modules, la conception des modules et des blocs dédiés, l'intégration des blocs IP (*Intellectual Properties*) des fournisseurs tiers, la simulation, la synthèse, l'émulation et le prototypage (si nécessaire sur les circuits intégrés confi-

gurables), le placement et routage, la fabrication des prototypes, la vérification et la validation, et enfin la fabrication définitive du système. Nous pouvons constater que vu cette liste qui n'est toutefois pas exhaustive, la conception d'un OCM est un travail laborieux. Signalons toutefois qu'il est difficile de définir une dichotomie exacte entre la partie matérielle et la partie logicielle du développement.

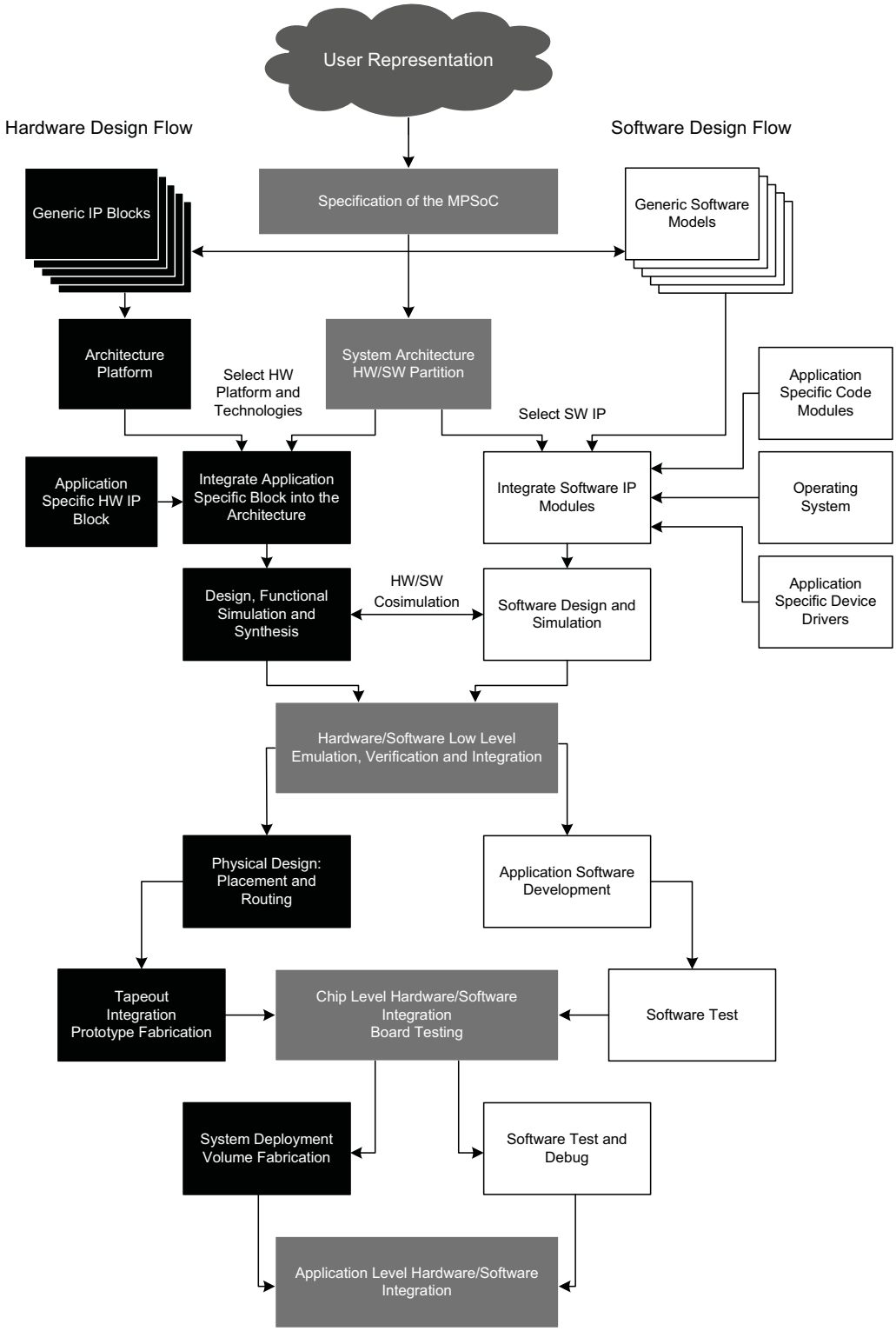

Figure 1.2 – Illustration du flot de conception d'un OCM inspirée du CAP7 [125]

Dans la partie matérielle, le but de la phase de spécification fonctionnelle permet au client ou à l'utilisateur de définir tous les comportements désirés et toutes les fonctionnalités attendues du système. Signalons que cette phase peut être une source d'incohérence et d'ambiguïté dans le fonctionnement du produit final à la fin du processus de développement. Plusieurs outils et méthodes sont proposés dans l'industrie pour résoudre ces problèmes. Une fois que la spécification est acceptée, la phase de la définition de l'architecture est entamée avec une différentiation des aspects purement matériels et des aspects logiciels : c'est ce qu'on appelle la décomposition ou la partition logicielle/matérielle (*HW/SW partition*) [10], [130]. Dans la phase de la différentiation, des fonctions peuvent être sous-traitées par des blocs fonctionnels appelés propriétés intellectuelles (IP), matériels ou fonctions logicielles achetées chez des fournisseurs tiers. Cette phase est très délicate et la différentiation doit être faite avec beaucoup de précaution, car une erreur ici peut être difficile à corriger dans la suite de la hiérarchie. Aussi, cette phase est déterminante pour la performance du système. En effet, une mauvaise décomposition entraine des interdépendances accrues entre les éléments et entraine une diminution de la capacité de transfert de données entre les unités électroniques du système. Nous en parlerons plus en détail dans les chapitres à venir.

Dans la phase de l'implémentation du système, des langages de description matérielle appelés *High-Level Description Langage* (HDL) ou des langages de haut niveau (e.g., SystemC) sont utilisés pour permettre une représentation fidèle du système. Sans perte de généralité, nous pouvons dire que les outils de conception assistée par ordinateur (*Computer Assistant Design* [CAD]) dont disposent l'industrie des circuits intégrés classiques pour la simulation, la vérification, la synthèse, l'émulation et le prototypage, sont assez conséquents et sophistiqués. Cependant, ces CAD peuvent-être non adéquats à la conception des systèmes OCM en 3D. Il est donc nécessaire que de nouveaux outils soient développés pour faciliter la spécification, l'analyse, la décomposition, la simulation et l'affectation (pour ne citer que ceux-là). Dans tous les cas, il y a un besoin de définir des formalismes adéquats pour la modélisation de ces systèmes, et développer des méthodes systématiques pour faciliter le travail des concepteurs de ces systèmes. Dans le cas spécifique des OCM, il est impératif de trouver des méthodologies qui permettent

d'automatiser les étapes de la conception et plus spécialement trouver des méthodes de programmation combinatoire pour optimiser ces étapes afin que dès la spécification, les caractères multiprocesseurs de ces systèmes soient pris en considération. D'autre part, il est important de signaler que les applications qui sont exécutées sur les OCM doivent êtres intrinsèquement parallèles pour profiter pleinement de l'aspect multifonctionnel et distribué de ces outils [126]. En effet, lorsqu'une application n'est pas assez décomposable, ou lorsque la décomposition ne permet pas de réduire conséquemment les communications entre les modules, son exécution sur un OCM diminue sa performance. Dans cette optique, il est plus facile d'utiliser des processeurs super scalaires [14]. Toutefois, les processeurs super scalaires ne sont conçus que pour l'exploitation du parallélisme au niveau des instructions. Dans la référence [170], Wall analyse le comportement de plusieurs programmes sous 375 différents angles de parallélisme. Il a trouvé que le parallélisme dans ces problèmes est faible ou très modéré. Dans des cas de simulations extrêmes où la fenêtre du *fetch* est de 2048 instructions et où le nombre d'instructions exécutées en un seul cycle est de 64, les meilleures solutions ont donné dix instructions exécutées en parallèles par cycle, ce qui donne un taux de 3,12%. Aussi, est-il prouvé que pour mieux exploiter le parallélisme dans les systèmes super scalaires, la fenêtre du *fetch* doit être de 2 à 4 instructions [81]. Par ailleurs, lorsque la fenêtre est grande, la gestion de la synchronisation cause des problèmes additionnels de performance et surtout de conception du système. La Figure 1.3 illustre les niveaux de parallélisme à chaque degré différent de granularité.

Bien que l'exploitation du parallélisme au niveau des instructions soit difficile à obtenir, Wall signale dans le même rapport d'analyse [170] que le parallélisme est assez abondant dans les applications à virgule flottante et à des niveaux de granularité plus élevés comme les boucles ou les tâches qui peuvent aller chercher des degrés de parallélisme très profonds. Ainsi, plusieurs méthodes et outils ont été développés pour exploiter de manière automatique le parallélisme dans les applications scientifiques. Bien que ces compilateurs soient efficaces, ils ne sont pas très répandus dans l'industrie des circuits intégrés. En effet, ils aident à exploiter le parallélisme au niveau logiciel. Dans cette thèse, nous nous intéressons au parallélisme au niveau des processus, et ceci, dès la phase de la

spécification et de l'architecture matérielle du système. Signalons aussi que nous nous concentrons en particulier sur les systèmes dédiés (*Application Specific System-on-Chip* [AS-SoC]) qui permettent une architecture selon les besoins des utilisateurs.

Dans cette thèse, nous cherchons à apporter quelques solutions que nous considérons efficaces en nous penchant spécifiquement sur trois problématiques majeures de la conception des systèmes OCM. La première problématique est l'organisation architecturale des unités composantes de ces systèmes afin de favoriser un meilleur approvisionnement des données aux processeurs, car l'élément critique de la performance de calcul (où d'autres paramètres comme la consommation d'énergie et la dissipation de la chaleur) est le transfert de données dans les réseaux de communication de ces systèmes, et par conséquent, leur organisation. Actuellement, les réseaux des OCM en 3D et leurs organisations sont dans un état embryonnaire surtout au niveau architectural. La deuxième problématique est de trouver la meilleure façon de décomposer les spécifications dans le but de créer une architecture multiprocesseur efficace et de déterminer de manière optimale le nombre d'unités afin d'accroître la performance du système. Il est évident que l'idée d'une spécification parallèle n'est pas nouvelle en soit. Cependant, le problème est les outils de spécifications parallèles sont souvent utilisés pour séparer la partie matérielle de la partie logicielle d'un système. Aussi, les spécifications parallèles sont faites le plus souvent sur des architectures déjà existantes, ce qui consiste à forcer une application sur une structure. Cette option peut entrainer des pertes de performance. D'autres problèmes que ces outils ou méthodes de spécification présentent sont souvent liés à leur complexité et au fait qu'ils sont souvent dédiés à des classes de problèmes limitées et non publiques comme le HardwareC [73]. La troisième problématique est celle d'une organisation optimale des blocs décomposés pour mieux réduire les flots de données dans le système. Plusieurs travaux ont été faits dans le domaine de l'organisation des composantes d'un circuit intégré [100]. Celui que nous utiliserons ici, c'est le problème de l'affectation quadratique [100]. Nous nous intéressons plus spécifiquement à un modèle d'affectation à topologies variables où le choix de la technologie est pris en compte. Aussi, nous nous penchons sur l'affectation optimale des valeurs des arcs de communication entre les nœuds du réseau afin de tirer un meilleur profit du parallélisme.

Notons que la technologie de base que nous visons ici, à savoir les systèmes OCM sur les circuits intégrés 3D, n'est pas encore bien maîtrisée. Ainsi, les méthodologies, les outils d'aide à la conception et les architectures et organisation de systèmes sont des avenues de recherche très actives.

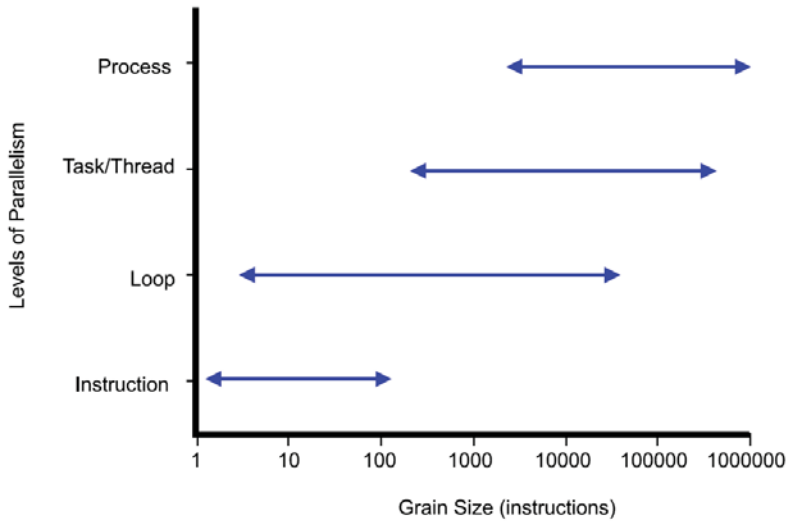

Figure 1.3 – Niveau de parallélisme dans une application [103]

1.3 Contribution de la thèse

Nous avons présenté dans la section précédente les problématiques sur lesquelles nous nous penchons dans cette thèse. Trois en ont été identifiées : (1) la recherche de performance dans un OCM en 3D par une nouvelle organisation des composantes de système ; (2) la recherche de performance par une méthodologie de décomposition des spécifications des systèmes ; et (3) la recherche de performance par une méthodologie d'affectation des éléments décomposés en (2) et des valeurs aux arcs de communication entre ces éléments. Nous présentons ainsi dans cette thèse trois contributions dont chacune s'adresse à l'un des points cités ci-dessus, comme illustré dans la Figure 1.4.

La première contribution est consacrée au problème de l'organisation des unités des systèmes. Nous explorons une nouvelle, mais simple, organisation architecturale qui est

basée sur la structure cristalline de chlorure de sodium (*NaCl*). Le *NaCl* est composé de deux éléments : le sodium (Na^+) et le chlore (Cl^-) que nous pouvons associer respectivement aux processeurs et aux unités de mémorisation. Nous nous intéressons aussi à l'agencement de ces éléments. Lorsqu'on analyse une structure du *NaCl* à l'aide d'un microscope, on se rend compte qu'elle est constituée d'une superposition régulière de couches, et que chaque élément sur une couche quelconque partage son électron libre avec ses voisins de la même couche et les voisins des couches juxtaposées à la sienne. Dans ce contexte, le chlore partage avec ses huit voisins sodiums son électron libre. De façon analogue, nous proposons une organisation des processeurs et de mémoire en une architecture que nous appelons *Isometric on-Chip Multiprocessor Architecture* (ICMA). Dans notre cas, chaque mémoire est connectée à ses six voisins processeurs et chacun de ces derniers est connecté à son tour à ses six voisins de mémoires. Chaque unité est ainsi connecté à quatre éléments sur le même plan, à savoir nord, sud, est et ouest ; à un élément en haut et à un élément en bas. Cette architecture exploite certains aspects de performance comme la structure régulière et l'exploitation de la localité. La structure régulière de l'architecture permet de réduire la distance des arcs de connexion dans tout le système et l'organisation des processeurs et des mémoires selon la structure de *NaCl* permet d'exploiter la localité dans la distribution des données. Dans son livre "*Introduction to Parallel Processing*" [138], Parhami a rapporté que les architectures des systèmes multiprocesseurs ou des superordinateurs se construiront comme des jeux de lego, où les pièces seront des « *Commodity Nodes* ». C'est spécifiquement cette prédiction que nous tentons de réaliser dans cette contribution.

La deuxième contribution est consacrée à l'analyse de la localité. Cette analyse est faite par la décomposition de l'application qui est exécutée sur l'architecture proposée dans la première contribution, ou la décomposition d'une spécification en vue de la création d'une telle architecture. Le but étant de créer les « *Commodity Nodes* » au niveau architectural. Nous nous intéressons en particulier à la modélisation matricielle de l'application en vue de son implantation sur un réseau de processeurs représentant le système sur puce. L'objectif est de trouver le nombre optimal de nœuds du réseau et maximiser la performance en exploitant la localité dans les transferts de données. Nous proposons une

méthode de regroupement qui transforme l'application ou la spécification fonctionnelle en un ensemble de cellules de telle sorte que les transferts intracellulaires sont maximisés et les transferts intercellulaires minimisés. L'application est considérée comme une matrice d'incidence $A = M \times N$, où M est le nombre de processus dans l'application et N le nombre de données. Chaque élément a_{ij} représente le taux de transfert de données entre le processus i et la donnée j. Pour cette décomposition, nous nous sommes basés sur les problèmes de formation de cellules (*Cell Formation Problem* [CFP]) de la technologie des groupes (*Group Technology* [GT]), qui manipule des entiers plutôt que des éléments binaires, pour créer notre modèle [182], [137]. Aussi, avons-nous introduit une notion d'équilibre de charge entre les cellules. Ceci permet de jouer sur les éléments qui sont hors des cellules, appelés *éléments exceptionnels*, et avec la taille de ces cellules. Nous avons utilisé des approches méta heuristiques pour optimiser notre méthode et pour produire des résultats expérimentaux *quasi* optimaux. Lorsque l'application est décomposée pour un système existant, le nombre de cellules doit être inférieur ou égal au nombre de nœuds du système. Dans le cas où le nombre de cellules est inférieur au nombre de nœud, des mécanismes de gestion énergétiques appelés « *Energy Controller* [EC] » mettront les nœuds non utilisés en veille.

Dans la troisième contribution, nous nous penchons sur l'assignation de ces cellules sur les nœuds du réseau en nous basant sur une méthode qui minimise les distances que parcourent les flots extracellulaires représentés par les éléments exceptionnels. En effet, l'affectation des processeurs et de mémoires (taches et données) dans une architecture parallèle ou distribuée a toujours été une phase critique dans la conception de celle-ci. Même s'il y a beaucoup de méthodes proposées dans la littérature pour résoudre le problème d'affectation, la nouvelle technologie en trois dimensions des circuits intégrés impose de nouveaux défis aux ingénieurs pour trouver de nouvelles méthodes d'affectation, particulièrement lorsque leurs objectifs impliquent plusieurs critères. De plus, le gain de performance à la suite d'une bonne affectation peut être compromis par les performances des arcs de communication entre les cellules et leur synchronisation. Il est alors impératif de trouver une méthode pour affecter les valeurs appropriées à ces arcs de communication. Dans cette contribution, nous proposons une méthode d'affectation

en deux étapes. La première étape appelée " *Assignment of Groups and Families to 3D Network Nodes* ", basée sur un réseau en 3D qui contient la structure de base de l'architecture, consiste à affecter les groupes de processus et les familles de données aux nœuds du réseau 3D selon la décomposition de la deuxième contribution. Dans un premier temps, nous proposons une approche de modélisation des flots de données entre les cellules. Ceci permet de créer plusieurs topologies et de savoir, dès la phase de l'architecture du système, laquelle de ces méthodes est la plus optimale pour l'application traitée. Ensuite, nous utilisons le problème d'affectation quadratique (QAP) pour l'assignation proprement dite. Cette affectation est basée sur la recherche tabu (TS) [70] proposée par Eric Taillard [161]. Dans la deuxième étape, nous nous attaquons au problème d'affectation des valeurs aux arcs. En effet, les caractéristiques des arcs dans les systèmes sur puces dépendent fortement des applications que ces systèmes traitent. Dans le comportement de systèmes sur puces, le choix des arcs est optimal pour la conception d'un système performant. Ainsi, une méthode de conception des arcs est nécessaire dès la phase de la spécification et de l'architecture du système. L'analyse de performance des arcs doit être incluse dans celle de tout le système, car il a été prouvé par Oswens et al. [136] qu'une énorme quantité d'énergie est utilisée pour transférer des données dans le réseau. La plupart des travaux qui ont été proposés dans la littérature pour améliorer ces transferts se focalisent sur l'augmentation et la gestion des mémoires tampons entre les nœuds [84], [129]. Nous proposons dans cette contribution une approche utilisant les problèmes d'affectation quadratique et la perturbation comme solution à ce problème.

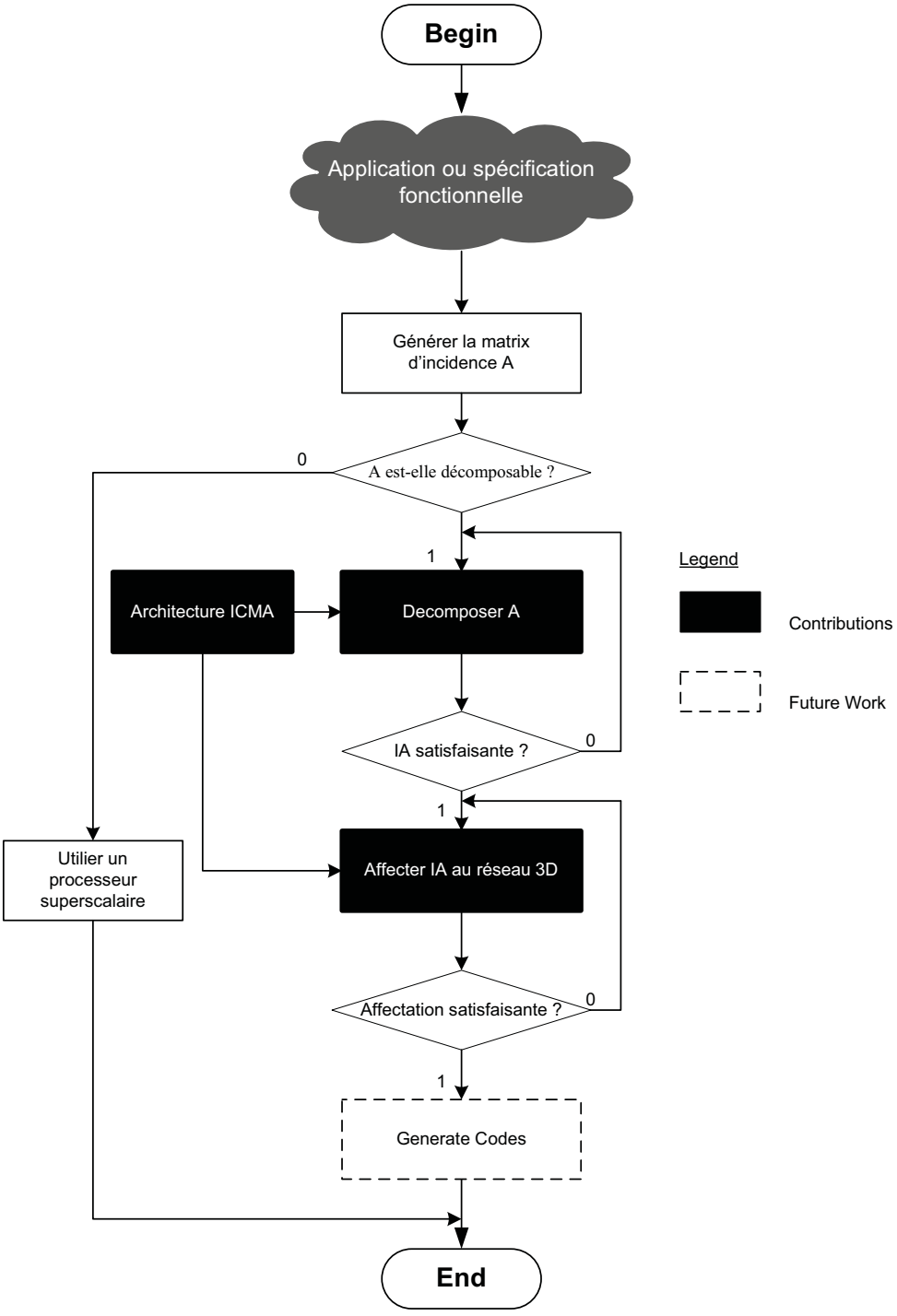

Figure 1.4 – Diagramme et organisation des contributions de la thèse

1.4 Organisation de la thèse

Dans le chapitre suivant, nous présentons la description et une revue de littérature de l'architecture des systèmes sur puce électronique. Nous présentons aussi une revue de littérature des sujets abordés dans cette thèse, c'est-à-dire, les architectures des systèmes 3D sur puce, les problèmes de formation de cellule, utilisé pour la décomposition d'une application en sous-application et finalement l'affectation de blocs de fonctions et de données à des nœuds d'un réseau au préalable prédéfini. Le chapitre trois présente notre première contribution qui est une proposition d'une architecture et son organisation. Le chapitre quatre présente notre deuxième contribution qui traite de la décomposition d'une application ayant pour architecture cible celle présentée dans le chapitre trois. Le chapitre cinq propose une méthode d'affectation des sous applications du chapitre quatre aux nœuds de l'architecture cible et l'affectation de capacités transmises qu'on peut assigner aux arcs de communication entre les nœuds du réseau pour une meilleur performance.

CHAPITRE 2

REVUE DE LITTÉRATURE

Bien que liés, les problèmes abordés dans cette thèse sont généralement traités séparément dans la littérature. Dans ce chapitre, nous tenterons de donner un bref aperçu de ce qui a est proposé sur l'architecture et l'organisation des systèmes multiprocesseur sur puce électronique (*On-Chip Multiprocesseur* [OCM]), sur la décomposition des applications ou des spécifications fonctionnelles, et ensuite sur l'affectation des blocs aux nœuds dans un réseau.

2.1 Les systèmes multiprocesseurs sur puce électronique

La réduction de la taille des transistors et la complexité fonctionnelle des circuits intégrés ont mis beaucoup de stress sur l'industrie de la microélectronique. Depuis plus d'une décennie, le paradigme des systèmes sur puce (*System-on-Chip, [SoC]*), et en l'occurrence des OCM, est utilisé comme norme dans les industries électroniques, des télécommunications, du multimédia, des réseaux et dans bien d'autres domaines [21]. Ce sont considérés actuellement comme l'un des défis les plus importants pour l'industrie des circuits intégrés et pour les centres de recherches académiques. Les OCM sont des circuits appelés (*Very Large-Scale Integration* [VLSI] ou *Gigascale Integrationdes* [GSI] qui composent plusieurs processeurs à d'autres éléments matériels (mémoires, IPs) pour former un système à une application.

Leur conception est basée sur les modèles, les techniques et les outils des réseaux classiques. Ils sont considérés comme des micro-réseaux sur puce [21]. La communication entre les composantes de ces systèmes est implémentée par un réseau appelé réseau sur puce (*Network-on-Chip* [NoC]). Selon Benini, ce réseau doit assurer les exigences d'une qualité de service (*Quality of Service* [QoS]) [21]. L'architecture spécifie les interconnexions des composantes et la topologie du NoC. Comme dans les réseaux classiques, les protocoles indiquent comment les unités doivent communiquer entre elles

[171]. Dans cette section, nous présentons dans un premier temps un survol sur les composantes des systèmes OCM et ensuite, leur organisation architectural.

2.1.1 Composantes d'un système multiprocesseur sur puce

Les systèmes OCM sont souvent composés de plusieurs circuits intégrés de différentes sortes et de différentes fonctionnalités. Ces circuits peuvent être regroupés en trois catégories : les processeurs, les mémoires et les liens de communication, comme l'illustre la Figure 2.1. Actuellement, plusieurs processeurs et mémoires peuvent être intégrés sur la même puce tout en restant structurellement autonomes.

2.1.1.1 Les processeurs

Vu le caractère dédié des systèmes électroniques actuels, les processeurs les plus utilisés pour les systèmes OCM sont souvent des processeurs embarqués et programmables ou des fonctionnalités intégrées sur des réseaux de portes logiques programmables (*Field Programmable Gate Array* [FPGA]). Les processeurs utilisés dans les systèmes OCM doivent être flexibles, modulables et paramétriques, afin de pouvoir les intégrer facilement dans une architecture. L'approche la plus efficace est le processeur dédié (*Application Spécification Integrated Circuit* [ASIC] ou *Digital Signal Processor* DSP). Les processeurs dédiés sont construits sur un jeu d'instructions très réduit qui ne traite que des besoins spécifiques pour des domaines d'application bien précis [87]. Toutefois, ils peuvent servir souvent à des classes d'application comme le multimédia ou le traitement d'images. Parmi ces processeurs, nous avons le TriMedia TM32 [54], [82] qui est un processeur pour les applications multimédia qui permet de transférer des données parallèles et de manière continuelle. Un autre exemple de processeur dédié est le *Micro Engine* ME du système IXP2850 de réseau WAN et LAN, qui est utilisé pour exécuter des fonctions à filins d'exécution multiples sur les paquets. Vu le nombre important de paquets dans un processeur de réseau, plusieurs ME sont utilisés pour permettre un traitement parallèle. Plusieurs autres processeurs dédiés comme le SPE [63] sont proposés par des entreprises afin de traiter un problème bien particulier dans l'architecture des systèmes OCM.

Bien qu'efficaces, les processeurs dédiés ne sont pas les plus populaires dans la conception des OCM ; le plus grand problème des ASIC ou des DSP, est qu'ils coûtent très cher en ressources et en temps à développer, ce qui augmente considérablement le coût total du système. Le processeur embarqué ARM (*Advanced RISC Machine*) [11], est le plus utilisé par les fabricants des OCM. L'avantage du ARM par rapport aux processeurs dédiés cités ci-dessus, est qu'il est générique et n'est pas physiquement fabriqué par ses fournisseurs. Ainsi, chaque processeur est conçu sur un noyau ARM au-dessus duquel des spécifications propres aux clients sont ajoutées. Nous citons comme exemple l'architecture du processeur de ST Nomadik qui contient le processeur ARM926E-JS [12] qui est basé sur le noyau ARM9.

Un autre noyau de processeur souvent utilisé dans la conception des OCM est le MIPS (Microprocessor without Interlocked Pipeline Stages). Comme le processeur ARM, le MIPS est un noyau de processeur qui permet de créer plusieurs processeurs dérivés. Bien qu'utilisé dans les ordinateurs et dans les serveurs de Silicon Graphic, on le retrouve souvent comme processeur embarqué dans des OCM. Ainsi, dans l'architecture du Nexperia [54], une variante de MIPS appelée le PR3940 a été développée pour s'occuper du système d'exploitation et pour contrôler les exécutions de certaines tâches.

Nous reconnaissons que les processeurs cités ci-dessus ne constituent pas une liste exhaustive et que plusieurs autres processeurs embarqués ont été développés et sont abondamment documentés dans la littérature.

Les processeurs softcores sont proposés par des fabricants de FPGA. Ces processeurs sont spécifiquement dédiés et synthétisables uniquement sur les circuits programmables de leurs fabricants. C'est le cas du Nios de Altera [6] et du MicroBlaze de Xilinx [178]. Plusieurs applications ont utilisé les processeurs MicroBlaze de Xilinx comme dans le cas de [157], [133] et [86], et des processeurs Nios pour le prototypage OCM à moindre coût. Quelques travaux sont à souligner dans le cas de l'intégration de Nios dans une application OCM comme dans les références [146], [183] et [55].

En ce qui concerne les processeurs généraux (CPU), dont les architectures les plus populaires sont basées sur le 80*x*86, leur conception permet de les utiliser pour la résolution de problèmes plus généraux. Ils sont souvent moins chers parce qu'ils sont faits

pour une utilisation générale et une production de masse. Cependant, leurs tailles sont généralement importantes et ils sont souvent lents pour être intégrés dans des systèmes de haute performance. De nos jours, seule le PowerPC, qui est en réalité une dérivée du MIPS, a été embarqué sur le FPGA Virtex-5 FX70T de Xilinx [177], et est utilisé pour les traitements d'ordre général.

En conclusion, nous pouvons dire qu'il y a une diversité de processeurs qui peuvent être utilisés pour la conception des OCM et que le choix ne dépend que des besoins de l'architecture ; ce qui fait qu'ils ne constituent pas un problème de performance.

2.1.1.2 Les mémoires

Contrairement aux processeurs, le choix des mémoires embarquées est limité, vu le nombre réel de sortes de mémoires disponibles dans l'industrie. Toutefois, leur organisation, leur hiérarchie et les protocoles de routages sont importants pour la conception de systèmes performants, et sont souvent différents d'un système à l'autre. Nous retrouvons par exemple le cas du TRIPS [30] ou du Tilera Tile64 [173] dont les mémoires sont organisées en deux niveaux de cache L1 et L2, et utilisent un contrôleur matériel alors que le Teraflops de Intel (80 processeurs) et le IBM Cyclops-64 utilisent des contrôleurs logiciels [48] et [77].

L'un des problèmes que les mémoires embarquées rencontrent est leur taille. Elles doivent être assez petites à cause de l'espace disponible et assez large pour favoriser le facteur de localité entre les données et les fonctions. Pour résoudre ce problème, des solutions comme la conception de mémoire sans cache ou l'utilisation des mémoires statiques à la place des mémoires dynamiques (plus complexe) [107] sont proposées. Ainsi, pour réduire la taille des processeurs, la consommation d'énergie et réduire la chaleur, les auteurs de l'article [17] ont proposé une alternative à la mémoire cache appelée *Scratchpad Memory*, qui est une circuiterie composée d'un décodeur et d'une table mémoire. Cette circuiterie occupe une partie de l'espace d'adressage de la mémoire principale et crée une association entre le reste de l'espace mémoire.

Dans [179], Xu *et al.* ont fait une étude de différentes organisations hiérarchiques des mémoires dans un système OCM. Ils ont démontré premièrement que l'utilisation

d'une hiérarchie de la mémoire cache L1 et L2 dans les processeurs peut augmenter la performance de calcul d'une application dont la fréquence d'accès à la mémoire est très élevée. Cependant, la différence entre la bande passante d'une mémoire externe au OCM et sa mémoire interne peut entrainer une sérieuse perte de performance lorsque plusieurs filins sont actifs. Deuxièmement, ils ont montré que les congestions dans le OCM sont causées par des accès importants des processeurs aux mémoires. Ainsi, il est nécessaire de calculer adéquatement le chemin de données qui connecte chaque processeur du système à la mémoire principale. Ils pensent par ailleurs qu'il n'est pas adéquat de privilégier une optimisation de la communication entre les composantes du système sur la puce au dépend de certaines paramètres comme la chaleur et la consommation de l'énergie. En théorie, il est démontré que chaque application a un modèle de communication unique [13]. Ainsi, Dally et Towles pensent toutefois que la performance d'un système sur puce dépend de son accès aux données, de la hiérarchie des mémoires et surtout du réseau d'interconnexion sur la puce [46].

Le problème d'accès aux mémoires est malheureusement le goulot d'étranglement. La recherche d'équilibre entre la hiérarchie de la mémoire, le chemin des données, l'accès à la mémoire externe et la recherche de chemins critiques est devenue un problème d'optimisation combinatoire dont la complexité dépend de l'application traitée.

2.1.1.3 Les liens de communication

Les liens entre les unités sont constitués d'une interconnexion ou d'un câblage de fils parallèles [154], [2] qui permettent de distribuer l'horloge, le courant électrique (Vcc, GND, Reset), la terre et d'autres signaux comme les données (DATA) et les commandes (CMD) entre les processeurs et les mémoires. Le besoin d'une transmission rapide des données à travers le système et d'une qualité des données et de services transmis exigent comme dans le cas des processeurs et des mémoires que les liens soient efficaces et performants. En effet, avec la diminution de la taille des transistors, les liens de communications se sont rapprochés et de nouveaux problèmes se sont posés. Parmi ces problèmes, nous avons la non fiabilité des données, les bruits électriques et les interférences. Par ailleurs, les spécifications physiques des lignes utilisées jusqu'à présent dans l'industrie

de semi-conducteurs limitent grandement leurs vitesses de transmission.

Deux catégories de liens de communications sont utilisées dans la conception des OCM. Les liens parallèles et les liens sériels. Les liens les plus utilisés sont les liens parallèles car ils permettent naturellement de transmettre des mots de données à chaque cycle d'horloge. Aussi, ils contribuent à la dissipation de la chaleur mais posent d'autres problèmes comme l'espace occupée, les bruits électriques, les erreurs dues à l'électro migration et à l'interférence. D'autre part, les liens sériels deviennent de plus en plus populaires parce qu'ils permettent de corriger certains problèmes comme la réduction de l'espace utilisé, l'intégrité des données et la diminution des erreurs, mais en posent d'autres comme la chaleur et les délais supplémentaires [49]. Nous remarquons ainsi que dans les OCM, les liens constituent les maillons faibles de la performance. Pour corriger ce problème, plusieurs travaux ont été proposés.

Dans [162], Tamhankar *et al.* ont proposé une approche agressive appelé Terror qui tolère les fautes temporelles causées par les aléas physiques afin d'augmenter la performance de la communication. La méthodologie réduit le nombre de mémoires tampons entre les nœuds NoC et propose un protocole qui consiste à corriger les erreurs par Go-Back N présenté dans la référence [171]. D'autres travaux se sont concentrés sur la correction d'erreur pour augmenter la performance des liens de communications entre les transmetteurs de données (e.g., [158]).

Pour corriger les problèmes que posent les transferts de données parallèles, Kangmin *et al.* ont présenté un outil qui propose un transfert en série [98] basé sur une méthode d'encodage de la donnée entre les nœuds. Leur but étant de minimiser le nombre de transitions sur une ligne de transmission sérielle en utilisant les corrélations entre les mots des données successives. Cette corrélation minimise le nombre de '1' logiques et augmente le nombre de '0'. Ce qui réduit l'énergie consommée et la chaleur dissipée.

Une autre approche pour optimiser l'utilisation des liens de communication sur les puces est d'utiliser des communications asynchrones. Dans [164], Teifel et Manohar ont présenté une architecture matérielle des émetteurs et des récepteurs de données sans horloge, ceci afin d'alléger les nœuds des générateurs d'horloges qui sont considérés comme coûteux. Le mode de transmission des données dans leur solution est sériel.

Toutefois, des multiplexeurs du côté des émetteurs et des démultiplexeurs du côté des récepteurs sont utilisés pour assembler et désassembler les données parallèles qui viennent des nœuds. Contrairement aux transmetteurs de données qui utilisent plusieurs cycles d'horloge pour assembler ou désassembler les données, leur méthode utilise un lien à trois fils avec une architecture basée sur une machine à états finis (*Finite-State Machine* [FSM]) à trois états et sur un protocole d'anneaux à jeton. Aussi, l'architecture proposée permet-elle de faire une synchronisation dynamique entre le taux des bits envoyés par l'émetteur et celui des bits échantillonnés par le récepteur. Signalons que dans la littérature, plusieurs autres travaux qui traitent des transmissions asynchrones dans les NoC ont été proposés, e.g. [50].

Plusieurs autres travaux ont été proposés pour la conception de liens de communication performants comme [164], [118], [50], [95], [144] et [158]. Toutefois, nous n'avons pas trouvé dans la littérature une approche qui propose une conception de liens de communication dédiés. Nous pensons que pour chaque système à concevoir, des liens dédiés entre les nœuds selon la particularité de chaque arc peuvent augmenter la performance du système tout entier.

2.1.2 Organisation des systèmes multiprocesseurs sur puce

Les composantes présentées ci-dessus sont souvent organisées de manière à augmenter la performance de traitement de l'ensemble du système. L'organisation des systèmes OCM n'est pas actuellement quelque chose de formelle. La plupart des systèmes sont organisés de manière *ad hoc*. Toutefois, comme dans les réseaux classiques, plusieurs structures régulières sont utilisées. La plus populaire des structures est l'organisation maillé en deux dimensions. La Figure 2.1 en est une illustration.

Dans [133], nous avons présenté plusieurs structures de systèmes OCM et nous avons remarqué que les structures régulières ne sont pas très populaires dans les architectures commerciales, mais qu'elles sont très présentes dans les OCM académiques. Selon Wolf, il est rare que l'architecture et l'organisation d'un système soient toutes nouvelles [175]. Il dit ce qui suit : " L'histoire de la conception de systèmes informatiques montre que les techniques développées pour une technologie sont souvent utilisées dans le déve-

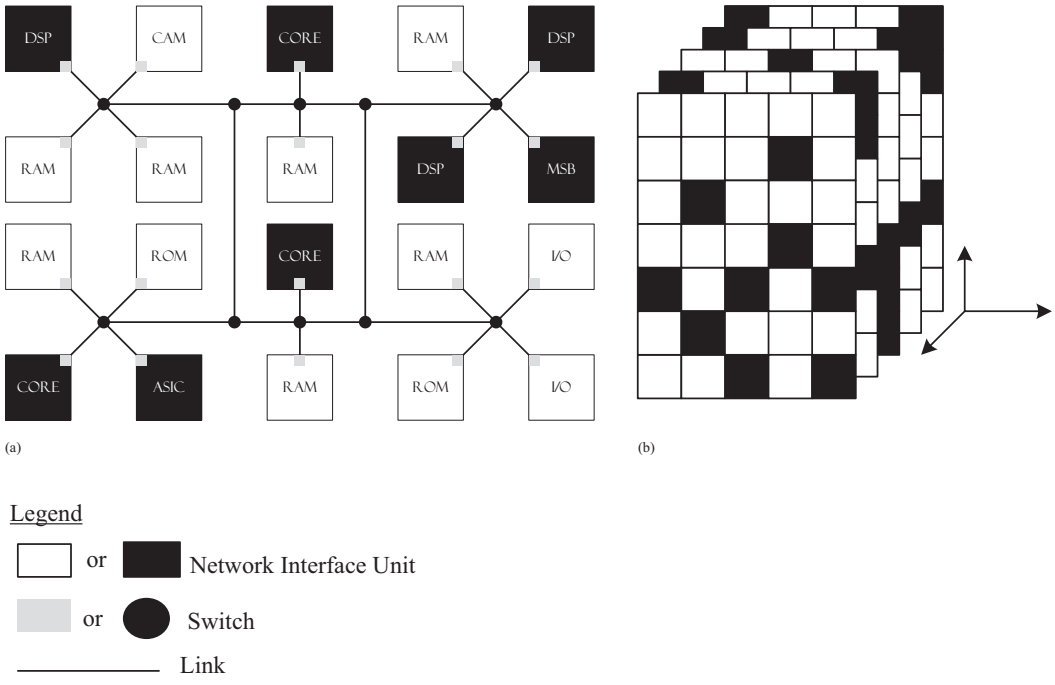

Figure 2.1 – Illustration d'un système OCM : (a) sur deux dimensions et (b) sur trois dimensions

loppement de nouvelles technologies : des mainframes aux mini-ordinateurs, des mini-ordinateurs aux microprocesseurs, etc. Et le traitement en parallèle a une riche histoire qui remonte aux premiers jours de l'informatique ". Ainsi, la plupart des architectures peuvent êtres considérées comme une composition du modèle de von Neumann où les composantes communiquent entre eux à travers un ou plusieurs bus, ou à travers un réseau d'interconnexion. Dans les deux cas, un arbitre ou un protocole de communication sont utilisés pour gérer les flots de données.

Plusieurs organisations architecturales sont proposées dans la littérature. Ces organisations peuvent être classifiées en deux groupes selon la diversité des processeurs : les architectures homogènes [76], [181], [60], [155] et les architectures hétérogènes [54], [139].

Les structures homogènes sont composées de deux ou plusieurs processeurs identiques en fonctionnalité et en structure, et sont souvent utilisées pour calculer des problèmes très récurrents (e.g. la multiplication matricielle ou l'évaluation polynomiale).

La Figure 2.2 illustre une organisation homogène représentant un tableau systolique appelé *Band Matrix Multiplication Systolic Array* [BMMSA]. L'organisation dans cette figure est un tableau systolique carré (2D) avec le chemin des données des opérandes représentées par les registres *A*, *B* et *C*. La partie a) de la figure représente les structures élémentaires des composantes du BMMSA.

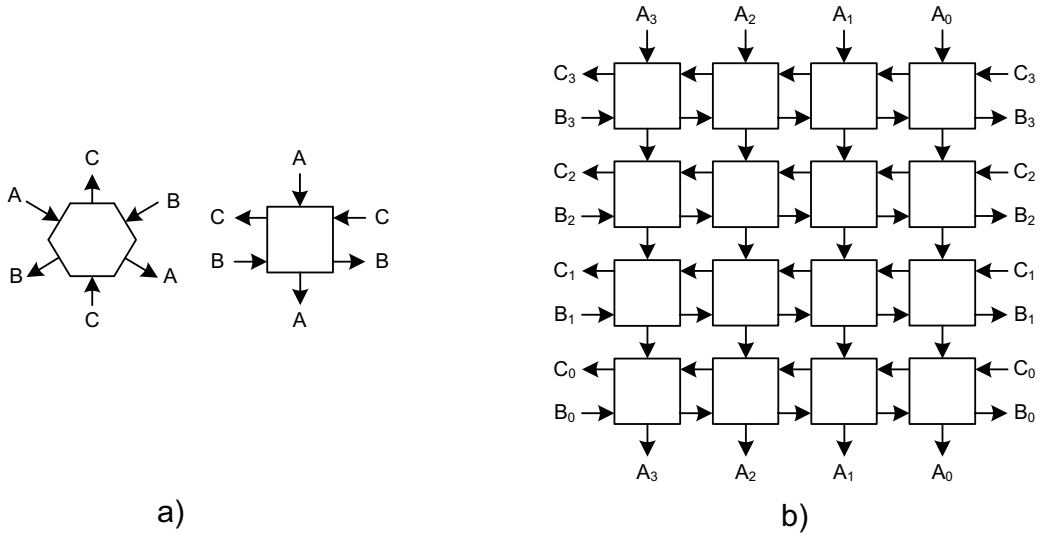

Figure 2.2 – Architecture BMMSA de tableau systolique [181]

Les structures hétérogènes sont quant à elle composées de processeurs divers en structures et en fonctionnalités. La Figure 2.3 présentée ci-dessous est un diagramme fonctionnel de NEXPERIA [54] qui illustre un système hétérogène. Cette structure a deux processeurs différents et plusieurs autres composantes de calcul. Comme nous pouvons le remarquer, c'est une architecture qui est construite autours de trois bus : MIPS PI Bus qui permet de connecter NEXPERIA aux périphériques systèmes, le TM32 PI Bus qui permet de connecter les unités multimédia à NEXPERIA et le bus de 64-bit qui permet d'accéder à la mémoire principale. Dans cette illustration, nous remarquons que la plupart des OCM sont organisés autours des bus de manière *ad hoc* selon les besoins du fabricant. Rappelons comme cela est abondamment montré dans la littérature que les bus constituent un goulot d'étrangement pour la performance de calcul.

Des structures plus génériques sont proposées dans la littérature pour des systèmes

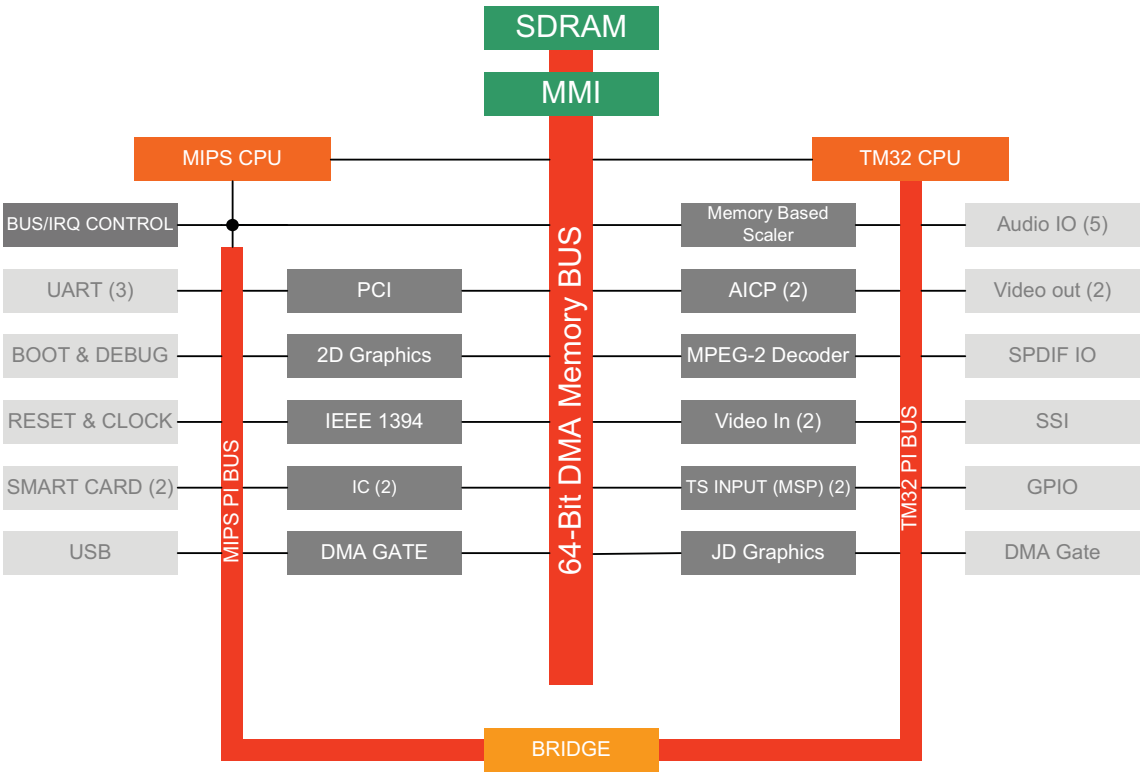

Figure 2.3 – Diagramme fonctionnel de NEXPERIA [54]

OCM hybrides. Parmi elles, nous pouvons citer l'architecture HPAM (*Hierarchic Processor-and-Memory*) proposée par Miled *et al.* [23]. Toute la structure est une superposition de plusieurs PAM (Processor and Memory). Chaque PAM est composé de processeurs et de mémoires identiques respectivement (structure homogène). Ainsi, le HPAM peut-être considéré comme une structure hétérogène de plusieurs couches homogènes. Chaque PAM est séparé de son voisin par une interface. La classification des couches PAM dans la hiérarchie du HPAM est basée sur la vitesse de leurs processeurs. Ainsi, de la couche 1 à n, le nombre de processeurs augmente, la vitesse des processeurs décroit et la taille des mémoires aussi augmente. Ce modèle étant plus générique, nous l'avons utilisé dans nos travaux pour démontrer la performance de l'architecture proposée dans cette thèse.

Les réseaux sur puce (NoC) sont considérés comme la solution naturelle pour l'organisation de OCM [90]. Kumar *et al.* ont défini les réseaux sur puce comme un in-

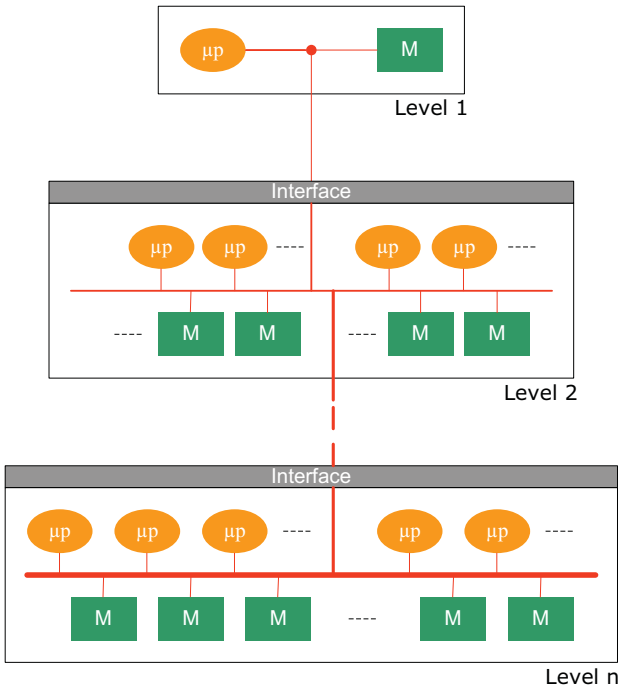

Figure 2.4 – Diagramme structurel de l'architecture HPAM [23]

contournable pour la communication sur puce organisée en trois couches : la couche physique, la couche liaison de donnée et la couche réseau [102]. Plusieurs travaux ont été consacrés à l'efficacité des réseaux sur puce par rapport à certaines paramètres de performance. Parmi ces travaux, nous retrouvons [22], [176] et [107]. Comme dans le cas des organisations citées précédemment, parmi les facteurs les plus importants de la conception d'un réseau sur puce on distingue son organisation et sa topologie. Plusieurs topologies régulières ont été proposées dans la littérature [92], [184] et [79]. Parmi ces topologies, il y a le réseau maillé, le réseau toré (Torus Network), le réseau " Fat Tree ", etc. Henkel *et al.* ont fait une étude comparative de la topologie régulière et de la topologie dédiée (*Application Specific Network on Chip* [AS-NoC]), et ils montrent que les topologies dédiées sont des topologies *ad hoc* basées sur la spécification de chaque système [80]. Les topologies dédiées sont ainsi plus efficaces que les topologies régulières et elles occupent moins d'espace. Ainsi, il est important de trouver une solution qui com-

bine l'utilisation d'une topologie régulière et les avantages d'une topologie dédiée. Par exemple, concevoir des routeurs adaptés aux besoins de chaque lien qui connecte deux composantes, optimiser les poids des composantes pour réduire l'espace occupé par les fonctions, et modéliser les données transférées ou des codes afin d'optimiser les flots de données dans le réseau.

Plusieurs techniques de routage des données sont proposées dans la littérature pour les réseaux sur puce [72], [101] et [45]. Nous ne les citons pas tous ici, mais nous faisons mention du routage " trou de ver (TV) " (*Wormhole Routing*) présenté dans [44]. Il est considéré comme le plus approprié pour réduire la taille des mémoires tampons dans les nœuds et ainsi la taille totale du système. La première utilisation de ce routage a été dans une architecture hypercube. Comme dans tous les réseaux sur puce, le nombre de nœuds et de connexions est fixé et déterminé à l'avance. Les paquets sont envoyés dans le réseau avec une adresse de petite taille afin de créer un chemin pour les transmetteurs avant même que le message ne soit envoyé. Lorsque le message est trop long, il est découpé en fragments de messages de très petites tailles appelés " flits ". Dans [85], Hu et Marculescu montrent les avantages de l'algorithme de routage TV par rapport aux autres algorithmes de routage dits adaptatifs. L'avantage de l'algorithme TV repose sur le fait que la plupart des réseaux sur puce sont conçus pour des applications spécifiques et pour cela, les flots de données entre les transmetteurs du réseau, qui varient rarement, peuvent êtres calculés à l'avance, d'où l'importance du choix d'un chemin dédié. Ni et McKinley considèrent cet algorithme comme une approche déterministe [128]. Contrairement au routage TV, l'approche adaptative basée sur " sauvegarder et envoyer " (*Save and Forward*) nécessite plusieurs ressources pour stocker et reconstruire les messages [69], et surtout utilise des chemins différents à chaque transfert entre deux nœuds.

Bien que les OCM soient un paradigme pour la conception des systèmes microélectroniques, nous pouvons, après analyse, dire que les différentes topologies, l'organisation des composantes, les protocoles de transferts de données, les hiérarchies dans la conception, les méthodes d'optimisation des paramètres comme l'espace, l'énergie, etc., ont permis de produire des systèmes performants.

Cependant, malgré toutes ces techniques, les réseaux sur puce ont malheureusement

les mêmes problèmes que les réseaux classiques dont les suivants :

- La congestion est un problème sérieux que nous devons encore étudier, car il peut arriver que plusieurs processeurs connectés au même routeur soient plus rapides que ce dernier. Ce qui conduit à une congestion et ainsi à la dégradation de la performance.
- La réduction de la taille des transistors des transistors et la densité des circuits intégrés de plus en plus croissantes entrainent des problèmes d'intégrité de données et de la qualité de service.
- Il y a encore très peu d'indications sur la manière dont les mémoires sont organisées et distribuées par rapport aux autres composantes des OCM.
- Même s'il est prouvé que l'approche réseau sur puce est plus efficace que l'approche de bus, il y a encore des besoins importants dans la fabrication des routeurs efficaces.

Dans cette thèse, nous proposons des solutions analytiques à certains de ces problèmes. Nous pensons par exemple que :

- La congestion peut être réduite si l'accès aux données est mieux planifié dès la phase de l'architecture et que l'on construit des routeurs dédiés entre chaque pair de nœuds.
- Si les données à transférer dans le réseau entre des nœuds non-voisins sont réduites à leur plus strict minimum, alors des bus sériels peuvent être utilisés comme liens de communication. Une solution peut alors être proposée au problème de congestion et celui de l'intégrité des données.

2.2 Décomposition des applications ou des spécifications fonctionnelles d'un système multiprocesseur sur puce

La phase de spécification fonctionnelle est considérée comme une phase cruciale dans la conception d'un circuit intégré. En effet, une mauvaise spécification entraine

toujours un système fonctionnellement non cohérant. Dans la conception des OCM, la spécification des besoins ou des fonctions, souvent complexes, doivent être suivie d'une décomposition (encore appelé partition) selon certains critères de similarité. La décomposition consiste alors à décomposer cette spécification ou application qui est difficile ou impossible à implémenter autrement, en un ensemble de sous spécifications ou de sous applications plus faciles à traiter.

Nous donnerons plus de détails sur la littérature de la décomposition des applications dans le chapitre 4. Toutefois, nous mentionnons ici que la technique de décomposition la plus utilisée dans l'industrie des circuits intégrés est la conception conjointe matérielle logicielle (*Hardware/Software Codesign* [HSC]) [73]. Depuis plusieurs années, le HSC est considéré comme l'un des facteurs les plus importants du développement des OCM [171]. Deux tendances sont suivies dans l'industrie : (1) une tendance matérielle qui consiste à écrire toute la spécification comme un modèle matériel et ainsi détermine les fonctions appropriées qui gagneront à être en logiciel, et (2) une deuxième tendance logicielle qui fait le contraire. Plusieurs travaux sont consacrés à la génération automatique de codes matériels et logiciels à partir de spécifications de systèmes selon les critères de réutilisabilité de code et des IP [15]. Les codes logiciels peuvent être compilés pour différentes plateformes alors que les codes matériels sont synthétisés pour des technologies appropriées. Il existe dans la littérature plusieurs outils qui sont capables de faire des décompositions ou des partitions sans l'intervention de l'homme. Cependant, ces générateurs de codes ne tiennent pas souvent compte de certaines particularités fonctionnelles et structurelles des applications traitées ou des spécifications du OCM.

Dans [120], une solution est proposée pour permettre des améliorations de la décomposition par raffinages successifs avec l'intervention des concepteurs, ce qui permet aux ingénieurs de corriger les erreurs et les limitations des outils. Cette approche est cependant risquée, car les concepteurs peuvent introduire de nouvelles erreurs à chaque intervention ou tout simplement, changer la spécification fonctionnelle du système.

Sans perte de généralité, nous pouvons dire que le fait que les outils des HSC soient orientés vers une décomposition logicielle et matérielle peut entrainer une perte de performance des systèmes. Pour relever les défis de la réutilisabilité, de la portabilité et de

la performance, de nouvelles méthodes de décomposition sont nécessaires. Dans cette thèse, nous proposons une méthode de décomposition basée sur la formation des cellules (*Cell Formation Problem* [CFP]). Le CFP est une approche de regroupement effective de la technologie de groupe utilisée dans l'industrie de la manufacture pour rassembler dans une même cellule de traitement les machines et les unités qu'elles traitent. Nous pensons que les analogies directes de machine/processeur et d'unité/donnée peuvent se faire dans notre cas.

2.3 Problème d'affectation dans les systèmes sur puce

Le problème d'affectation, associé au problème de décomposition, est très fréquent dans la conception des OCM. Par définition, le problème d'affectation consiste à placer des composantes synthétisées sur une puce électronique tout en respectant des critères de non chevauchement et des contraintes d'optimisation. Généralement, l'affectation comme la partition matérielle est faite après la synthèse en trois étapes : l'affectation globale qui permet de définir *grosso modo* quelles espaces les composantes occupent sur la puce, la légalisation qui permet de résoudre les problèmes de chevauchement des composantes et des portes logiques sont réglés, et enfin, le raffinement qui permet de détailler l'affectation globale.

Durant ces dernières années, plusieurs travaux ont été consacrés aux problèmes d'affectations dans la conception des circuits intégrés. Cependant, des études récentes ont montré qu'avec la diminution de la taille des transistors et l'accroissement de la densité fonctionnelle des circuits intégrés, les approches conventionnelles d'affectation ne produisent plus les résultats optimaux. La plupart des méthodes et outils d'affectation basés sur des architectures de bus [106], [75], ont une limitation inhérente pour traiter la complexité des circuits intégrés actuels.

Des travaux sont proposés dans l'industrie et par des unités de recherche académiques pour corriger ces problèmes [37], [91], [25]. Ces travaux exploitent les nouvelles architectures et topologies comme les OCM. Traditionnellement, deux tendances sont à considérer dans les méthodes d'affectation. Il y a les méthodes qui préconisent la sépara-

tion de l'affectation du HSC. Dans tous les cas, l'objectif est d'optimiser des paramètres comme la distribution de l'horloge, minimiser la consommation d'énergie ou maximiser la dissipation de la chaleur dans les circuits. Parmi les approches utilisées pour le placement (avec partition ou pas), nous avons la méthode de la coupe minimum [35], [143], les méthodes analytiques [152], [168] qui sont considérées comme les meilleures en terme d'optimalité, et les méthodes hybrides qui combinent les deux approches précédentes [160], [93]. L'utilisation d'algorithmes méta heuristiques qui permettent de faire des explorations dans de très larges espaces de recherche ont permis aux méthodes analytiques de régler les problèmes de densités fonctionnelles des OCM [34].

Nous constatons malheureusement que les problèmes d'affection sont pour la plupart des méthodes souvent a posteriori à la synthèse et qu'à cette étape de la conception, certaines limitations des phases de développement précédant la synthèse et jumelées à la partition HSC peuvent entrainer des pertes de performance difficiles à corriger. Nous proposons dans le chapitre 4 [134] de cette thèse une méthode d'affectation dès la phase de la spécification fonctionnelle pour un système OCM.

CHAPITRE 3

THE ISOMETRIC ON-CHIP MULTIPROCESSOR COMPUTER ARCHITECTURE : ORGANIZATIONAL APPROACH

E. Elie, A. Hafid, M. Turcotte and J. A. Ferland

Abstract

A new architecture organization, called Isometric On-Chip Multiprocessor Architecture (ICMA), for parallel and distributed computing is introduced. This architecture integrates an equal distance organization of processors interleaved with memories for a three dimensional Integrated Circuit (3D-IC). ICMA is inspired by crystal structure of *sodium chloride* (NaCl) and based on bicolor three-dimensional mesh network structure rules. In this paper, we present ICMA that addresses topological organization for performance computation in On-Chip Multiprocessor (OCM) architectures. We also propose simple and effective communication scheme that combines both direct memory access and 1-Hamming distance wormhole routing algorithms, for a deadlock free network. Direct Memory Access ensures data transfer between neighboring nodes while 1-Hamming distance wormhole routing ensures data transfer between nodes that are not neighbors. Each node has its own network interface ; each interface is composed of six sets of channels connecting each node to its six neighbors. The routing is fully synchronous ; it uses absolute addressing of memory combined with node identifier. It executes 1-Hamming routing in an anticlockwise route from the source to the destination. A round robin algorithm is used when more than one access request is made on a node. The goal is to increase the proximity between data and processors, and hence, to increase the performance of the OCM systems. We developed two prototypes to evaluate the proposed algorithms.

Keywords—3D-IC, Interconnect, Parallel and distributed computing, On-Chip Multiprocessor, Scalability, Multiprocessor architecture, Hamming, Wormhole routing, Pa-

rallel computing, Isometric architecture, High connectivity, Interconnection, Scalability.

3.1 Introduction

The performance required by CPU-hungry applications, such as bioinformatics applications, is dramatically increasing. To support this type of applications, significant improvement in computer systems design is needed ; composing processors into a network to design a system is becoming a standard [78], [45], [102], and [21]. Typically, system design uses von Neumann computational approach with buses connecting processors to shared memories. This approach is seen as limited for improvement and the x8086 architecture based design leaded by Intel and IBM, for decades, is believed to be a wrong approach to upgrade/increase the performance. To overcome complexity, performance and flexibility limitations, multiprocessor architectures have been proposed and extensively studied for decades [60], [155]. Even though major efforts have been done in multi-computer interconnections, there is still a lot to do in on-chip parallel computer system design, especially in the era of three dimension integrated circuits (3D-IC).

It is only recently that major chip makers have started the commercialization of multi-core chips. Prior to these multi-core computers, many system architectures have incorporated either hypercube architectures or mesh architectures for parallel computer system design. On-Chip Multiprocessor architectures are usually compositions of single microprocessor chips networked by external physical wires. As the number of functional logics on a single silicon chip widens, network-on-chip has been proposed for Ultra Large Scale Integration (ULSI) IC system design. Even with this design, clock signal distribution, synchronization of modules, transmission of messages, and memory access contention still represent a major bottleneck that may cause considerable performance degradation.

On-Chip Multiprocessor (OCM) system is seen as solution for the design problem and has become the system design paradigm. Designing a network of processors on the conventional planar or Two-Dimension Integrated Circuit (2D-IC) has limitations on floor planning ; indeed, it limits the system design and performance improvements be-

cause of the distance data has to travel from one end of the chip to another [18]. These interconnections usually lead to long and irregular data transfert paths (and thus considerable delay), synchronization issues of clocks in different domains, and clock distribution problems. Furthermore, these limitations, in mixed-signal (analog and digital) system design, (supposed to enables better performance gains and low power consumption), lead to data corruption and signal integrity problem [132], [133]. Lastly, 2D Integrated Circuit design increases dramatically the design cycles and thus the chip cost [18] in an industry where being first in the market is critical.

In response to these limitations, solutions have been proposed including the 3D IC design [155], [165]. The 3D IC is an advanced die level vertical interconnects packaging technology offering low cost, ultra high speed and high density semiconductor IC. It enables vertical packaging for memory arrays and processing elements enabling considerable performance gain in communications between memory units and processing elements [104].

This paper addresses the design of a new system architecture and organization geared towards CPU-hungry applications, especially scientific applications. This architecture, Isometric on-Chip Multiprocessor Architecture (ICMA), is a three dimensional multiprocessor and multi-memory architecture. ICMA consists of processing elements (PE) interleaved with memory units (MU) to provide data proximity to PE. It is organized like the crystal structure of NaCl and following the rules of a bicolor graph structure. We believe that a better organization of processors and memories into on-chip architectures can considerably increase system performance [65] and [108]. The architecture is designed to be flexible, scalable and easily upgradeable for fast prototyping and design. We also (1) describe two techniques to access data in ICMA-based systems for better performance ; (2) evaluate the performance of these techniques ; and (3) implement two ICMA-based prototypes that show the feasibility of these techniques. With the first prototype, we validate the proposed architecture simulating matrix multiplication on an ICMA. We compare the performance of ICMA with that of HPMA [20] ; HPMA is composed of a number of processing elements and memory units which makes it more suitable to compare with ICMA. With the second prototype, implemented on a Xilinx

Multimedia Demo FPGA Board, we perform computation of mass-spring system.

ICMA-based systems can be built thanks to new technologies, such as the 3D IC [155]. The 3D IC is an advanced die level vertical interconnects packaging technology offering low cost, ultra high speed and high density semiconductor IC. It enables vertical packaging for memory arrays and processing elements enabling considerable performance gain in communications between memory units and processing elements [155]. The 3D IC can be used to analyze the impact of the (physical) isometry (i.e., equal distance between any 2 neighboring elements of ICMA), between the memory units and processing elements, on ICMA system performance. However, in this paper, we consider only logical isometry because we do not have a hardware prototype of ICMA.

This paper is organized as follows. Section 3.2 presents the Isometric On-Chip Multiprocessor Architecture and its organizational approach. Section 3.3 presents a brief description of ICMA performance. Section 3.4 introduces the decomposition of an application specification into components and Section 5.9 concludes the paper and presents future work.

3.2 Isometric On-Chip Multiprocessor Architecture

The Isometric On-Chip Multiprocessor Architecture is based on the crystal structure of the Sodium Chloride (NaCl).

3.2.1 Crystal Structure of the Sodium Chloride

Sodium chloride (NaCl), well known as salt or halite, is compose of extended arrays of aggregated ions of sodium (Na+) and of chlorine (Cl-) [53]. In NaCl structure, there are no identifiable discrete molecules, where a single molecule can be taken separately from the rest of the structure. Its coordination number (CN) for each atom, which is the number of nearest neighbor atoms, is 6. In other words, each ion of sodium has 6 ions of chlorine as neighbors and each ion of chlorine has 6 ions of sodium as neighbors. Consequently, the structure of the NaCl is then a three dimensions and isometric mesh (see Figure 3.1), where ions of sodium and chlorine are interleaved. Without lose of

generality in crystallography, we consider that the chlorine of the NaCl is analogous to a memory unit and the sodium is analogous to a processing element in ICMA architecture. ICMA is then a three dimension mesh architecture where memory units and processing elements are interleaved.

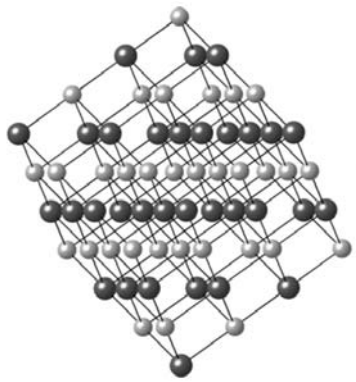

Figure 3.1 – Crystal Structure of Sodium Chloride [53]

3.2.2 ICMA Topology

Like in the crystal structure of NaCl, the CN of ICMA is 6. Each PE is connected to 6 neighboring MU and each MU is connected to 6 neighboring PE. Its topology of ICMA is regular and isometric as the one of NaCl (Figure 3.2). The basic element is a sub-lattice formed of four PE and four MU.

Figure 3.2 shows a sample ICMA topology of eight interconnected cubes. Neighboring cubes share their interconnected elements (PE or MU) (see Figure 3.5). Each interconnected element has six links to connect it to its six potential neighbors. These links can be six full-duplex links, six half-duplex links or twelve simplex. Figure 3.3.a shows PE connected to six MU while Figure 3.3.b shows MU connected to six PE. An extension to ICMA is build by adding a two dimension array of PE and MU. An element of this array, illustrated by Figure 3.4, can be seen as what Parhami in [138] called *"commodity node"*. As mentioned previously in the introduction, *"commodity node"* designs and interconnects are becoming important parts of fast prototyping and improvement of system capabilities.

Two different configurations are considered in ICMA Topology : (1) *homogeneous* configuration and (2) *heterogeneous* configuration. The homogeneous configuration contains processing elements of same kind (e.g., processor, adder, mutiplier, ...), memory units of same kind (type and size) and connection links of same kind (line size, frequence, ...) between the components (PE and MU). This configuration can be applied for example for 3D systolic array. The heterogeneous configuration in contrary contains different kinds of processing elements, different kinds of memory units and different kinds of connection links. This configuration can be applied for a 3D image processing pipe (IPP) as used in handheld industry.

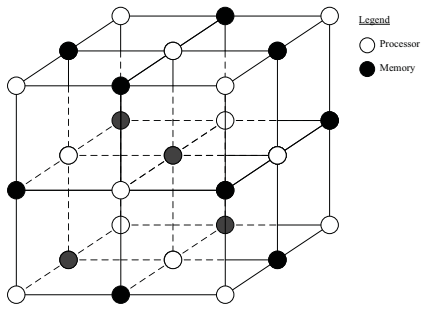

Figure 3.2 – Isometric On-Chip Multiprocessor Architecture

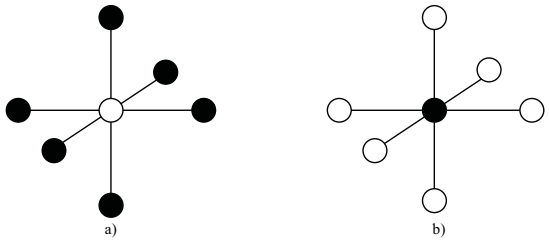

Figure 3.3 – ICMA node connection

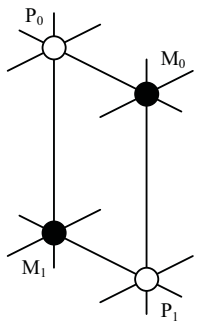

Figure 3.4 – ICMA Commodity Good

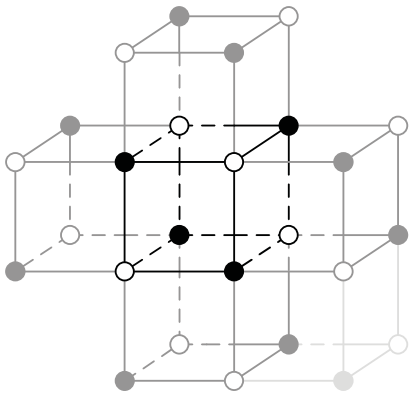

Figure 3.5 – Generic composition of ICMA sub-lattice

3.3 Performance of the Isometric On-Chip Multiprocessor Architecture

The performance of ICMA has not yet been exhaustively studied because, to the best of our knowledge, there is no benchmark to evaluate multiprocessor system in 3D-IC environment. Analytical and simulation models with computing benchmarks for ICMA are under development and will be published in near future. Nevertheless, with a careful distribution of data, we can intuitively state that the performance of ICMA based systems can be better than the performance of conventional multiprocessor systems (with all reserved for computer applications). For, in ICMA based systems, multiple processors can access multiple memories at the same time ; hence, it increases dramatically the parallelism and reduces data communication latency. It has been proved that the ratio

of performance on number of processors, decreases dramatically when the number of processors exceeds 8 for applications that require bigger data sort out [123]. Hence, the strength of ICMA approach lays on its capability to carefully distribute and share data on interleaved memories.

3.4 Isometric On-Chip Multiprocessor Architecture Prototypes

Two prototypes are proposed for the validation of ICMA. The first is designed for matrix multiplication and the second for Mass-Spring System.

3.4.1 Matrix Multiplication Prototype

Matrix multiplication logic is composed of four different sets of modules : (1) a PU ; (2) an ICMA interface to PU ; (3) a set of six MUs ; and (4) an ICMA interface to MU. The operands always flow from two MUs (to PU) and the result, once computed, in accordance with the matrix line and column size is written back to a third MU.

Figure 3.9 shows the processing element interface (PEI) state machine. In our implementation, each state machine connects three different MUs and each PEI has two such state machines. Figure 3.10 shows the register block of each PEI ; this block has 8 32-bits registers (one register for each connected node) and 2 32-bits registers for the multiplication result (64-bits word). Figure 3.11 shows the register block with the two data sources of each channel : one source from the PE and one source from the MU. Figure 3.12 shows how the processing element is connected to ICMA network by the PEI. The PE is composed of a multiplier and an adder. At a given clock cycle t, the processing element multiplies two words stored in the register block during the previous clock cycle $t-1$ and adds the multiplication result to the previous result called *preResult*. *preResult* is always available because it is the output of the last two registers of the register block. This field is always updated at each clock cycle during the matrix multiplication process.

Figure 3.13 shows the memory organization for the matrix multiplication ICMA architecture. The signals *interdata* and *interaddress* for internal data and for internal address respectively, are the outputs of two 6×1 multiplexers provided by the 6 connected

processing elements.

A typical message of matrix multiplication in ICMA is shown in Figure 3.9. The first word represents control information and the second word represents the payload. The flit address represents the address of the MU of the *ingress* transaction, the flit dst represents the id of the destination node and the flit src represents the source of the *egress* transaction. Let us now compute the maximum latency between two nodes for the matrix multiplication of a 64 nodes ICMA prototype. We assume that all connection channels in ICMA have the same bandwidth, node interfaces operate in synchronous mode with 1ns transmission delay and the simulation clock cycle is 5 ns.

The maximum channels between two given nodes of ICMA organized in $4 \times 4 \times 4$ network is given by the formula $3 \times (\sqrt[3]{n} - 1)$; it is equal to 9 channel in this example $(n = 64)$. Thus, the message latency is 108 ns.

Let us also compute the bandwidth for the same ICMA prototype. At anytime, each channel of ICMA can handle a transaction of 64-bits in 10 ns. The bandwidth of each channel is then 6.4 Gbs and the whole network is connected by 192 channels $(n = 4)$. The data rate of the overall ICMA is then 1228.8 Gbs. The number of available channels, left opened for external input/output connections, is 96 for the 64 nodes ICMA ; these channels can sustain a bandwidth of 614.4 Gbs. This example shows the potential of data communication in ICMA.

Even though there is no evidence in the literature, to the best of our knowledge, of any 3D IC based benchmark to evaluate ICMA, we proposed to compare its performance to a prototype of Hierarchical Processor and Memory Architecture (HPMA) [20]. The choice of HPMA is based on the fact, like ICAM, it is multi-level hierarchical architecture where each level can be seen as a block for the next level (*commodity good*).

3.4.2 Hierarchical Processor and Memory Architecture

Hierarchical Processor and Memory (HPAM) architecture is presented in [20] as a cost-effective multiprocessor system design to deliver high performance computation capabilities. The architecture basic element is a combination of processors and memories (PAM) [20]. Each element is composed of homogeneous parallel processors and memory

units. HPAM can be seen as a heterogeneous multiprocessor architecture composed of homogeneous multiprocessor architectures (per level) ; each level of the hierarchy is used as a block for the next level (Figure 3.6). HPAM architecture presents $n - levels$ hierarchy ; each level is separated from the next by an interface. The organization of this hierarchy is based on the speed of the processors. From 1 to n, the number of processors increases, their speed decreases and the size of memory units increases. Each level of the hierarchy has homogeneous processors ; however, processors can change from one level to the other.

Miled et al. [20] present HPAM architecture as an analogy to the memory organization and hierarchy. The authors consider two classes of applications that require 1014 computations : time-constrained fixed-size applications and large-scale complex applications. A time-constrained fixed-size application must finish its execution within a specific time frame with a fixed size dataset (e.g., simulation of interactive and immersive visualization system). Large-scale complex applications are grouped in SPEC High-Performance benchmarks [56] (e.g., General Atomic and Molecular Electronic Structure System (GAMESS) used in pharmaceutical and chemical industries).

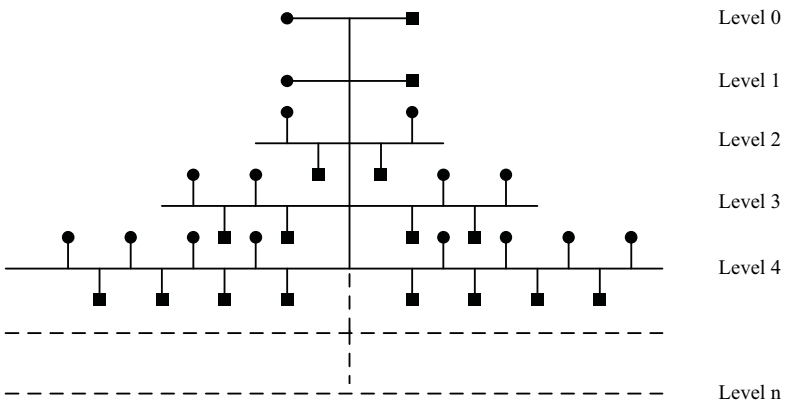

Figure 3.6 – Architecture of the HPAM prototype

The two classes of applications have been computed on HPAM of 10, 100 and 1000 processors using first one level hierarchy and then two levels hierarchy. The authors reported [20] that the two levels hierarchy system has a better ratio of cost over performance than the one level hierarchy system.

3.4.3 Component Specifications of the Matrix Multiplication Prototypes

ICMA for matrix multiplication and HPAM prototypes use the same number of PEs and MUs. A PE is an IP (Intellectual Property) that is composed of *load* and *store* instructions, a register file, a multiplier that performs the first multiplication in 16 clock cycles and each other multiplication in one clock cycle, and an adder that performs an addition of two words in the same clock cycle as the multiplication. The MU is a 1Kb single port RAM. In ICMA, PE is directly connected to MU. In the HPAM prototype, we use PCI local bus protocol to connect nodes. In both cases, prototypes are composed of 16 PEs and 16 MUs. We ran the simulations using Mentor Graphics ModelSim 6.1.e environment.

3.4.4 Computation of Matrix Multiplication by ICMA

Table 3.I – Pseudo code of $n \times n$ matrix multiplication

```
...
for(i=0 ; i<n ; i++)
   for(j=0 ; j<n ; j++)
      for(k=0 ; k<n ; k++)
         C[i][j] = A[i][k] + B[k][j]
...
```

Figure 3.6 and Figure 3.7 show how the matrices are computed and distributed on the memories of ICMA and HPAM prototypes. Let γ, α and β be three matrices of size $n \times n$. Let us consider the computation of the element $\gamma(i, j)$ where i is a line from matrix and j is a column from matrix α. Let us divide the matrix α (and β) to m sub-matrices.

As multiplication needs full lines of matrix α and full columns of matrix β to be performed, the accesses to these lines and columns are done in parallel because each processing element has a copy of matrix α in one connected memory and a copy of matrix β in another connected memory. The flow of operands from memory to processing elements is then automatic and is done within the same clock cycle. The resultant matrix of each processing element multiplication is a portion of γ and is a matrix of size

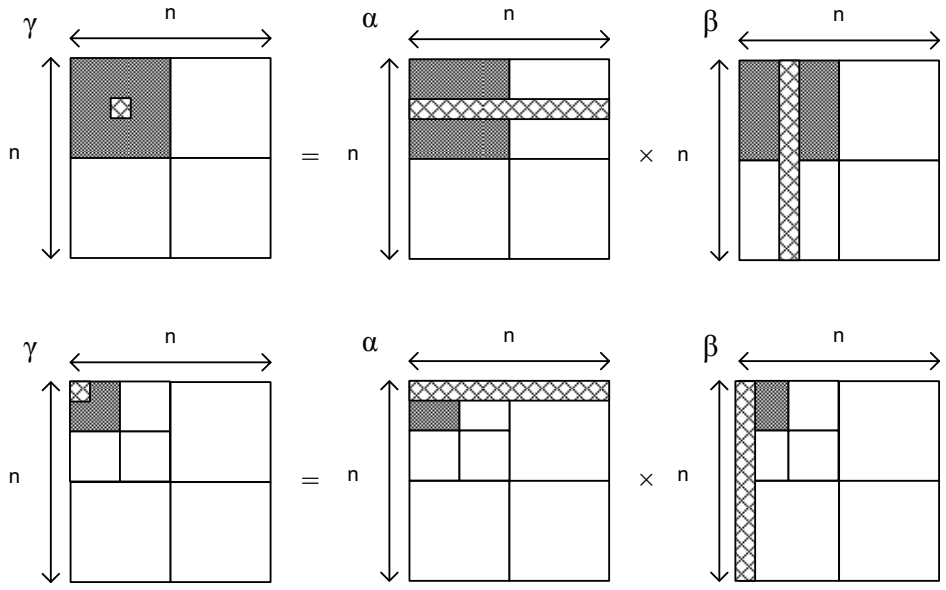

Figure 3.7 – Computation of matrix γ with full operand memories

$(n/m) \times (n/m)$ and is located in a third separate memory. The write back of the multiplication result is done with no delay. Figure 3.7 shows that the size of α and β is $n \times n$ and of each resultant matrix is $(n/2) \times (n/2)$. The matrix γ of overall computation is the composition of all resultant matrices from different memories.

Figure 3.8, unlike Figure 3.7, shows that matrices α and β are divided and distributed on connected memories of ICMA prototype. Each matrix slice is $(n/4) \times (n/4)$ of size and each memory hosts only one matrix. Consequently, a line or column of matrix α or β is stored on more than one memory. To have a full line i or column j for computation, processing elements need to read line or column complements from remote memories. Figure 3.7 shows that the computation of each hatched block requires a processing element to read a complement of line i or column j from different matrix. As multiplication and addition are commutative operations, the order of access is not important ; in this simulation, the order of accesses to memories is hardcoded in processing element interfaces.

In ICMA prototype, each PE can compute at each clock cycle two multiplication and two addition operations because operands are available at each clock cycle. Hence, a

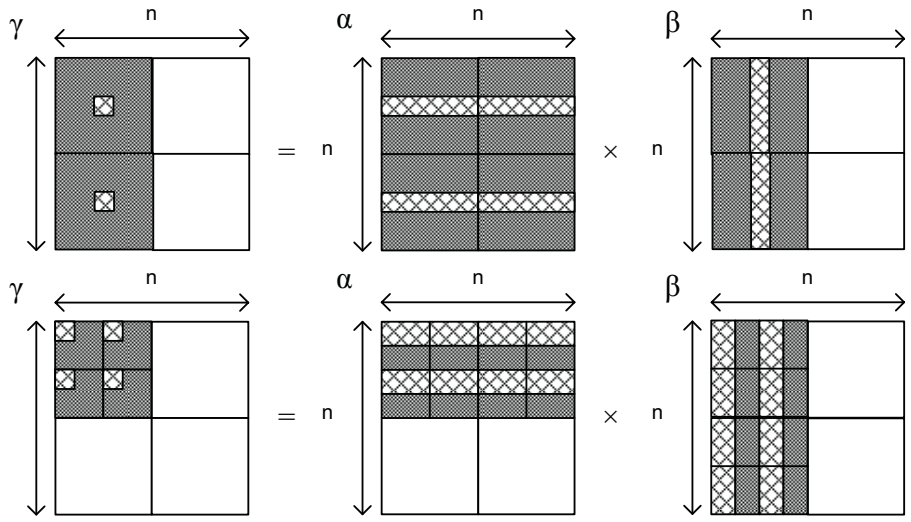

Figure 3.8 – Computation of matrix γ with decomposed operand memories

multiplication and an addition are done at each clock cycle ; a result is written at each 32 clock cycles, plus 16 latency clock cycles for the first multiplication.

In HPAM prototype, the execution time of the matrix multiplication depends on the level of the architecture and the number of processors engaged in the computation. At level 0, the computation is done as with general purpose computing systems. The prototype has one memory unit and one processing element. Since we are using a PCI local protocol the read instruction takes 4 clock cycles while the write instruction takes 3 clock cycles. Even if the computation is done in burst transaction mode, it takes 8 clock cycles to read two operands and 3 clock cycles to write the result of the multiplication back. At level 1, the multiplication is done as by two separate processors. The computation at each level of the HPAM is serial when the number of processors and memories are greater than 1 at that level. Let us recall that the interconnection medium between nodes on the HPAM is a local bus using a PCI protocol.

We have successfully designed the matrix multiplication prototype for direct memory access using VHDL on Mentor Graphics ModelSim 6.1.e simulation and debugging environment. Figure 3.9 to Figure 3.13 show implementation details of data transfert in ICMA in the case of matrix multiplication. This prototype uses exclusively direct

memory access. Table 3.I shows an example of a pseudo code of matrix multiplication that is performed by ICMA.

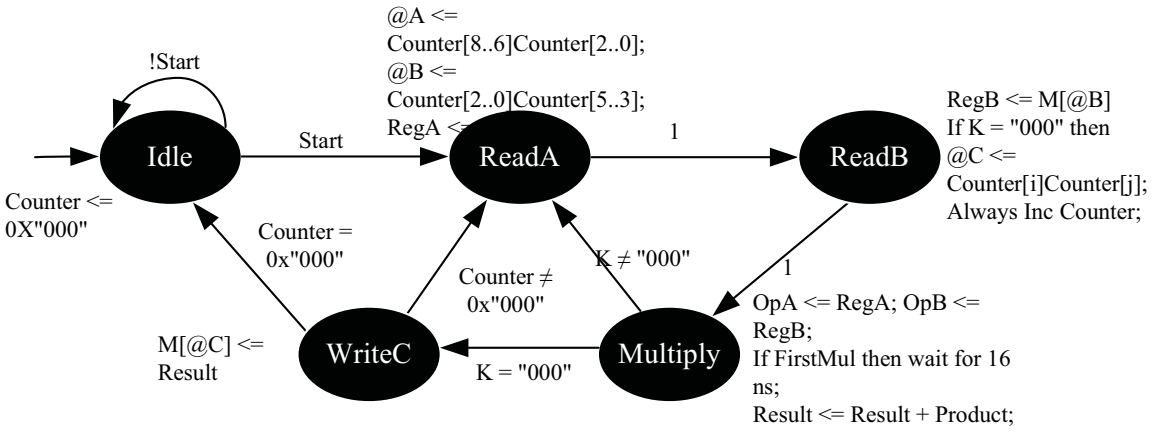

Figure 3.9 – State machine implemented in the processing element interface for the matrix multiplication

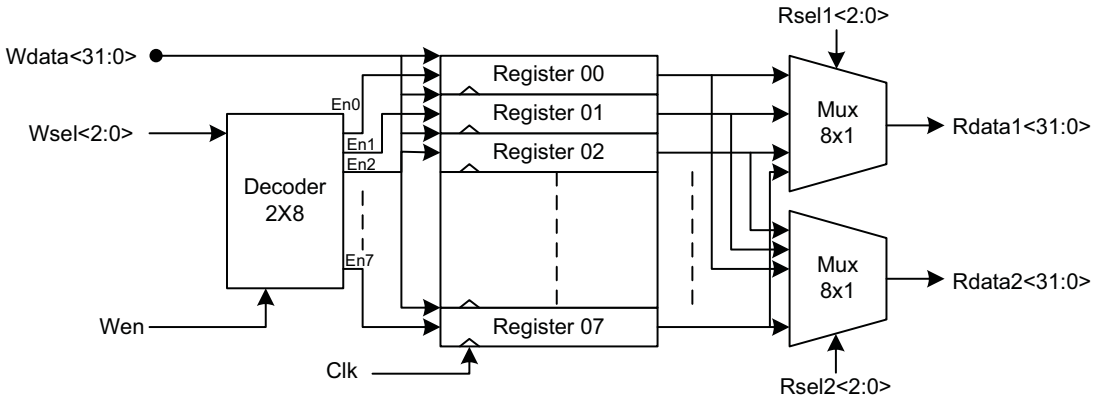

Figure 3.10 – Register bloc of processing element interface

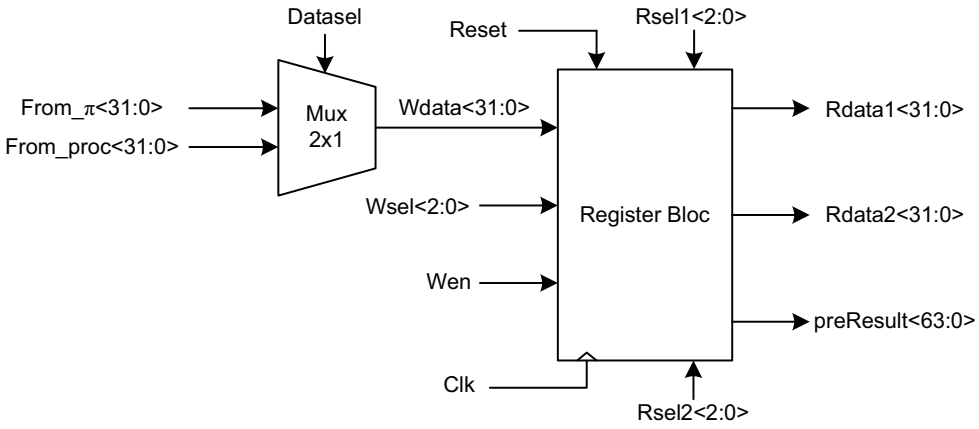

Figure 3.11 – Block Diagram of Register Bloc with Source Information

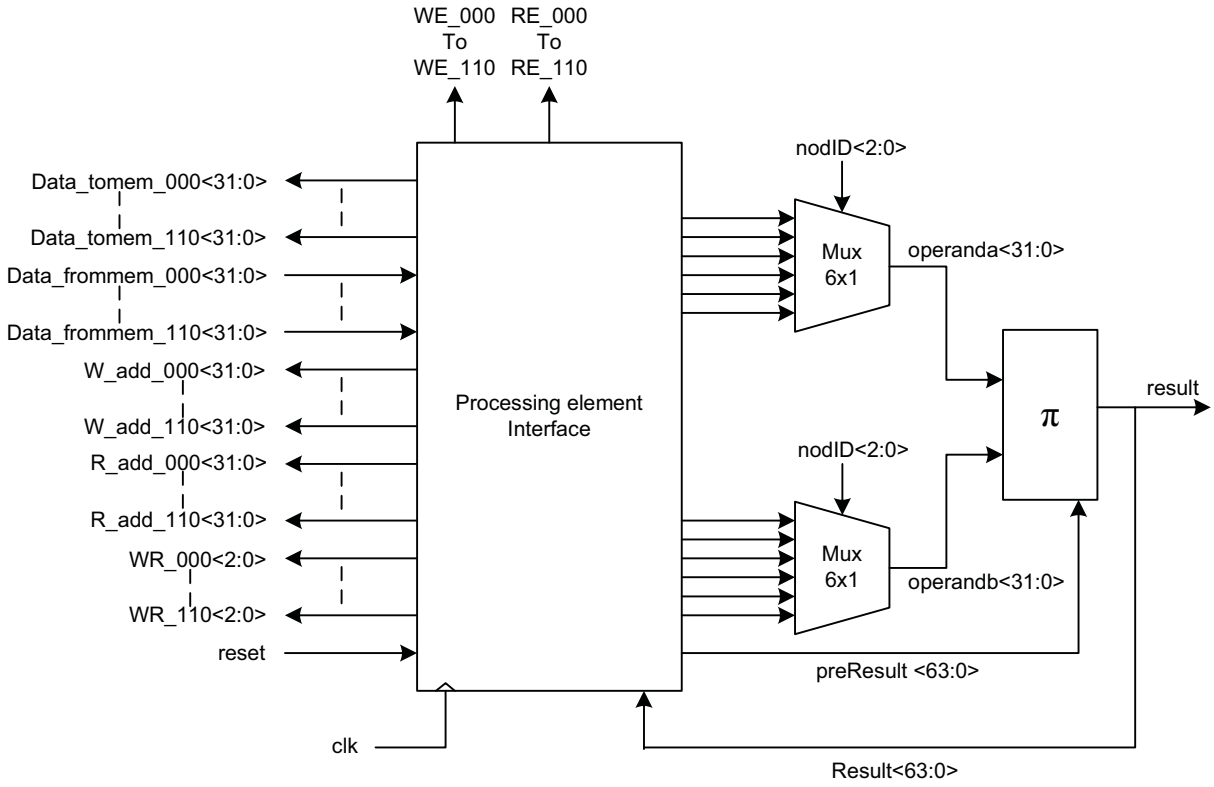

Figure 3.12 – Block Diagram of Processing Element and its Interface

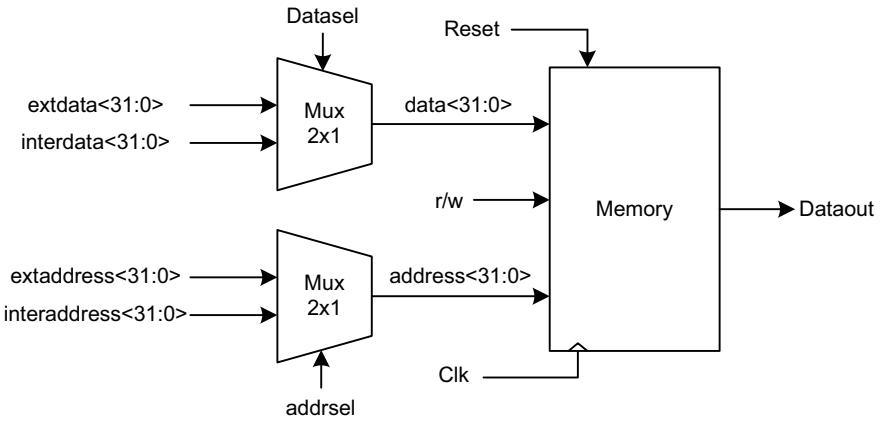

Figure 3.13 – Block Diagram of Memory Unit with Data and Address Sources

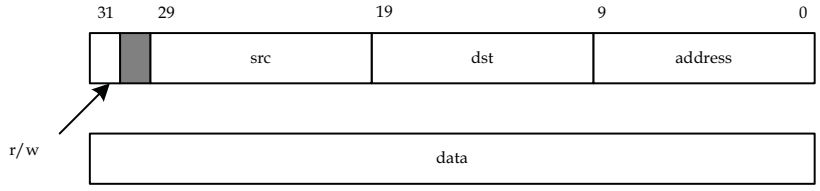

Figure 3.14 – ICMA Network Message Structure

3.4.5 Result Analysis for Matrix Multiplication

We report the results of an empirical evaluation of ICMA and HPAM prototypes. We measure the execution time for a 32×32 matrix multiplication using a symmetric distribution of the data on memory units and of processes on processing elements.

Figure 3.15 shows that ICMA performs better than HPAM by one order of magnitude (in terms of execution time) ; the upper graph represents the results for ICMA while the lower graph represents the results of HPAM. For example, using one processor ICMA takes 32784 nanoseconds to perform matrix multiplication while HPAM takes 314368

nanoseconds for the same operation ; this performance difference is caused by the sequential access of the processing element to the memory in the HPAM prototype using PCI bus on the same level of computation ; ICMA has access to six different memory units at each clock cycle. Increasing the number of processors (in HPAM) at the same level of computation does have no impact ; we have the same execution time for two processors and for four processors (e.g., no performance improvement when increasing the number of processors from 8 to 16).

The results show that ICMA outperforms HPAM by one order of magnitude. This supports our belief that better performance of a multiprocessor architecture can be achieved by a better organization of processing elements and memory units.

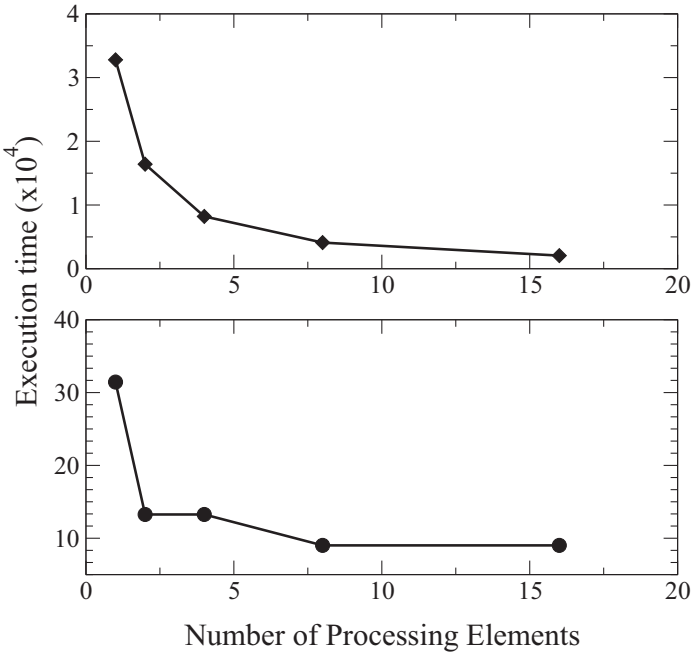

Figure 3.15 – ICMA and HPAM Execution Time for 32×32 matrix

3.4.6 Mass Spring System Prototype

As shown in Figure 3.16, a mass spring system is a two dimension mesh $m \times n$ of masses where each mass is link to its neighbors springs. In the following, for each

mass (i, j), we consider spring connections with masses at position $(i, j+1)$, $(i+1, j)$, $(i, j-1)$ and $(i-1, j-1)$. Anytime a stress is applied to a mass, a wave is propagated throughout the system until the springs return to their stable condition. We use the mass spring system to characterize the behavior and the 1-Hamming wormhole communication scheme of ICMA. We compute the mass spring system behavior with a prototype of ICMA and show how the ICMA behaves in this context.

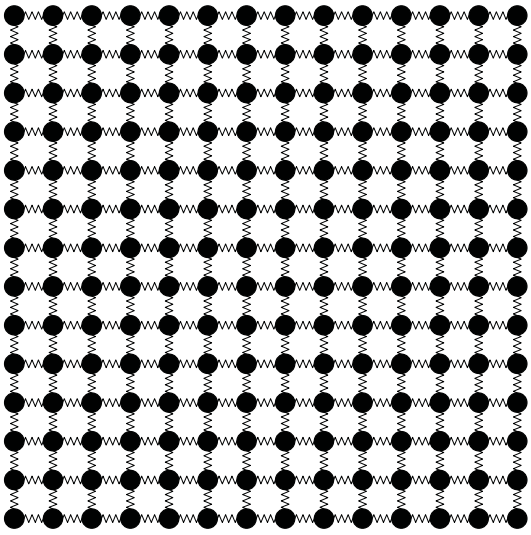

Figure 3.16 – A Mass-Spring System

3.4.6.1 Architectural Components of Mass Spring System Prototype

- Four 32-bit MicroBlaze processors P0, P1, P2, P3 [178]
- Four memory units M0, M1, M2, M3 of 1024 32-bit words
- Fast Simplex Link bus [178] to connect Microblaze processors and the Memories
- MicroBlaze and memory controllers used to transfer packet sent by the computation processes based 1-Hamming Wormhole communication protocol
- VGA controller and graphical interface

- Two different dual port memories are used : one for the abscissas x and the second for the ordinates y of the masses. Both memories are managed by the same memory controller. At each VGA controller clock cycle, the current positions (x, y) of the masses are read from the memories and the new positions are computed and write back to these memories

- Mutiple Clock Domain (MCD) provides 27 Mhz clock for the computation processes and 25 Mhz for the display unit and the VGA controller

3.4.6.2 MicroBlaze Interface in ICMA

The Microblaze interface implements six parallel processes. The first two processes Microbalze uses to communicate with its connected MUs, correspond to Xilinx Platform Studio read and write primitives. Three memory management processes are used ; one for each connected memory. A barrier process called *The_Lock* manages and synchronizes access to the Microblaze Fast Simplex Link (FSL). To have access to the FSL bus, *The_Lock* process must lock the bus and have an exclusive access for message transmission. *The_Lock* process has a special priority management function that helps avoiding ICMA communication deadlock. This function assigns the lowest priority to a node that has just locked the FSL bus and made a transaction. A round robin algorithm is then used to assign the FSL link to other nodes if it is requested.

3.4.6.3 Memory Unit Interface

The memory interface implements one VHDL writing process. This process limits the access to only one message at a time. Once a message sent to a memory is processed, its interface inverses the source ID and destination ID flits before sending the response to the processor. The advantage of using one process is to simplify the access to the memory and avoid message collisions. Nevertheless, the message collisions are managed by the process *The_Lock*. Likewise, as each channel has its own message management process in the processor interface, the communication is done in parallel and hence deadlocks are avoided.

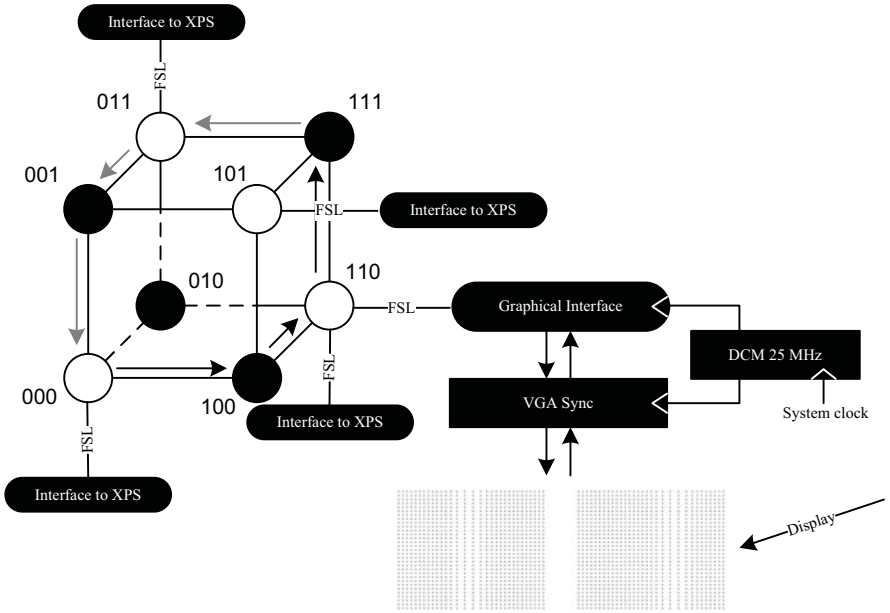

Figure 3.17 – Architecture for the Implementation of 2D Mass-Spring System

3.4.7 Computation of 2D Mass-Spring System with ICMA

The 2D Mass-Spring system is used to emulate the proposed ICMA architecture. Figure 3.10 shows the Mass-Spring system as 14×14 matrix of masses placed on two dimensions interconnected with springs. The springs are placed in horizontal and vertical directions. Due to chip size limitation on the prototyping board, we only used four Microblaze processors and four 4KByte memories. However, the communication protocol used for this configuration can be used with any other configuration with an arbitrary number of processors and memories. All four processors work in parallel to simulate the actions of the applied power on the masses by the springs.

Let us now define the behavior of the system. For each mass, we must compute the sum of forces applied by the four connected springs on x coordinates and on y coordinates. Let us assume that the mass and the spring constants are equal to 1, d is the distance between two masses at a given time t, and l is the spring length. The forces f_x and f_y on x and y respectively for a given spring are :

$$d = \sqrt{(x_{adjacent} - x)^2 + (y_{adjacent} - y)^2} \tag{3.1}$$

$$f_x = \frac{(x_{adjacent} - x) \times (d - l)}{d} \tag{3.2}$$

$$f_y = \frac{(y_{adjacent} - y) \times (d - l)}{d} \tag{3.3}$$

Given f_x and f_y, we can compute the current velocity, vx and v_y, of a spring on position x, y as follows :

Let v_x and v_y be these velocities, and dt the simulation step.

$$v_x = v_{x_{previous}} + f_x \times dt \tag{3.4}$$

$$v_y = v_{y_{previous}} + f_y \times dt \tag{3.5}$$

The new position of a mass at x and y can be computed as follows :

$$p_x = p_{x_{previous}} + v_x \times dt \tag{3.6}$$

$$p_y = p_{y_{previous}} + v_y \times dt \tag{3.7}$$

Equation 3.1 to Equation 3.5 are computed during the same clock cycle to keep data coherence and synchronization. To reduce the inter processor communication overheads, the 14×14 mass-spring system matrix is decomposed into four 7×7 mass-spring system matrices (49 masses) as shown in Figure 3.11 ; each sub-matrix is computed by one processor. The four processors share information only on masses of rows 7 and 8 and of columns 7 and 8. Similarly, the coordinates of masses are stored in the four memories. However, coordinates of masses on rows 7 and 8 and on columns 7 and 8 are stored in the memory located at the left of each processor, using the anticlockwise direction. Figure 3.11 shows processors 011 and 000 share information on masses of row 8 located in memory 100.

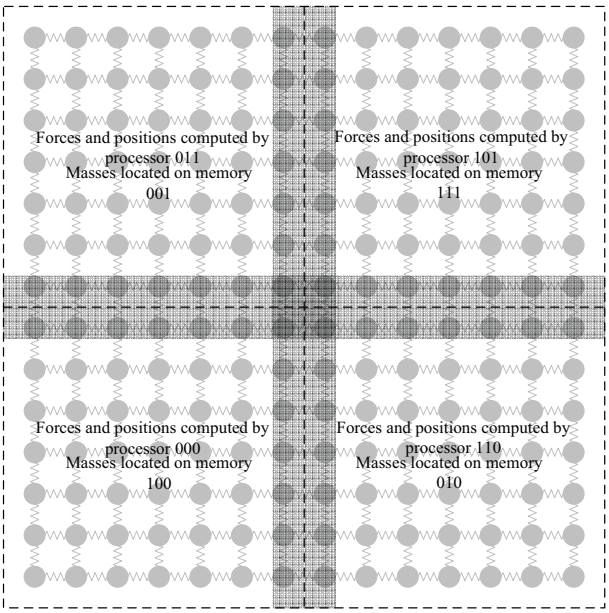

Figure 3.18 – Distribution of masses on processors and memories

The data coherence is critical in order to write back the correct position ; hence, a master processor is used to synchronize the computation of new positions. In Figure 3.17, the Microblaze processor 110 is the master processor for the whole system synchronization.

All ICMA interfaces are shown in Figure 3.19 ; each interface contains a buffer, a

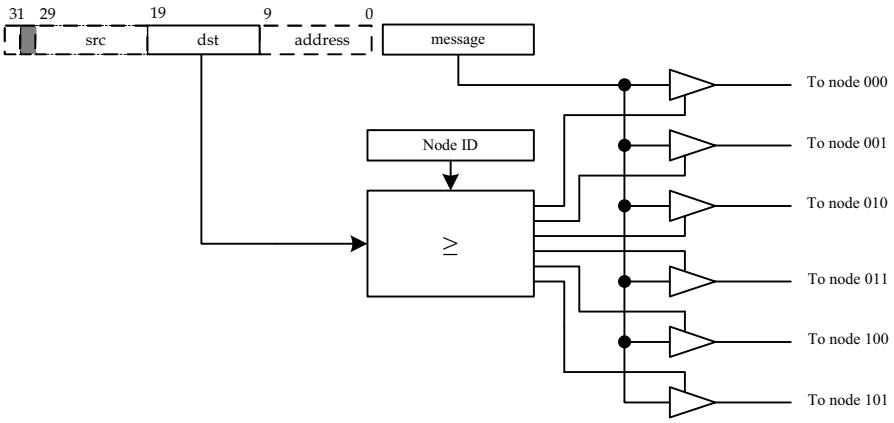

Figure 3.19 – Blobk Diagram of Interface Unit

comparator and an output arbitrator. The comparator compares the ID of the current node with the destination address. If the destination address matches the node ID, the message is processed by the current node. Otherwise, the message is transferred to the new intermediate or final destination node. ICMA for Mass-spring system functions as a virtual path network (VPN). Paths through the network are dedicated. Messages from a source to a destination always take the same path through the network based on the 1-hamming wormhole algorithm. In the case where more than one node sends messages to (or through) a node, all messages are received and buffered into a buffer queue.

The 2D Mass-Spring System is implemented on Virtex-II Multimedia development board. This board features a Virtex-II XC2V2000-FF896 as the user application FPGA. A soft core processor, 32bit Xilinx Microblaze, is used on the Multimedia board as an engine for application processing. On board ZBT RAM of 18Mbit, divided in five independent banks running at 130MHz with byte write capability, is used as storage for microprocessor code and data store for video frame buffering.

ICMA for 2D mass-spring system implementation uses ISE 6.3.03i and Xilinx Platform Studio (XPS). The programming of a Microblaze processor and its peripherals are made easy by XPS ; each Microblaze comes with its own programming interface that

helps programming it in C. Microblaze processors and memory interfaces are designed using VHDL ; these interfaces are responsible for the synchronization of the processors and memory accesses.

3.5 Data transfert in ICMA

In this section, we present the details of the two data transfert techniques to be used in ICMA-based systems : direct memory access between neighboring nodes and Hamming wormhole routing algorithm [45] between non-neighboring nodes. To the best of our knowledge, this paper presents the first attempt to use an integrated approach of data transfert using Direct Memory Access (DMA) and Hamming Distance combine with Wormhole (HDW) algorithm in 3D multi processors and multi-memories based systems. We use the DMA protocol for neighboring nodes communication and the HDW protocol for the remote nodes access.

ICMA direct memory access is initiated when a PE requests *read/write* transactions from a neighboring MU. The PE interface sets a signal to indicate a direct access request to the MU interface ; this operation is used when the instruction and operands are known to be in the neighboring MUs. A direct path is set between the PE and the MUs for all the transactions related to that instruction. Nevertheless, a message, transmitted between neighboring nodes, contains the source address and destination address.

Figure 3.20 shows the data flow from MUs to the PE ; each PE interface can handle six transactions in parallel. Let a and b be two operands located in two different MUs and r be the result of their computation (e.g., addition, multiplication) located in a third MU. Let i, j, k, v, t and w be the indices of memory locations a, b and r in their memories respectively. The read transactions for data a_{ik}, b_{kj} and a_{vt}, b_{tw}, are performed in parallel ; they are used to compute in parallel r_{ij} and r_{vw} respectively. The write transactions for the results r_{ij} and r_{vw} are performed in parallel.

In ICMA, two different instructions can be executed in parallel if three MUs are used for the computation of each instruction. Let us recall that PE is superscalar and can compute multiple instructions in parallel. Likewise, three different instructions can

be executed in parallel if two MUs are used for the same computation. This means that the instruction operands come from two different MUs, connected to PE and the result is written back to one of the two MUs. A maximum of six memory accesses can be requested in parallel ; in this case, each PE and MU tandem behaves like a general purpose computer. For communication with remote nodes, we propose to use deterministic routing, namely 1-hamming wormhole algorithm with anticlockwise direction paths, to enable data transfert between non-neighboring nodes in ICMA.

When a node receives a message, it checks whether it is the message destination. If the response is yes, it processes the message ; otherwise, it compares the destination ID with its own ID, and sends the message to the connected node that has the first most significant different bit. For example (Figure 3.5), the transaction from 000 to 111 goes through 100. When the node 100 receives the message and checks that the destination ID is different from its own ID, node 100 sends the message to the node 110, because the next different bit between 100 and 111 is the second most significant. The algorithm is repeated until the destination address matches the node ID.

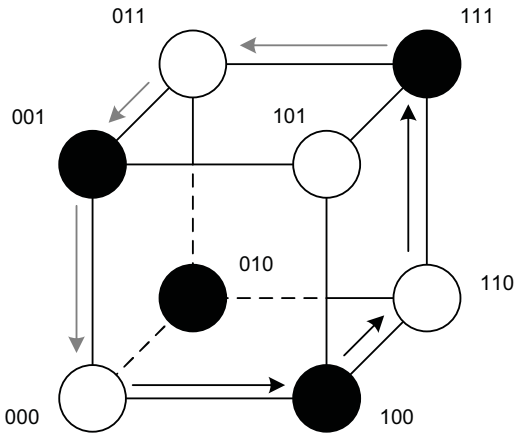

Figure 3.20 – Message path with node ID in ICMA architecture network

Data transfer in ICMA based systems must solve three problems : channel contention, deadlock and livelock. In the case of channel contention (i.e. more than one message is sent through the same node), multiple buffers are used by the node interface : one buffer for each channel. Using buffers to solve contention is not new ; furthermore, it increases latency. In ICMA, the latency increase is minimal to null ; indeed, the pro-

bability of buffering is small (almost zero in most of the cases) since data is usually placed in neighboring MUs to the PEs that process this data. Application decomposition (in terms of processes and data sets) and placement of these data sets and processes in ICMA nodes is the subject of an upcoming paper. Indeed, the objective is to place data sets (in MUs) near processes (in PEs) that require/use these data sets.

With respect to deadlock, since *egress* and *ingress* messages are routed in the same direction, and each node interface has its own buffers, the protocol provides a deadlock free network. With respect to livelock, since each node of ICMA has a unique ID and the naming is based on HDW algorithm, the problem of livelock is nonexistent ; a message in this architecture will never loop around the network forever.

To evaluate the performance of the data transfert protocols, we define two metrics that can be used to evaluate ICMA : (1) bandwidth ; and (2) maximum latency between any 2 nodes.

The bandwidth, C, of ICMA is defined as follows :

$$C = \sum_{i=1}^{n} W_i \tag{3.8}$$

where n is the number of channels, and is the bandwidth of channel i $(1 \leq i \leq n)$. In the case of homogeneous ICMA (see Section II), we have

$$C = n \times W_i \tag{3.9}$$

The message latency between nodes is determined by the number of nodes the message passes through from the source node to the destination node. For homogeneous configurations, the formula of the message latency lmessage is given by the expression :

$$l_{message} = T_{clock} \times (\varphi D_{pe} + \delta D_{me} + 2M) + T_{execution} \tag{3.10}$$

where T_{clock} is the system clock time, D_{pe} is a compulsory delay when a message passes through a processing element, D_{me} is a compulsory delay when a message passes through a memory unit which is not the target memory, M is the distance from the processing

element that initiates the transaction and the target memory unit, δ and φ are the number of memory units and processing elements respectively, and $T_{execution}$ is the execution time of the target memory unit. For heterogeneous configurations, the formula of $l_{message}$ is given by the expression of Equation 3.11 :

$$l_{message} = \sum_{i=1}^{n} T_{clock_i} \times D_{pe_i} + \sum_{j=1}^{m} T_{clock_j} \times D_{me_j} + \sum_{k=1}^{p} M_k + T_{execution} \tag{3.11}$$

where and n, m, p are the number of PEs, the MUs and connections between nodes from the initiator of the transaction to its destination respectively. For example, in the case of a 1024-node ICMA, organized as $8 \times 8 \times 16$, the maximum latency $l_{message}$ between two nodes for two 32 bits words is : $l_{message} = 480ns + T_{execution}$.

In the case of massive data requirements, additional external memory units can be connected to processors (with available interfaces) that act as Memory Management Interface (MMI). On-chip memory units act then like cache memories. In the case the on-chip memory units are sufficient for the application computation, all the data are distributed and hosted on chip. ICMA has two different configurations : homogeneous configuration and heterogeneous configuration. The homogeneous configuration is composed of identical processing elements (PE) and identical memory units (MU) for recurrent executions of the application processes. The heterogeneous configuration is composed of different structures of processing elements and memory units for applications with no recurrent processes. In the case of homogeneous and heterogeneous configurations, communication is based on interdependencies of instructions, data coherences and bus protocols used in the structure.

3.6 The Limits of the ICMA Architecture

The evaluation of the ICMA is based on two prototypes : the matrix multiplication prototype and the mass spring system prototype. The fact that we are unable to measure the performance of the ICMA is because of the following constraints : (1) 3D IC technology is in its embryonic phase and not mature enough for evaluated properly compare

to 2D-IC design ; (2) even though some CAD tool companies are working on the 3D-IC design environment, to the best of our knowledge, there is no available tool that can be used to measure accurately the performance of the ICMA ; and (3) the lack of specific benchmarks for the 3D-IC design make the evaluation of the ICM difficult.

We are working on the development of analytical and simulation models with computing benchmarks for ICMA and will be hopefully published in near future.

3.7 Conclusion

A lot of research has been done in developing computer architectures to increase performance. However, existing multi-processor architectures have limitations. In this paper, we proposed a new architecture organization, called Isometric on-Chip Multiprocessor Architecture (ICMA), which proposes solutions to these limitations. ICMA is based on the NaCl structure and is believed to alleviate the performance problems faced by existing distributed and shared memory multiprocessor architectures. Currently, we are working on the modeling of ICMA as a queuing network for the evaluation of its performance. Meanwhile, we developed methods to decompose applications or specification of an OCM and to map the resulting components (processes and data) to ICMA nodes (processors and memory units).

We also proposed two techniques (direct memory access and 1-hamming wormhole routing algorithm) that can be used to enable communication among the nodes (PEs and MUs) in ICMA. We developed two ICMA prototypes : a simulated prototype and real/hardware prototype. The prototypes show clearly the feasibility and the benefits of ICMA. Indeed, now we are confident that ICMA can be realized (i.e., become reality) given recent advances in 3D IC.

Now that the feasibility of ICMA including communications between any of its two elements has been shown, techniques to decompose complex applications and map the resulting components to the elements (i.e., PEs and MUs) of ICMA are needed. This is the topic of our future work ; We are also planning to implement (in the near future) ICMA on a high performance FPGA board for the computation of RNA Secondary

Structure prediction called Profile-Dynalign [15]. The projection of a 3D ICMA on 2D FPGA structure will impact its performance ; however, it will give us a good understanding of ICMA and help us infer key findings to the 3D ICMA. This being said, our ultimate goal is to implement ICMA on 3D IC.

CHAPITRE 4

A NEW MULTI-PHASE CELL FORMATION PROBLEM METHOD FOR APPLICATION DECOMPOSITION FOR ON-CHIP MULTIPROCESSOR SYSTEM DESIGN

E. Elie, J. A. Ferland, A. Hafid, M. Turcotte, and J. Bellemare

Abstract

As interconnects become important parts of On-Chip Multiprocessor (OCM) system design, effective decomposition, also known as partitioning, of computational applications is critical and challenging issue for OCM architecture. This decomposition aims to break down complex computational applications into less complex and easy to solve components, with the objective to minimize OCM communication cost. Methods based on graph partitioning, simulated annealing, domain decomposition, recursive spectral bisection and multi-constraint graph partitioning are proposed in the literature. However, most of these methods in integrated circuit (IC) design focus on post-synthesis circuit partitioning. With the trends in gigascale integrated system (GIS) new approaches are needed ; these approaches must take into consideration the decomposition problem at the beginning of the design process (computational problem specification level). In this paper, we propose a new method for computational problem decomposition for OCM architecture. This method, called Integer Cell Formation Problem (ICFP), is based on Cell Formation Problem (CFP). The objectives of ICFP are to (1) minimize communications among the components of OCM ; (2) load balance communications among the components of OCM ; and (3) find an optimal number of nodes for OCM design. The propose method is a multi-phase optimization problem resolution that rearranges processes and data according to similarity rules. It transforms application into a set of cells so that intra-cell communications are maximized and inter-cell communications are minimized leading to better computation performance. The application under decomposition is vie-

wed as an incidence matrix $A = M \times N$, where M is the number of processes of the application, N represents the number of data, and each element a_{ij} of A is the access rate to a datum j by a process i. We first describe how Cell Formation Problem is applied to application decomposition then we provide illustrative examples and analysis to demonstrate the effectiveness of the proposed method ; we will also show that the introduction of a load balancing factor into the cell formation objective function can yield better performance of the application or functional specification. Experiments are run in Linux environment to validate our proposed method.

Keywords—Application decomposition, Cell Formation Problem, Clusturing, Load balancing, On-Chip Multiprocessor, Network-on-Chip.

4.1 Introduction

On-Chip Multiprocessor (OCM) architecture, with its accompanying communication network-on-chip (NoC), is an emerging paradigm for the design of gigascale integrated system (GIS). Consequently, interconnects become important parts of OCM and there is increasing need to address the control of communication flow among OCM components. To get the best of OCM, effective application decomposition, also known as application partition, of CPU hungry applications is needed. This decomposition aims to break down complex applications into less complex and easy to solve components, with the objective to minimize OCM communications. In this paper, we propose a new method for computational problem decomposition for OCM architecture. This method, called Integer Cell Formation Problem (ICFP), is based on Cell Formation Problem (CFP). CFP from group technology (GT) is mostly used in production engineering ; it is defined as a production philosophy that identifies similar parts and groups them together to take advantage of their similarities in manufacturing and design processes [137]. It also aims to simplify the flow of processed parts (data) among the machines (functions) by reorganizing the parts into families and machines into cells so that each family is almost processed in one cell. Applying CFP methods to computational application decomposition is expected to yield significant performance improvement, because the flow of communications

between cells is greatly reduced [29].

The objectives of our proposed method ICFP is to (1) minimize communications among the components of OCM ; (2) load balance communications among the components of OCM ; and (3) find an optimal number of nodes for OCM design. ICFP is a multi-phase optimization problem resolution. It transforms computational applications into sets of cells so that intra-cell communications are maximized and inter-cell communications are minimized leading to better performance. The application under decomposition is viewed as an incidence matrix $A = M \times N$, where M is the number of processes of the application, N represents the number of data, and each element a_{ij} of A is the access rate to a datum j by a process i. We first describe how Cell Formation Problem is applied to application decomposition ; then, we provide illustrative examples and analysis to demonstrate the effectiveness of the proposed method ; we will also show that the introduction of a load balancing factor into the cell formation objective function can yield better performance of the computational application. Experiments are run in Linux environment to validate our proposed method.

4.1.1 Motivation

Interconnects become important parts of OCM design and there is increasing need to address the control of communication flow between OCM components. To get the best of OCM, effective decomposition of computational applications is needed. Oswens *et al.* [136] show that enormous computational resources are used to move and store data across OCM. Hence effective design of OCM must minimize the move of data. Also, with the trends in 3D-IC technology [165], 3D On-Chip Multiprocessor (3D-OCM) system design is becoming one of the hottest topics in the integrated circuit industry ; it is shown that a large number of components (processors and memories) can be placed on 3D-OCM [165]. When communication flow is not properly planned, OCM can experience performance degradation. Therefore, reducing communication flow between these components is critical for OCM performance. To this end, effective modeling and decomposition of computational problems are critical.

Our motivation is to provide a method for the decomposition of computational ap-

plication that can be used in (1) new OCM design, when the computational problems are taken at the functional specification level (early phase of the design process) ; and (2) extending compilers to decompose computational problems for existing OCM. The proposed method allows to (1) minimize communications among the components of OCM ; (2) provide a load balancing mechanism to balance communications among the components of OCM ; and (3) help finding an optimal number of nodes for OCM design.

4.1.2 Approach

Our method consist of (1) transforming a computational application of OCM into an incidence matrix where the rows i of the matrix represent the functions or processes, the columns j represent the data processed by these functions and the entries a_{ij} of the matrix represent the total interactions between functions and their processed data ; (2) transforming the incidence matrix (A) into an interaction matrix (IA) that defines interactions between sets of processes (process groups) and sets of data (data families) ; (3) making use of IA to define an optimal number of process groups (processors) and data families (memories) of OCM.

In (1) syntactical and semantical analyses are performed to determine the interactions between each function and each data of the computational problem in order to produce the incidence matrix ; this step is out of the scope of this paper. Nevertheless, the incidence matrix can be obtained with minor modification of a conventional compiler. This matrix is considered in our proposed method as given and is used as input. In (2), making use of the incidence matrix and for a given number of cells K, we rearrange the rows and the columns of the incidence matrix in order to cluster the processes and data into K cells in a way that (i) functions and data that have higher interactions are placed in the same cell ; (ii) the intra-cell communications are maximized ; and (iii) inter-cell communications are minimized ; a load balancing factor is added to balance the load among the cells. In (3), based on the incidence matrix, K is varied in order to find an optimal number of cells for a given computational application.

4.1.3 Contribution

Our contributions, in this paper, can be summarized as follows :

1. *Generalization from zero – one to Integer Cell Formation Problem* : We propose a new multi-phase decomposition solution of applications called Integer Cell Formation Problem (ICFP) based on zero-one Cell Formation Problem [5]. More specifically, the proposed ICFP transforms the specification of a given application into incidence matrix (A) and rearranges the columns and the rows of A based on similarity rules to produce in interaction matrix (IA).

2. *Load balancing* : The key goal of any Cell Formation Problem algorithm is to minimize the number of inter-cell communications represented by exceptional elements (e.g ., [7], [38], [142], [182]) and maximize the number of intra-cell communications. This goal leads often to unbalanced cell loads that can cause performance degradation of OCM. Thus, we introduce a load-balancing factor to measure the loads of the different cells in order to penalize the rearrangement of the rows and columns that produces imbalanced cells (inters of load).

3. *Experiments* : We implement in C++ the proposed ICFP model to demonstrate the effectiveness of our solution.

The minimization of inter-cell communications and the balancing of the cell loads are conflicting objectives ; the load-balancing factor adds more complexity to an already complex problem. It has been shown that Cell Formation Problem is an NP-hard combinational computer problem [16]. In this paper, we propose metaheuristic methods to provide a polynomial time resolution for the application decomposition problem.

4.1.4 Paper Organization

This paper is organized as follows. Section 4.2 presents related work. Section 4.3 describes the details of the proposed ICFP ; it presents first Cell Formation Problem and then Integer Cell Formation Problem with load balancing factor. Section 4.4 presents a

use case of the decomposition of a randomly generated incidence matrix. Section 4.5 presents a practical application of ICFP to determine an optimal number of components for OCM design given an application specification. Section 5.9 concludes the paper and presents future work.

4.2 Related Work

The problem of decomposition in computer system design has been extensively studied and various methods have been proposed. Two major groups of approaches have been identified : (1) hardware and software co-design (HSC) decomposition for embedded systems [73] and (2) the decomposition is included in the layout process of OCM design, where it places components on integrated circuit. Both groups of approaches aim at partitioning computational applications via a transformation into graphs or hypergraphs [110] ; they use, in general, heuristic or metaheuristics algorithms to perform this transformation. The graphs or hypergraphs are planar (2D) and represent the application for OCM design, where the vertices are the circuit elements and the edges are the connecting links [110].

Since most of the existing decomposition solutions are proposed in their own environment and that there are no benchmarks to measure their performance, it is impossible to compare these solutions to one another as stated by López in [116].

Nevertheless, it is shown that hardware-software decomposition can be either fined grained or coarse grained [116]. In COSYMA project [59], fine grained decomposition is applied to an application, considering firstly all the application as software, and then extract hardware parts from it, using simulated annealing on graph syntax. Another fine grain solution is VULCAN [73] where the application, specified as hardware application in HardwareC is decomposed by iterative extraction of the software parts. Coarse grained decomposition concentrates on high level of abstraction and is more flexible than the fine grained solutions [175], [96], [153]. Henkel [79] presents a more complete study on HSC decomposition where coarse grain and fine grain solutions are combined with multi objective functions.

In [120], a solution is proposed to improve the decomposition by successive refinements with the intervention of the designers, allowing engineers to correct the mistakes and limitations of tools. This approach is not error immune because the designers can introduce new errors at each intervention or simply change the functional specification of the system.

To use the full potential of gigascale OCM and with the era of three dimension integrated circuits, it is important to plan at specification level how the computational application can be distributed on all the components of OCM ; therefore, new decomposition approaches are needed. In this paper, we propose a new decomposition method, based on Cell Formation Problem, which minimizes communications among the components of OCM, provides a mechanism to balance work load on the components (processors and memories) of OCM and helps finding an optimal number of nodes for OCM design. We believe that the proposed approach can significantly improve OCM performance.

4.3 Computation Application Decomposition Using Cell Formation Problem

In this section, we present the decomposition method based on Cell Formation Problem (CFP). We propose a hybrid approach that combines a local search method, called "destroy and rebuild", and a genetic algorithm (GA). The algorithm 1 shows the steps of the resolution. In this algorithm, an application specification is required as input ; this input is analyzed to to determine dependencies between processes and data of the application. An incidence matrix is generated as a result of this analysis ; an entry of this matrix represents the number of times a process accesses a data in the application. Before the decomposition of the incidence matrix, we analyse whether it is decomposable or not. This analysis can be done by the χ^2 distribution. If the incidence matrix is decomposable, we use our proposed method to decompose it. Otherwise the application is computed by a conventional general purpose computer.

Algorithm 1 *Application decomposition for OCM Architecture*

INPUT Application specification
Analysis of the dependency between processes and data of the application
Generate the corresponding incidence matrix A
Evaluate whether A is decomposable
if A is decomposable **then**
    Generate the initial solution
    Generate population based on the local search algorithm
    Apply the genetic algorithm to the set of the best solution
    Evaluate effectiveness of the current solution
    if the solution is optimal **then**
        Stop
    else
        Goto 7
    end if
else
    Compute the application with a superscalar computer.
end if

4.3.1 Cell Formation Problem

Cell Formation Problem is basically a zero-one integer programming model used in reducing the movements of parts or products between machines that process them, and consequently the throughput time. It consists of clustering into cells the incidence matrix entries based on some similarity rules [121]. This method consists of two key operations : (1) machine-group formation and (2) part-family formation. The machine-group formation puts together different machines in order to manufacture one or more part-families. The part-family formation joins parts or products together based on some similarity functions that take into account location, geometry, load, and/or processing requirements. The performance of the cell formation process depends on how the machine-cells are formed.

To formulate Cell Formation Problem, we refer to the incidence matrix A containing the processing information of the manufacturing requirements. Its entries a_{ij} are either 0 or 1. If $a_{ij} = 1$, then the machine i processes the part j, while $a_{ij} = 0$ means otherwise. Figure $1.(a)$ illustrates an incidence matrix with seven machines and eleven parts. Cell

Formation Problem can be reduced to find a block diagonal form in incidence matrix $A[a_{ij}]$. These blocks are built by appropriately reorganizing the columns and the rows of the incidence matrix. Figure 4.1.(b) illustrates the resulting matrix after appropriate rearrangement of the columns and rows.

In the following, we generalize the zero-one Cell Formation Problem to Integer Cell Formation Problem. McCormick et al [122], present a model of the generalization of the zero-one Cell Formation Problem using a bound energy algorithm (BEA) to permutated rows and columns in order to maximize the value of array elements. Arabie and Hubert [8] revisited McCormick et al's BEA approach and have come to the conclusion that the algorithm works as a serial operation where once the rows and columns of the incidence matrix have been arranged, a second phase of partitioning the result is needed to have an appropriate solution. Arabie and Hubert have pointed out [8] that McCormick et al, "had no rigorous approach to partitioning the models but instead relied on visual inspection and subjective judgment". The main objective of Arabie and Hubert [8] was to present the taxonomy of the type of data of the incidence matrices and the structure of these matrices used in different applications where the BEA algorithm has been used. They discussed the properties and the limitations of the objective function of the bound energy algorithm. In [9], Arabie et al, present a variant of McCormick's bound energy algorithm where an additional step of partitioning is performed once the rows and columns rearrangement is carried out.

In the following, we present a more complete and effective all-in-one approach to solve the generalization of the zero-one Cell Formation Problem.

4.3.2 Appling the Cell Formation Problem Approach for Application Decomposition

In the rest of the paper the terms function and process are equivalently used.

The incidence matrix $A = [a_{ij}]$ represents a computer system application specified with m functions, $i = 1,\ldots,m$, and n data, $j = 1,\ldots,n$; where each entry a_{ij} is an integer denoting the number of times function i accesses data j by a load-store instruction. The goal is to form computational units called cell (each including a group of functions and

data families) that are as *autonomous* as possible. Ideally, we would like that all data processed by a function group to be in the data family belonging to the cell. However, this type of situation is uncommon in the real world since there are communications between data and functions from different cells ; these data and functions are called exceptional elements. An exceptional function accesses more than one family of data, and an exceptional data requires access by processes that belong to more than one cell. Maximizing autonomy therefore corresponds to minimizing the number of exceptional data and function communications.

As Boctor's Matrix shows in Figure 4.1, for a small example, that a quick visual inspection can easily detect the cells, the groups and the families. For more complex applications with a large number of entries, it is important to develop analytical procedures or mathematical models for the problem ; these models are based on some similarity functions derived from the application requirements.

In this paper similarity functions, between two data, measure how similar two data are in terms of process accesses whereas similarity functions, between two processes measures how similar these processes are in terms of data processed. Similarity functions based approach for cellular formation has been proposed by McAuley [121] ; it is based on Jaccard similarity coefficients.

In the following, a machine is seen as *process* or *function* and product or part is seen as *memory element* or *data*.

4.3.3 Algebric Notations

TABLE 4.I represents the notations and definitions of symbols and variables used in this paper.

4.3.4 Problem Formulation

Given an application specification A, a cardinal number of a set K representing the cell number, and a NoC structure, what is the best possible decomposition of A, running on the NoC structure that can give the best execution performance ?

Table 4.I – Variables Used in the Cell Formation Procedure

Symbol	Description
m	Number of process in the application
n	Number of data in the application
I	Process set $1,...,m$ representing all processes
J	Data set $1,...,n$ representing all data
A	Incidence matrix $(m \times n)$ with entries a_{ij}
IA	Interaction matrix $(K \times K)$
K	Number of cell $1,...K$
C_k	k-th process group
F_k	k-th data family
C	Process-group $(C_1,...,C_K)$
F	Data-family $(F_1,...,F_K)$
(C_k,F_k)	Process and data included in the k-th cell
(C,F)	Solution $((C_1,F_1),...,(C_K,F_K))$
x_{ik}	Process membership boolean variable
y_{jk}	Data membership boolean variable
$d_{ijk}(x,y)$	Remote access function of data j by process j
(C^0,F^0)	Initial solution $((C_1^0,F_1^0),...,(C_K^0,F_K^0))$ for heuristic procesdure
INA	"Is Not Assigned" Process subset to be assigned to a group as initial solution

The model intends to simultaneously maximize the performance of the network by reducing the inter-cellular communication and increase the intra-cellular communication while balancing the load of the cells.

To formulate the generalized Cell Formation Problem, consider the following two sets :

$I =$ set of m functions,

$J =$ set of n data.

The application incidence matrix $A = \left[a_{ij}\right]$ indicates the interaction between functions and data where a_{ij} indicates the number of times load-store instructions are performed on a data by a function.

$$a_{ij} \begin{cases} \geq 1, & \text{if function } i \text{ process data } j \quad (4.1a) \\ = 0, & \text{otherwise.} \quad (4.1b) \end{cases}$$

Furthermore, data j may be processed by several functions. A computation cell k $(k = 1,...,K)$ includes a subset (group) of functions $C_k \subset I$ and a subset (family) of data $F_k \subset J$. The problem is to determine a solution including K computation cells $(C,F) = \{(C_1,F_1),...,(C_K,F_K)\}$ as *autonomous* as possible ; thus the K cells induce partitions of the function set and of the data set : $C_1 \cup ... \cup C_K = I$ and $F_1 \cup ... \cup F_K = J$, and for all pairs of different cell indices k_1 and $k_2 \in \{1,...,K\}$, $C_{k_1} \cap C_{k_2} = \emptyset$ and $F_{k_1} \cap F_{k_2} = \emptyset$. In Figure 4.1, we use the zero-one matrix for illustration. The second matrix indicates a partition into 3 different cells illustrated in the gray boxes. The solution includes the 3 function groups $\{(6,7),(1,2),(3,4,5)\}$ and the 3 data families $\{(4,5,8,10),(1,2,6,9),(3,7,11)\}$.

The *exceptional elements* $(5,4)$ and $(3,1)$ correspond to entries having a value $a_{ij} \geq 1$ that lay outside of the gray diagonal blocks.

To construct the mathematical formulation of the problem, we introduce the following binary variables : For each pair $i = 1,\ldots,m;\ k = 1,\ldots,K$

$$x_{ik} = \begin{cases} 1, & \text{if process } i \text{ belongs to cell } k \quad (4.2a) \\ 0, & \text{otherwise} \quad (4.2b) \end{cases}$$

Machines \ Parts	1	2	3	4	5	6	7	8	9	10	11
1	1	1	0	0	0	1	0	0	0	0	0
2	0	1	0	0	0	1	0	0	1	0	0
3	1	0	1	0	0	0	1	0	0	0	1
4	0	0	1	0	0	0	1	0	0	0	0
5	0	0	1	1	0	0	0	0	0	0	1
6	0	0	0	1	1	0	0	0	0	1	0
7	0	0	0	0	1	0	0	1	0	1	0

(a) Incidence Matrix

Machines \ Parts	4	5	8	10	1	2	6	9	3	7	11
6	1	1	0	1	0	0	0	0	0	0	0
7	0	1	1	1	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	0	0	0	0
2	0	0	0	0	0	1	1	1	0	0	0
3	0	0	0	0	1	0	0	0	1	1	1
4	0	0	0	0	0	0	0	0	1	0	0
5	1	0	0	0	0	0	0	0	1	1	1

(b) Solution Matrix

Figure 4.1 – Boctor's Matrix

For each pair $j = 1,\ldots,n;\ k = 1,\ldots,K$

$$y_{jk} = \begin{cases} 1, & \text{if data } j \text{ belongs to cell } k \quad (4.3a) \\ 0, & \text{otherwise} \quad (4.3b) \end{cases}$$

From (2) and (3) we deduce that

$$x_{ik}y_{jk}a_{ij} = \begin{cases} a_{ij}, & \text{if } i \text{ and } j \text{ belong to cell } k \quad (4.4a) \\ 0, & \text{otherwise} \quad (4.4b) \end{cases}$$

To mesure the *autonomy* of the solution in the $zero-one$ version, different mesures have been proposed, and Starker and Khan carry out a comparative study in [147]. The most commonly used mesure [102] by the authors to compare the efficiency of their methods is the following :

$$\frac{a^{In}_{\geq 1}}{a + a^{In}_{0}}$$

where $a = \sum_{i-1}^{P}\sum_{j=1}^{P} a_{ij}$ denotes the sum of entries a_{ij} greater or equal to 1 in the matrix A, $a^{Out}_{\geq 1}$ denotes the sum of *exceptional elements* which are the entries a_{ij} greater than or equal to 1 outside the diagonal gray blocks, and $a^{In}_{\geq 1}$ and a^{In}_{0} are the sum of a_{ij} greater than or equal to 1 and the number of 0 entries in the gray diagonal blocks, repectively.

In our case, the objective function includes an autonomy factor and a load balancing factor. Since the element a_{ij} of the matrix A are integer specifying the level of interaction between function i and data j, the element a^{In}_{0} (indicating the number of 0 entries in the gray diagonal blocs) is not as relevant as in the $zero-one$ version. For this reason, we evaluate the autonomy factor as follow :

$$\text{Autonomy Factor} = \frac{a^{In}_{\geq 1}}{a} \quad (4.5)$$

It is easy to verify that $a^{In}_{\geq 1} = \sum_{k=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}x_{ik}y_{jk}$,

$a^{Out}_{\geq 1} = a - \sum_{k=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}x_{ik}y_{jk}$,

To specify the load balancing factor, we denote by $T = \frac{a}{K}$ the target average number of operations in each cell. Hence the load balancing factor is expressed as follow :

$$\text{Load Balancing Factor} = \sum_{k=1}^{K} \left| T - \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ik} y_{jk} a_{ij} \right| \tag{4.6}$$

It indicates the sum of deviations in the cells from the target value.

The objective function is maximizing

$$\text{Eff} = (AutonomyFactor) - \gamma(LoadBalancingFactor) \tag{4.7}$$

where γ is the load balancing factor indicating the relative importance between the two factors.

The following model $M(x,y)$ the cell partitioning problems solved in this paper summarizes :

$$\text{Max Eff} = \frac{a_{\geq 1}^{In}}{a} - \gamma \sum_{k=1}^{K} \left| T - \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ik} y_{jk} a_{ij} \right| \tag{4.8}$$

Subject to :

$$\sum_{k=1}^{K} x_{ik} = 1, \ \forall i \in \{1,\ldots,m\} \tag{4.9}$$

$$\sum_{k=1}^{K} y_{ik} = 1, \ \forall j \in \{1,\ldots,n\} \tag{4.10}$$

$$\sum_{i=1}^{m} x_{ik} \geq 1, \ \forall i \in \{1,\ldots,m\} \tag{4.11}$$

$$\sum_{j=1}^{n} y_{ik} \geq 1, \ \forall j \in \{1,\ldots,n\} \tag{4.12}$$

$$x_{ik} = 0 \ or \ 1 \ \forall i \ \in \{1,\ldots,m\}; \ and \ \forall \ k \ \in \{1,\ldots,K\} \tag{4.13}$$

$$y_{jk} = 0 \ or \ 1 \ \forall j \ \in \{1,\ldots,n\}; \ and \ \forall \ k \ \in \{1,\ldots,K\} \tag{4.14}$$

Constraint (9) and (10) ensure that each function and each data is assigned to exactly one cell, respectively. Constraints (11) and (12) ensure that each cell includes at least one function and one data. Finally, the variables are binary in (13) and (14). The number of cells K for each problem to its value in the best-known solution, and constraints (11) and (12) eliminate any empty cell is fixed according to the NoC structure.

4.3.5 Local Search Approach

Our local search resolution includes two main procedures that are called successively. For a given current solution, we modify successively the groups of functions and the families of data until no modification is possible. Then we apply a diversification strategy to modify the groups and the families of the current solution in order to search more extensively the feasible domain. To modify the groups, we use successively two different procedures : the first one destroys the current solution by eliminating the groups currently assigned to some machines, and the second one recovers a new feasible solution by assigning these machines to new groups. Note that this strategy can also be applied to modify the families of the current solution. The approach can be summarized in algorithm 2 *Local Search* :

Algorithm 2 *Local Search Algorithm*

1. Generate an initial feasible solution (C^0, F^0).
   Set the current solution $(C, F) = (C^0, F^0)$.
   Go to 2.a.
2. Repeat for *numstart* applications of destroy & recover strategy.
3. Modifing the groups and the families
   3.a. Mofify the part familiy on the basis of the machine groups. **If** no modification is possible, **then** move to the destroy & recover strategy at 4.a. **else** move to 3.b.
   3.b. Modify the machine groups on the basis of the parts families. **If** no modification is possible, **then** move to the destroy & recover strategy at 4.b. **else** move to 3.a.
4. Destroy & Recover Strategy
   4.a. Eliminate the family assigned to some parts, and recover a new feasible solution by assigning these parts to new families. Goto 3.b.
   4.b. Eliminate the group assigned to some machines, and recover a new feasible solution by assigning these machines to new groups. Repeat 3.a.
5. Update the best solution.

Note that the best solution is updated each time a new current solution is generated. The different procedures are now summarized in the following sections.

4.3.5.1 Initial Solution

To generate the initial solution we use a procedure quite similar to the one proposed in [142]. First, we determine K function groups $C_1^0, ..., C_K^0$. The K data families $F_1^0, ..., F_K^0$ are specified on the basis of the K function groups known.
Denote that $a_{i\bullet} = \sum_{j=1}^{n} a_{ij}$ and $a_{\bullet j} = \sum_{i=1}^{m} a_{ij}$ the number of data processed by function i and the number of functions processing j, respectively. To initiate the function groups formation, select the K functions having the largest value $a_{i\bullet}$, and assign them to the different groups C_i^0. Then each of the other function left is assigned to the C_i^0 including functions processing mostly the same data. More specifically, denote *INA* as the set of functions left. The assignments are completed as follows :

1. For all functions $i \in INA$, determine the group

$$\bar{k}(i) = \underset{k=1,...,K}{\text{Min}} \left\{ \frac{1}{|C_k^0|} \sum_{j=1}^{n} \sum_{i_k \in C_k^0} |a_{ij} - a_{i_k j}| \right\} \tag{4.15}$$

$$gr_i = \underset{k=1,...,K}{\text{ArgMin}} \left\{ \frac{1}{|C_k^0|} \sum_{j=1}^{n} \sum_{i_k \in C_k^0} |a_{ij} - a_{i_k j}| \right\} \tag{4.16}$$

2. Determine the function $\bar{i} \in INA$

$$\bar{i} = \underset{i \in INA}{\text{ArgMin}} \{\bar{k}(i)\} \tag{4.17}$$

   and assign $\bar{i}$ to group $C_{gr_{\bar{i}}}^0$; *i.e.*, $C_{gr_{\bar{i}}}^0 = C_{gr_{\bar{i}}}^0 \cup \{\bar{i}\}$.

3. Eliminate $\bar{i}$ from *INA* and repeat 1) until *INA* become empty.

On the basis of the K function groups $C_1^0,\ldots,C_K^0$, determine K data families $F_1^0,\ldots,F_K^0$.
For each data j denote :
$\tilde{a}^{In}_{\geq 1j}(k) = \sum_{i\in C_k^0} a_{ij}$ the number of functions in group k that are processing data j,
$\frac{\tilde{a}^{In}_{\geq 1j}(k)}{a_{\bullet j_t}} + \gamma\left(T - \sum_{\substack{t=1\\ j'\in F_k}}^{n} \tilde{a}^{In}_{\geq 1j'}(k) - \tilde{a}^{In}_{\geq 1j}(k_t)\right)$ an approximation of the impact of assigning data j to family k.

Then each data j is assigned to the family $F^0_{\tilde{k}j}$ where

$$\tilde{k}(j) = \underset{k=1,\ldots,K}{\text{ArgMax}}\left\{\frac{\tilde{a}^{In}_{\geq 1j}(k)}{a}\right\}$$

in order to generate a good initial solution (C^0,F^0) having the grouping efficiency

$$\text{Eff} = \frac{\sum_{j=1}^{n} \tilde{a}^{In}_{\geq 1j}\left(\tilde{k}(j)\right)}{a} - \gamma\sum_{k=1}^{K}\left|T - \sum_{i=1}^{m}\sum_{\substack{j=1\\ C_k=F_k}}^{n} a_{ij}\right|. \tag{4.18}$$

Note that if some family F_k^0 is empty, then we apply the *repair process* to reassign one data to it inducing the smallest decrease of the grouping efficiency.
Then this initial solution becomes the current solution (*i.e.*, $(C,F) = (C^0,F^0)$), and we initialize the modification procedure by moving to the function groups on the basis of the data families.

4.3.5.2 Modifying the Function Groups and Data Families

Consider the current solution (C,F). The procedures to modify the function groups on the basis of the data families and to modify the data families on the basis of the function groups are similar to the process to determine the data families in the preceding section where we generate the initial solution. For the sake of completeness, let us summarize the procedure to determine the new function groups $\bar{C}_1,\ldots\bar{C}_K$ on the basis of the

data families $F_1,\ldots,F_K$. For each function i denote :
$\bar{a}^{In}_{\geq 1i}(k) = \sum_{j\in F_k} a_{ij}$ the number of data in group k that are processed by function i,
$\frac{\bar{a}^{In}_{\geq 1i}(k)}{a_{\bullet i_t}} + \gamma\left(T - \sum_{\substack{t=1\\ i'\in C_k}}^{n} \bar{a}^{In}_{\geq 1i'}(k) - \bar{a}^{In}_{\geq 1i}(k_t)\right)$ an approximation of the impact of assigning function i to group k.
Then each function i is assigned to the group $\bar{C}_{\bar{k}(i)}$ where

$$\bar{k}(i) = \underset{k=1,\ldots,K}{\text{ArgMax}}\left\{\frac{\bar{a}^{In}_{\geq 1i}(k)}{a}\right\} \tag{4.19}$$

in order to generate a good initial solution $(\bar{C},F)$ having the grouping efficiency

$$\text{Eff} = \frac{\sum_{i=1}^{m} \bar{a}^{In}_{\geq 1i}\left(\bar{k}(i)\right)}{a} - \gamma\sum_{k=1}^{K}\left|T - \sum_{i=1}^{m}\sum_{\substack{j=1\\ C_k=F_k}}^{n} a_{ij}\right|. \tag{4.20}$$

Note that if some group $\bar{C}_k$ is empty, then, we apply the *repair process* to reassign one function to it including the smallest decrease of the grouping efficiency. Now, if the function groups remain identical (*i.e*, $C_k = \bar{C}_k, k = 1,\ldots,K$), then we cannot modify the solution with this modification process. In this case, we move to appling the destroy & recover strategy in step 4.b. of the algorithm 2. Otherwise the new current solution is obtained by replacing C_k by $\bar{C}_k, k = 1,\ldots,K$.

4.3.5.3 The procedure to determine the new data families $\bar{F}_1,\ldots,\bar{F}_K$ on the basis of the function groups $C_1,\ldots,C_K$

As in Section 4.3.1, we determine for each data j
$\tilde{a}^{In}_{\geq 1j}(k) = \sum_{i\in C_k} a_{ij}$ the number of functions in group k that are processing part j,

$\frac{\tilde{a}^{In}_{\geq 1 j}(k)}{a_{\bullet j}}$ an approximation of the impact of assigning data j to family k.
Then each data j is assigned to the family $\bar{F}_{\tilde{k}(j)}$ where

$$\tilde{k}(j) = \underset{k=1,\ldots,K}{\text{ArgMax}} \left\{ \frac{\tilde{a}^{In}_{\geq 1 j}(k)}{a_{\bullet j}} \right\}$$

in order to generate a good initial solution $(C, \bar{F})$ having the grouping efficiency

$$\text{Eff} = \frac{\sum_{j=1}^{n} \tilde{a}^{In}_{\geq 1 j}\left(\tilde{k}(j)\right)}{a} - \gamma \sum_{k=1}^{K} \left| T - \sum_{i=1}^{m} \sum_{\substack{j=1 \\ C_k = F_k}}^{n} a_{ij} \right|. \tag{4.21}$$

Note that if some family $\bar{F}_k$ is empty, then we apply the *repair process* to reassign one function to it inducing the smallest decrease of the grouping efficiency. Now, if the data families remain identical (*i.e.*, $F_k = \bar{F}_k,\ k = 1, \ldots, K$), then we cannot modify the solution with this modification process. In this case, we move to apply the *destroy&recover* strategy in step 4.a of algorithm 2. Otherwise the new current solution is obtained by replacing F_k by $\bar{F}_k,\ k = 1, \ldots, K$.

In the conventional $zero - one$ cell formation problem, Ng proposes in [127] a procedure to move parts on the basis of machine group or to move machines on the basis of part families in the spirit of our proposed procedure. On the one hand, Ng's procedure is an ascent method since parts or processes are moved only when the group efficiency increases. On the order hand, our procesure is simpler to implement, but the moves do not necessarily induce an increase of the grouping efficiency. This strategy allows a better diversification to search more extensively the feasible domain.

4.3.5.4 Destroy&Recover Strategy

This strategy is in the spirit of the large neighborhood search presented by Shaw, where a large number of variables are modified simultaneously. It also relates to other implementations introduced by Pisinger and Ropker ([140]), Schrimpf *et al* in [148], and

by Dees and Karger in [47].

In step 4.a., the procedure is applied to modify the assignment of $q = \lceil \%n \rceil$ data on the basis of the function groups. The basic principle is to select q data that are moved to alternate families in order to reduce the grouping efficiency as little as possible. First, for data j, we determine

$$f_{jk} = \left(\frac{\tilde{a}^{In}_{\geq 1 j}(k) - \tilde{a}^{In}_{\geq 1 j}(\tilde{k}(j))}{\tilde{a}^{In}_{\geq 1 j}(\tilde{k}(j))} \right) - \left(\frac{|T_k - \tilde{a}^{In}_{\geq 1 j}(k)| + |T_{\tilde{k}(t)} + \tilde{a}^{In}_{\geq 1 j}(\tilde{k}(j))| - |T_k| + |T_{\tilde{k}(t)}|}{2T} \right)$$

, $k = 1, \ldots, n$, $k \neq \tilde{k}(j)$

$$ok(j) = \underset{k \neq \tilde{k}(j)}{\text{ArgMin}} \left\{ f_{jk} \right\}$$

$$f_j = \underset{k \neq \tilde{k}(j)}{\text{Min}} \left\{ f_{jk} \right\} = f_{jok(j)}$$

Then select the q data $j_1, \ldots, j_q$ having the smallest value of f_j, and modify their families as follows :

$F_{\tilde{k}(j_i)} = F_{\tilde{k}(j_i)} - \{j_i\}$ and $F_{\tilde{ok}(j_i)} = F_{\tilde{ok}(j_i)} \cup \{j_i\}$.

Now if any family is empty, then the repair process described before can be applied to introduce data in it. Furthermore, this new current solution is used to initialize step 3.b. This procedure is adapted to modify $q = \lceil \%m \rceil$ on the basis of part families. Then the new current solution is then used initialize the step 3.a.

4.3.6 Hybrid Method with a Genetic Algorithm

Even if the numerical results of our method in the $zero - one$ approach indicate that the local search procedure generates quickly good results, we hybridize this method with a genetic algorihm procedure in order to search more extensively the feasible domain. The hybrid method is a steady state genetic algorithm where two offspring solutions are generated at each generation. The local search procedure is then applied to improve each of these new offspring solutions. The procedure is summarized as follows :

Algorithm 3 *Hybrid Method*

Generate an initial population of S feasible solution.
for *nag* generation **do**
1. Select two parent populations from S.
2. Perform a crossover operation with probability *pcross* to generate two new offspring solutions.
3. If necessary, apply the repair process to each offspring solution to insure that no group or no family is empty.
4. Perform a mutation operation on each offspring solution with probability *pm*.
5. Apply the local search to improve each offspring solution.
6. Update the population by keeping the best $|S|$ solutions from the current population and the two improved offspring solutions generated.

end for

First, the initial solution generated by the local search procedure of the algorithm, described in section 4.3. is included in the population S. To generate each of the other solutions, we select randomly to generate the function groups or the data families with each alternative having a probability of 0.5. In the first case, each function is randomly assigned to a group k. We also prevent each group from being empty by applying a repair process to move a function from the group including the most to the empty group. Then, the data families are determined on the basis of the K function groups as in the initial solution section. The local search procedure is applied to improve the solution which is included in the population. The procedure to complete the second alternative is similar. The role of functions and data are exchanged.

To complete the genetic algorithm, a proper encoding of the solutions is required. A feasible solution $(C,F) = \{(C_1,F_1),\ldots,(C_K,F_K)\}$ is encoded as vector having $(n+m)$ components $(P_1,\ldots,P_n,M_1,\ldots,M_m)$ where P_j is the index of the family including part $j,\ j = 1,\ldots,n$ and M_i is the index of the group including process $i,\ i = 1,\ldots,m$.

Two parent solutions are selected according to a tournament where 4 individuals are selected randomly from the population S, and the best solution becomes the first parent solution. The second parent solution is selected similarly.

To determine the two offspring solutions, a *uniform crossover* is completed. More

specifically, suppose that the two parent solutions are

$$(P_1^1, \ldots, P_n^1, M_1^1, \ldots, M_m^1)$$

and

$$(P_1^2, \ldots, P_n^2, M_1^2, \ldots, M_m^2)$$

.

Algorithm 4 *Genetic Algorithm*

Generate a crossover mask vector of bits having $(n + m)$ elements $(B_1, \ldots, B_n, B_{n+1}, \ldots, B_{n+m})$.
The offspring solutions
$(OP_1^l, \ldots, OP_n^l, OM_1^l, \ldots, OM_m^l,)$, $l = 1, 2$, are specified as follows :
for $j = 1$ to n **do**
  if $B_j = 1$ **then**
    $OP_j^1 = P_j^1$ and $OP_j^2 = P_j^2$
  end if
  if $B_j = 0$ **then**
    $OP_j^1 = P_j^2$ and $OP_j^2 = P_j^1$
  end if
end for
for $i = 1$ to m **do**
  if $B_{n+i} = 1$ **then**
    $OM_i^1 = M_i^1$ and $OM_i^2 = M_i^2$
  end if
  if $B_{n+i} = 0$ **then**
    $OM_i^1 = M_j^2$ and $OM_i^2 = M_j^1$
  end if
end for

A repair process is applied to introduce a part or a process in any empty family or group.

The mutation operator is also applied to modify slightly an offspring solution. Four different elements are selected randomly : a part $\tilde{i} \in \{1, \ldots, n\}$, a family $\tilde{k} \in \{1, \ldots, K\}$, a process $\bar{j} \in \{1, \ldots, m\}$, a group $\bar{i} \in \{1, \ldots, K\}$.

Then the part $\tilde{j}$ is moved to the family $F_{\tilde{k}}$, and the process $\bar{i}$ to the group $C_{\bar{k}}$. Note that the elements are selected to avoid creating empty family groups. Finaly the local search

procedure is applied to each offspring solution.

4.4 Experiments

In this Section, we use ICFP to solve the decomposition of the incidence matrix shown in Figure 4.2, for simplicity, the entries of the incidence matrix are randomly generated. The transformation of the computed application into an incidence matrix is out of the scope of this paper. The objective of the experiments is to (1) illustrate the decomposition of an incidence matrix and (2) show the impact of the load balancing factor γ on the cell load.

The incidence matrix (see Figure 1) contains 41 functions and 27 data ; the experiments have been implemented in C++ and run using Linux-based machine (AMD Sempron (t) Processor 3100, clock 1GHz, 256KB of cache, and 1GB of RAM).

The result of the decomposition of the incidence matrix into $K = 6$ cells with $\gamma = 1$ is illustrated in Figure 4.4. We observe that the large values of the incidence matrix are now concentrated in the six cells represented by the diagonal of the matrix (C_i, F_j), where $i = j$; this is expected since the rows and columns have been rearranged by ICFP.

The relationship between the cells and the exceptional elements is expressed by the $K \times K$ contingency matrix illustrated in Figure 4.4. Recall that the exceptional elements are the non-zero entries of the interaction matrix that are outside the cells. The entries of the contingency matrix are the sum of the entries of the cells of the resulting matrix, illustrated in Figure 4.3, and the entries of the exceptional elements are delimited by the boundaries of the cells (rows and columns) in the same Figure.

Now, let us study the impact of the cell number K on the exceptional elements, the variation of cell loads, and the exceptional elements assuming a load balancing factor γ. The matrix (see Figure 4.4) shows the communication weight between the processing elements C_i and between the memory elements F_j of a cell k. The diagonal of the contingency matrix illustrates the closeness of the loads in each cell ; this is expected since load balancing factor γ is set to 1. We observe that the summations of the rows C_i are relatively close to one another and so are the summations of the columns F_j. This illus-

trates the fact that the exceptional elements have relatively low impact on the processor and memory loads

TABLE 4.II illustrates the grouping efficiency (Eff) and the execution time (ET), expressed in seconds. Its entries show that when K increases Eff decreases ; more specifically, Eff decreases rapidly from $K = 1$ to $K = 4$ and slowly from $K = 5$ to 8 ;. However, the execution time (ET) grows exponentially, as expected.

4.5 Application : Selection of the Proper Number of K Cell for an OCM Design

In the following, we use our proposed ICFP decomposition approach to analyze the problem of how to find, given aa application specification the proper number of nodes $(2K)$ for effective OCM design. Recall that the specification is transformed into an incidence matrix after it has been analyzed. The incidence matrix (A) is checked whether it can be decomposed or not.

We use the same random entries incidence matrix shown in Figure 4.2 ; each computed result matrix (e.g. see Figure 4.3) with different cell numbers is transformed into a contingency matrix (e.g. see Figure 4.4), also called Interaction Matrix (IA).

4.5.1 Variation of K Number of Cells

The objective is to determine the optimal number of cells, K, by varying K ; indeed, we aim at finding an optimal solution for the decomposition of a given incidence matrix (A). We run our method on the incidence matrix while varying K from 1 to 10 ; as shown in Figures 4.3 and 4.4, we compute the sums of the cells after the genetic algorithm execution ; the results are summarized in TABLE 4.II.

Figures 4.5 to 4.7 illustrate the search of K. For each interaction matrix IA (see Figure 4.4. Recall that all the cells as well as the exceptional elements are summed up according to the boundary of each cell. Figures 4.5 and 4.6 are illustrations of the impact of the variation of K from 1 to 10 on family loads and group loads respectively. TABLE

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}	d_{17}	d_{18}	d_{19}	d_{20}	d_{21}	d_{22}	d_{23}	d_{24}	d_{25}	d_{26}	d_{27}
f_1	0	0	0	0	0	0	0	0	30	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	70
f_2	0	0	111	0	0	0	0	0	0	0	0	0	0	77	0	0	0	0	0	0	2	0	0	0	0	102
f_3	0	90	93	0	0	0	14	0	0	0	0	0	0	0	0	0	0	14	0	0	63	0	61	0	0	0
f_4	0	105	0	0	0	0	0	0	0	0	84	0	0	0	0	0	99	0	0	0	92	0	0	0	0	0
f_5	6	0	0	0	0	0	0	8	15	5	98	0	0	0	0	0	111	0	0	0	0	0	86	0	0	0
f_6	0	93	0	0	0	0	0	0	0	0	71	0	0	0	0	0	0	0	15	51	57	0	38	0	0	0
f_7	0	56	0	0	0	0	0	0	0	0	89	0	60	0	0	0	0	0	0	0	76	0	0	0	0	0
f_8	0	0	20	0	0	0	0	0	0	0	90	0	0	0	0	0	0	0	0	0	54	0	59	0	0	0
f_9	0	81	0	0	0	0	0	7	0	0	83	0	0	0	0	0	0	0	0	0	77	105	67	0	0	0
f_{10}	0	87	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	0	0	10	12	0
f_{11}	0	0	0	0	36	0	0	0	0	0	0	0	63	0	0	0	0	0	0	56	0	98	0	0	0	76
f_{12}	0	0	0	0	0	78	0	0	0	0	0	84	43	0	0	0	0	0	0	54	0	87	0	0	0	0
f_{13}	0	0	0	0	0	0	0	0	11	0	0	0	73	0	0	0	0	0	0	98	0	59	0	0	0	0
f_{14}	0	0	0	0	55	67	0	0	0	0	0	69	56	0	0	0	0	0	0	0	0	0	0	0	1	0
f_{15}	0	0	0	0	76	77	0	0	0	0	0	0	97	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{16}	0	0	0	0	73	0	0	0	0	0	0	0	67	4	0	0	0	0	0	56	0	0	0	0	0	0
f_{17}	0	0	0	0	45	0	0	0	0	0	6	0	0	0	0	0	0	0	0	76	0	23	0	0	0	0
f_{18}	67	0	0	0	0	0	0	0	0	0	0	34	0	0	0	0	0	0	55	0	0	0	0	80	0	0
f_{19}	0	0	0	0	0	0	0	70	0	0	0	66	0	0	7	0	0	78	0	0	0	0	0	87	65	0
f_{20}	87	0	5	0	0	0	65	0	0	0	0	0	0	0	0	0	7	61	0	0	0	0	0	76	0	0
f_{21}	54	0	0	0	0	0	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78	0
f_{22}	67	9	0	78	0	0	0	0	0	87	0	0	0	0	0	0	0	150	0	0	0	0	0	0	45	0
f_{23}	98	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	75	0	0	0	0	0	0	0	0
f_{24}	76	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0	0	90	0	0	0	0	0	0	0	0
f_{25}	67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78	0	0	0	0	0	45	0
f_{26}	0	0	0	76	0	0	0	0	0	0	0	0	0	0	15	0	0	0	95	0	0	0	0	78	0	0
f_{27}	0	0	0	0	0	0	0	0	0	0	10	76	0	0	0	0	0	0	0	0	0	0	0	0	107	100
f_{28}	0	0	22	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	201	0	0	0	0	0	0	0
f_{29}	0	0	0	92	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	20	0
f_{30}	0	0	0	90	0	0	0	0	5	0	0	0	0	0	0	0	0	0	98	0	0	0	0	0	0	0
f_{31}	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	90	0	0	0	8	100	0	0
f_{32}	0	0	0	67	0	20	17	0	0	0	0	0	6	0	0	0	0	0	89	0	0	0	0	99	0	0
f_{33}	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	90	0	0	0	0	54	0	0
f_{34}	0	0	0	98	0	0	0	0	0	90	0	0	0	0	0	88	0	0	0	0	4	0	0	0	102	0
f_{35}	0	6	0	0	0	0	0	102	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	8	70	8
f_{36}	0	0	0	0	0	0	0	0	0	92	0	0	0	0	11	110	0	0	0	0	0	0	0	0	0	0
f_{37}	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	98	0	0	7	0	0	0	5	0	78	0
f_{38}	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	90	0	10	0	0	5	0	0	0	0	0
f_{39}	0	0	0	0	0	12	0	90	0	144	0	0	0	0	0	99	0	0	0	0	0	0	0	0	0	0
f_{40}	0	0	0	0	0	0	16	102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	88	10
f_{41}	0	0	0	0	0	0	0	83	0	102	0	0	0	0	0	90	0	0	0	8	0	0	0	0	0	0

Figure 4.2 – Incidence Matrix with 41 Functions and 27 Data

		F1				F2			F3			F4				F5				F6			
		d_2	d_{11}	d_{17}	d_{21}	d_4	d_{19}	d_{24}	d_8	d_{10}	d_{16}	d_{26}	d_1	d_7	d_{12}	d_{18}	d_{35}	d_3	d_9	d_{14}	d_{15}	d_{27}	d_5	d_6	d_{15}	d_{20}	d_{22}
C_1	f_4	105	84	99	92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	f_5	0	98	111	0	86	0	0	8	5	0	0	6	0	0	0	0	0	15	0	0	0	0	0	0	0	0
	f_6	93	71	0	57	38	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	51	0
	f_7	56	89	0	76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0
	f_8	0	90	0	54	59	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0
	f_9	81	83	0	77	67	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	105
	f_{10}	87	0	0	90	0	0	10	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0
C_2	f_{28}	0	0	0	0	76	95	78	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0
	f_{29}	0	0	0	0	90	201	0	0	0	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0
	f_{30}	0	0	0	0	92	0	100	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	10	0	0	0
	f_{31}	0	0	0	0	90	98	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
	f_{32}	0	0	0	0	0	90	100	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	f_{33}	0	0	0	0	67	89	99	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	20	0	0	0
	f_{34}	0	0	0	0	0	90	54	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0
C_3	f_{35}	0	0	0	0	98	0	0	0	90	88	102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	f_{36}	6	0	0	0	0	0	8	102	0	0	70	0	0	0	0	10	0	0	0	0	8	0	0	0	12	0
	f_{37}	0	0	0	0	0	0	0	0	92	110	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0
	f_{38}	0	0	0	0	0	7	0	0	0	98	78	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0
	f_{39}	0	0	0	0	0	0	0	0	120	90	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	5
	f_{40}	0	0	0	0	0	0	0	90	144	99	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0
	f_{41}	0	0	0	0	0	0	0	102	0	90	0	0	16	0	0	0	0	0	0	0	10	0	0	0	0	0
C_4	f_{18}	0	0	0	0	0	55	0	0	0	0	0	67	0	34	0	80	0	0	0	0	0	0	0	0	0	0
	f_{19}	0	0	0	0	0	0	87	70	0	0	0	0	0	66	78	65	0	0	0	7	0	0	0	0	0	0
	f_{20}	0	0	7	0	0	0	76	0	0	0	0	87	65	0	61	0	5	0	0	0	0	0	0	0	0	0
	f_{21}	0	0	0	0	0	0	0	0	0	0	0	54	34	0	0	78	0	0	0	0	0	0	0	0	0	0
	f_{22}	9	0	0	0	78	0	0	0	87	0	0	67	0	0	150	45	0	0	0	0	0	0	0	0	0	0
	f_{23}	0	11	0	0	0	0	0	0	0	0	0	98	0	0	75	0	0	0	0	0	0	0	0	0	0	0
	f_{24}	0	0	0	0	0	0	0	0	0	0	0	76	98	0	90	0	0	0	0	0	0	0	0	0	0	0
	f_{25}	0	0	0	0	0	78	0	0	0	0	0	67	0	0	0	45	0	0	0	0	10	0	0	0	0	0
	f_{27}	0	10	0	0	0	0	0	0	0	0	100	0	0	76	0	107	0	0	0	0	0	0	0	0	0	0
C_5	f_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	50	70	0	0	0	0	0
	f_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	111	0	77	0	102	0	0	0	0	2
	f_3	90	0	0	63	0	0	0	0	0	0	0	0	14	0	14	0	93	0	0	0	0	0	0	0	0	0
C_6	f_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	76	36	0	63	56	98
	f_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	84	0	0	0	0	0	0	0	0	78	43	54	87
	f_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	73	98	59
	f_{14}	0	0	0	0	0	0	0	0	0	0	1	0	0	69	0	0	0	0	0	0	0	55	67	56	0	0
	f_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	76	77	97	0	0
	f_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	73	0	67	56	0
	f_{17}	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	0	0	76	23

Figure 4.3 – Illustation of the Decomposition of the Incidence Matrix using our Proposed Cell Formation Problem

	F_1	F_2	F_3	F_4	F_5	F_6	$\sum C_i$
C_1	1843	25	20	18	35	216	**2157**
C_2	8	1509	50	37	42	36	**1682**
C_3	23	113	1838	36	29	47	**2086**
C_4	37	374	257	1763	22	0	**2453**
C_5	214	0	0	28	533	2	**777**
C_6	6	0	1	153	91	1513	**1764**
$\sum F_j$	**2131**	**2021**	**2166**	**2035**	**752**	**1814**	

Figure 4.4 – Contingency Matrix

Table 4.II – Efficiency and Execution Time

K	1	2	3	4	5	6	7	8	9	10
Eff	10	8.76	6.06	5.07	4.77	4.27	3.05	2.44	1.94	1.80
ET.(sec)	0	1.21	3.95	12.86	16.27	147.8	421.26	864.44	1399.44	2001.19

4.III summarizes the computed minimum and maximum numbers of interactions for group and family, respectively, for $K = 1..10$.

We observe that the behavior of the family and group load can be compared to the function $f(x) = \frac{1}{x}$; an optimal solution corresponds to the minimum number of cells where the gradient of the function is minimal. In Figures 4.5 to 4.7, we observe that from interval $K \in [5,10]$, the steepness of the plots is almost constant. Thus, the best solution corresponds to $K = 5$.

Figure 4.7 shows where the difference between the largest cell load and the smallest cell load is minimal, considering the family and group loads respectively ; balancing the loads of cells is required for better performance.

Based on these two considerations, summarized in Figure 4.7, we can conclude that to design OCM for the specification represented by the incidence matrix A in Figure 4.2, the effective number of cells is 5.

Table 4.III – Variation of Group and Family Sizes

K	1	2	3	4	5	6	7	8	9	10
Max. C_i Size	10919	5928	4135	3154	2494	2453	2157	1737	1764	1646
Min. C_i Size	10919	4991	2934	1764	1682	777	777	440	442	440
Max. F_j Size	10919	5893	4309	3265	2382	2166	2021	1799	1435	1409
Min. F_j Size	10919	5026	2883	1814	2021	752	1084	512	752	636

4.5.2 Impact of Cell Number K on the Exceptional Elements

In addition to the family and group loads, we analyze the impact of the variation of K on the exceptional elements. Recall that the exceptional elements indicate the inter-cell interactions and are the entries of the interaction matrix (IA) that are outside the gray boxes. As we have mentioned previously in the case of the analysis of family and group loads, the smallest slope of the plot describes the best solution area region. In TABLE 4.IV, $\sum Exc$ indicates the sum of all the exceptional elements of the result matrix illustrated in Figure 4.3. The value of $\sum Exc$ is increases when K increases. This can be explained by the fact that when K increases and the cells are being balanced the number of exceptional elements in the matrix increases.

In Figure 4.8, the regions of the interval $[8,9]$ as well as that of $[4,6]$ have the two lowest slopes. Even though the interval $[8,9]$ has the lowest slope, the combination with the information shown in Figures 4.5-4.7, shows that the optimal number of cells for a better decomposition of the incidence matrix is in the interval $[4,6]$. Since our objective is to minimize the number of exceptional elements, an optimal number of cells, in this example, is 5.

Table 4.IV – Impact of Number of Cell on Exceptional Elements

Nb of Cell	1	2	3	4	5	6	7	8	9	10
$\sum Exc.$	0	633	1235	1562	1704	1920	2576	3608	3622	3822

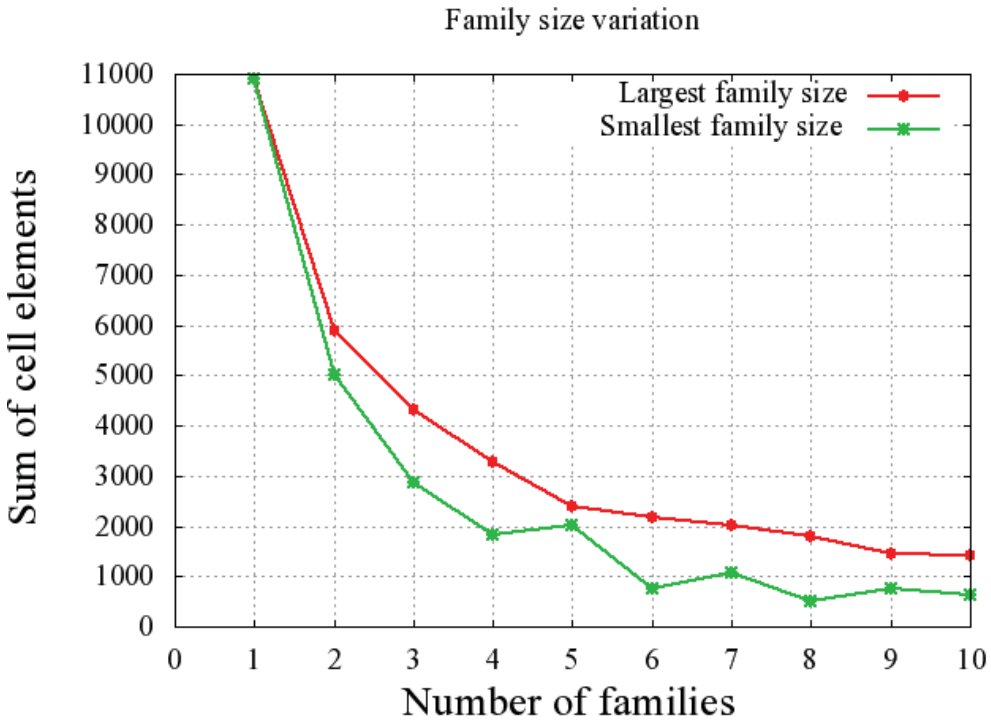

Figure 4.5 – Family Size Variation

4.5.3 Variation of the Load Balancing Factor γ

In this Section, we study the impact of the variation of the load balancing factor γ on the grouping efficiency (Eff). This factor is used to decide which between load balancing and number of exceptional elements is more important for the computation ; e.g., in homogeneous system configurations, the load balance factor is more important than the exceptional elements while in heterogeneous system configurations, reducing the exceptional elements can be more important with imbalance cell loads. In Table 4.V, we summarize the efficiency value of γ between 0 and 1 for the formation of 2 to 10 cells ; $K = 1$ has no exceptional element. Figure 4.9 illustrates the impact of the variation of γ. The slope of the efficiency function decreases for all K as the number of γ grows. In this case, a good solution of γ is the one with the smallest slop (i.e., γ has less impact on the cell formation). Figure 4.9 shows that the solution region can be represented by

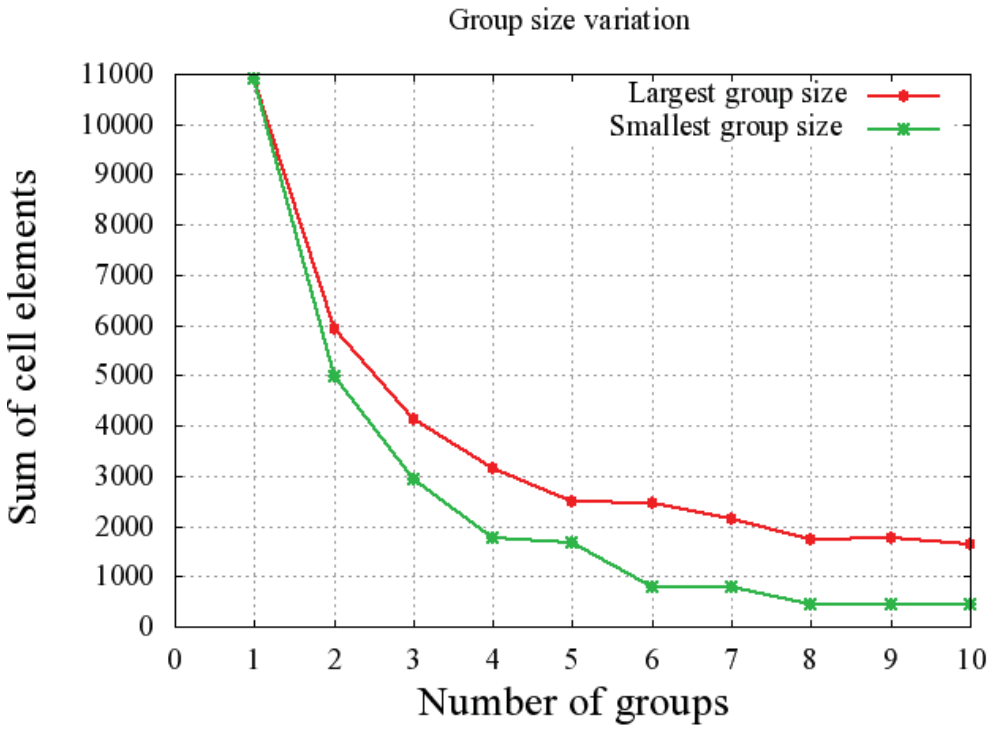

Figure 4.6 – Group Size Variation

cell number K between 3 to 5. This confirms our findings (see above) that the optimal number of cells is 5.

4.6 Conclusion and Discussion

In this paper, we considered a multi-phase cellular formation for application or OCM specification decomposition. A modified formulation of the zero-one Cell Formation Problem to an integer Cell Formation Problem is proposed. This formulation provides a method for solving the legacy application decomposition for parallel or distributed computers or a model for the design of new Application Specific OCM. The method seeks and generates the best solution for a given application. Due to the complexity of

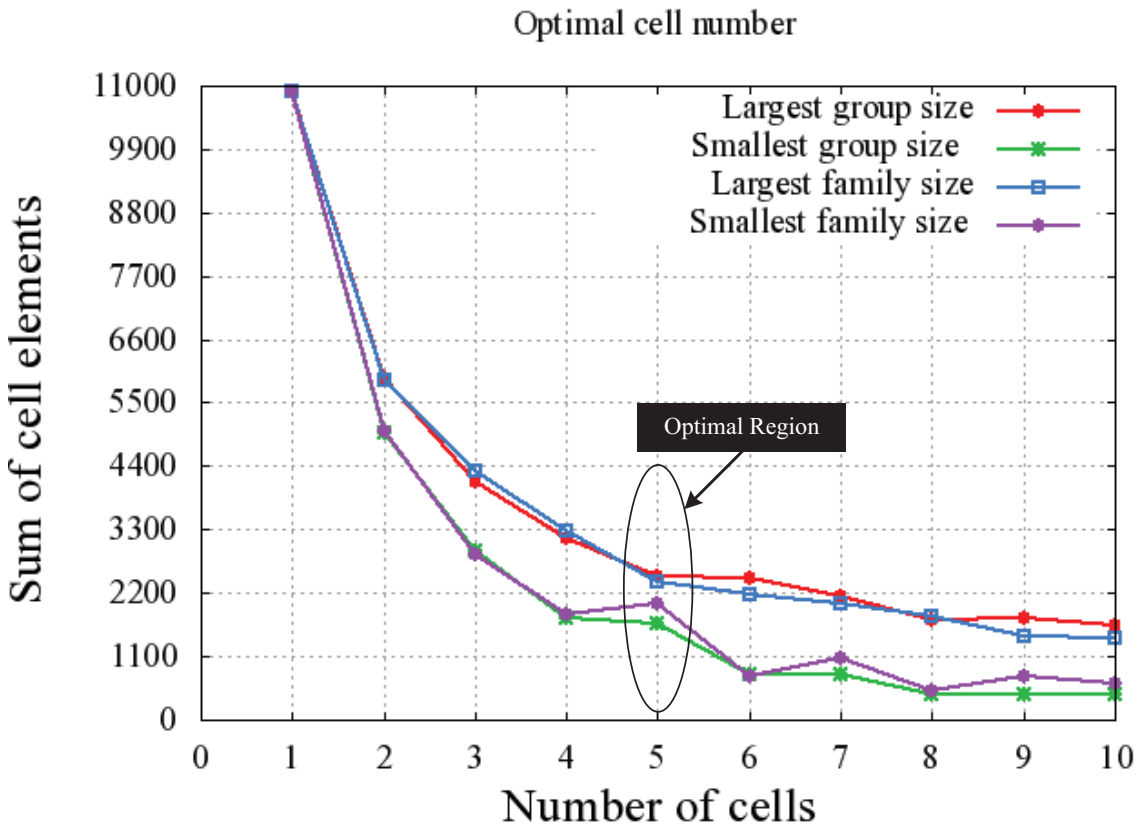

Figure 4.7 – Optimal cell number for the application incidence matrix

the problem, we proposed a three phase's heuristic algorithm where two meta-heuristic models are used to refine the solution. A computational experiment was proposed to illustrate the modeling of an application.

We have used the proposed method to find the effective number of node (K processors and K memories) of a given application. In the case of an existing OCM, the computed application is decomposed based on the number of nodes available in the targeted OCM. Thus, we use the variation of the load balancing factor γ for the exceptional element in the system.

For future research, we are considering to work on cellular formation where a temporal objective in addition to the current objectives specified in this paper. For this objective, a timing coefficient can be defined and the scheduling of the inter-cells commu-

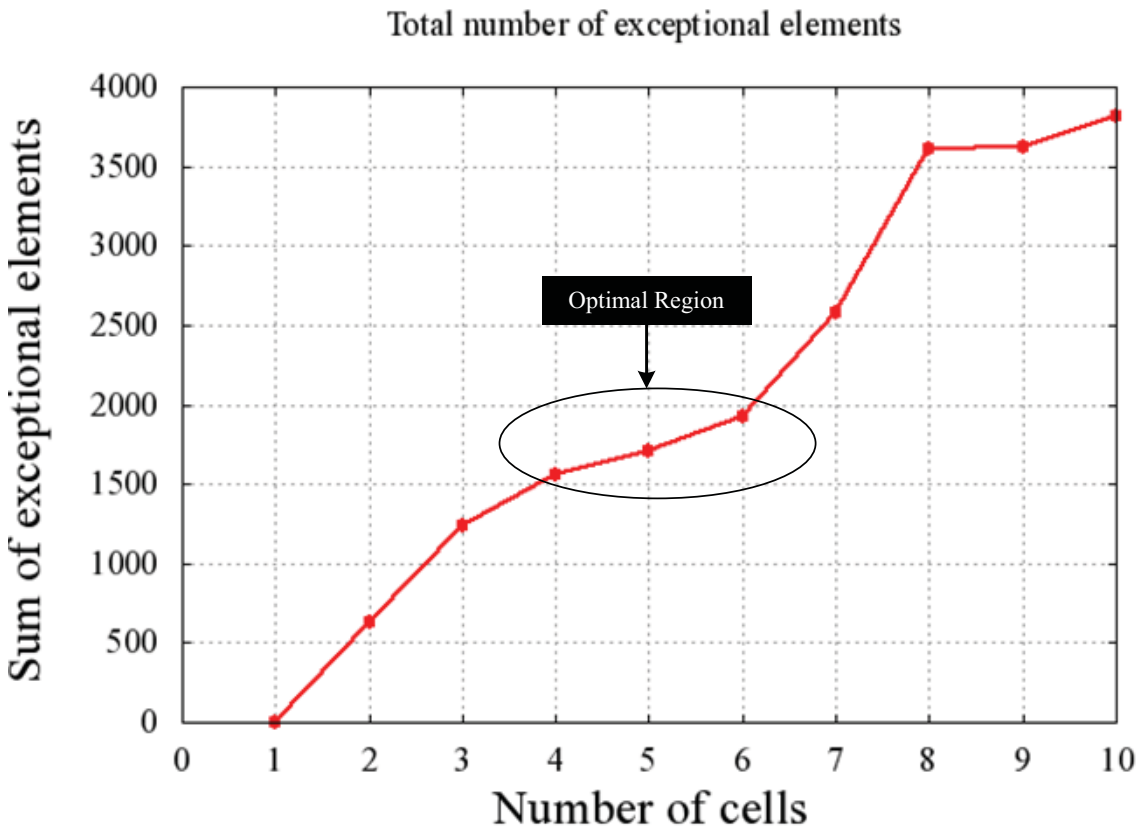

Figure 4.8 – Optimal cell number for the incidence matrix in Table 4.IV

nications should be considered. We are also working on the assignment of the cells of an application (produced by the proposed method) to the three dimensions Network-on-Chip structure in order to maximize the performance while minimizing the inter-node communications and reducing the data transfer in the network.

Table 4.V – Efficiency with penalty factor γ variation

γ	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
$K = 2$	1	0.94	0.93	0.92	0.91	0.90	0.89	0.88	0.88	0.87	0.86
$K = 3$	1	0.89	0.86	0.85	0.84	0.83	0.82	0.81	0.80	0.79	0.77
$K = 4$	1	0.86	0.83	0.81	0.80	0.79	0.77	0.76	0.74	0.73	0.71
$K = 5$	1	0.85	0.81	0.80	0.78	0.77	0.75	0.73	0.72	0.70	0.69
$K = 6$	1	0.83	0.80	0.77	0.75	0.73	0.71	0.70	0.68	0.66	0.64
$K = 7$	1	0.82	0.77	0.73	0.70	0.67	0.64	0.61	0.57	0.55	0.52
$K = 8$	1	0.81	0.73	0.68	0.64	0.61	0.57	0.54	0.50	0.45	0.41
$K = 9$	1	0.80	0.71	0.65	0.61	0.57	0.52	0.46	0.42	0.37	0.33
$K = 10$	1	0.80	0.68	0.63	0.58	0.54	0.46	0.42	0.38	0.34	0.30

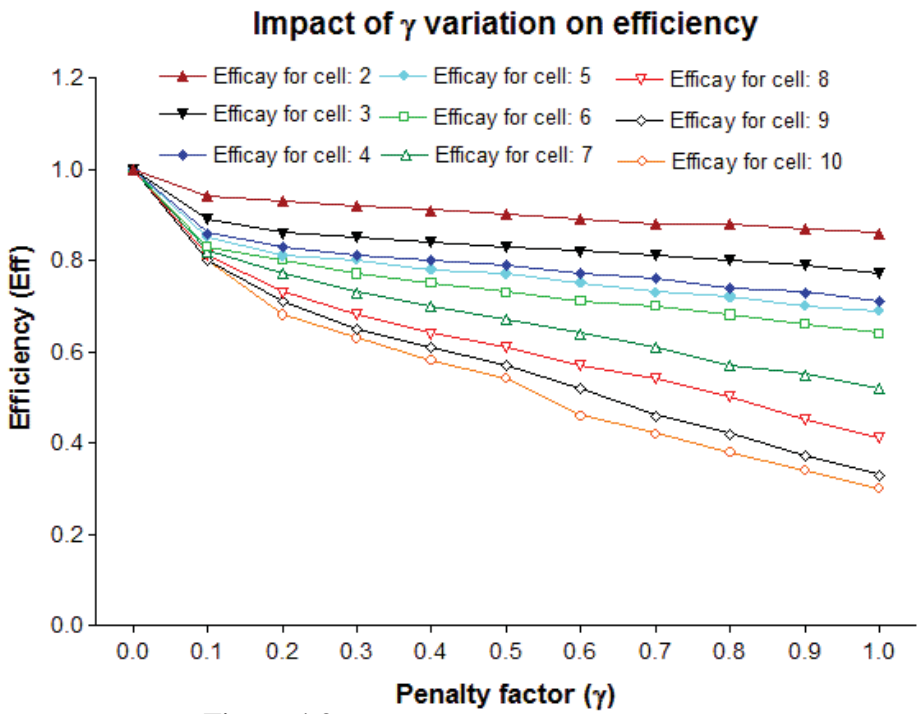

Figure 4.9 – Variation of load balancing factor γ

CHAPITRE 5

ASSIGNMENT OF COMPONENTS AND LINK CAPACITIES TO THREE DIMENSION ON-CHIP MULTIPROCESSOR SYSTEMS

E. Elie, J. A. Ferland, A. Hafid and M. Turcotte

abstract

On-Chip Multiprocessor (OCM) component assignment is a critical and challenging issue for system design. This assignment aims to place components (processors and memories) to specific locations on the chip such that some cost metrics are optimized. This constrained optimization problem has been extensively studied for decades. Methods based on graph theory, integer linear programming, heuristic and stochastic approaches are proposed in the literature. However, most of these methods focus on fine-grain (net list) assignment and do not consider the link capacity aspect. With the trends in three dimension integrated circuit (3D-IC) technology and high functional density of chips, new methods are needed ; these methods must take into consideration the assignment problem at the beginning of the design process (application specification level). In this paper, we propose a coarse-grain assignment method at the application specification level ; this method consists of assigning (1) processors and memories to the nodes of OCM and (2) capacities to the links between each pair of neighboring nodes. To realize the proposed method, we transform the assignment problem of components into a problem belonging to the family of the known quadratic assignment problem (QAP) and we use a perturbation-based approach for link capacity assignment. The objective is to provide a decision support method for engineers for an efficient On-Chip Multiprocessor design. A simulation program using C++ in Linux environment is developed to evaluate our proposed method.

keywords—Combinatorial optimization problem (COP), Network-on-Chip (NoC), On-Chip Multiprocessor, Quadratic assignment Problem (QAP), Tabu search (TS).

5.1 Introduction

Layout synthesis or assignment of OCM components (i.e., processors and memories) in parallel computer systems aim to place these components in specific locations on a chip such that some cost metrics (i.e., performance, chip size, wire length, latency, power consumption, and heat dissipation) are optimized. This constrained optimization problem has been extensively studied for many decades [110], [141] and has been proved to be NP-Hard [145]. As a result of new technology trends, the reduction of the die size and the fact that considerable number of processors and memories can be placed on the same chip, many academic and industrial contributions proposed new methods to solve the assignment problem (also known as placement and routing problem).

The assignment problem is formally defined as follows [40], [67], [105]. Given a cost matrix $C = (c_{ij})$ and a permutation matrix $P = (p_{ij})$, the objective is to resolve the following optimization problem :

$$\begin{aligned} &\text{Minimize} \quad \sum_{i,j=1}^{n} c_{ij} p_{ij} \\ &\text{subject to} \quad p_{ij} = p_{ij}^2,\ \sum_{i=1}^{n} p_{ij} = 1,\ \sum_{j=1}^{n} p_{ij} = 1,\ \forall\ i\ j. \end{aligned}$$

where (c_{ij}) denotes the fixed cost of assigning facility i to location j, and (p_{ij}) denotes the assignment matrix.

In this paper, we propose a two-phase optimization method to assign (1) processors and memories to the nodes of OCM and (2) capacities to the links between each pair of neighboring nodes. The objective is to provide a decision support method for engineers for efficient assignment of components in OCM design.

The proposed method can be used in the following two cases : (1) In the case of an existing OCM system, this approach can help to dynamically allocate traffic to the links once the assignment of the components is completed. Likewise, based on the solution of the link capacity assignment, the switches route data though the network and (2) in the case of the design of a new OCM system, this approach helps in the application specification phase, especially to choose the appropriate OCM organization (i.e., the way

processors and memories are placed in the OCM) and the appropriate link capacities for effective design of OCM. An organization of OCM is called herein a configuration.

5.1.1 Motivation

With the trends in 3D-IC technology, analyses of 3D On-Chip Multiprocessor (3D-OCM) computers have abounded during the last couple of years. The assignment of components has been one of the hottest topics in 3D-OCM design. However, many of these analyses focus their attention on fine-grain (net list) assignment. Also, most of these assignment methods do not take the 3D-IC aspect into consideration. Indeed, most of assignment tools have serious deficiencies in predicting the behavior of 3D-OCM at the application specification level. Our motivation is to provide a method for early assignment evaluation (architectural phase) before the design of OCM, hence, predicting the performance of OCM.

5.1.2 Approach

Our method consist of (1) transforming application specification of OCM into an incidence matrix ; (2) transforming the incidence matrix into an interaction matrix (IA) that defines interactions between sets of processes (process groups) and sets of data (data families) ; (3) making use of IA to assign the process groups and data families to processors and to memories of OCM respectively ; and (4) assigning capacities to links of OCM topology produced in (3). (1) and (2) are out of scope of this paper. In (3), the IA is converted into a flow matrix. Unlike the IA matrix that indicates the interactions between processors (on the rows of the matrix) and the memories (on the columns of the matrix), the flow matrix indicates the interactions between all the components of the OCM ; it indicates interactions between processors and processors, memories and memories, and processors and memories. With this conversion, we transform the assignment problem of components into a problem belonging to the family of the known Quadratic Assignment Problem (QAP) ; this allows us to adapt our assignment problem to existing QAP resolution methods [161]. In (4), starting from the OCM topology produced in (3), we perturb

(i.e., change) the capacities of the links using a perturbation approach derived from perturbation theory [1] ; the objective is to find the best link capacity to connect each pair of neighboring nodes. This perturbation is performed through a loop where at each iteration ; a set of capacity values is assigned to a selected subset of links. At each iteration of the loop, the corresponding near-optimal solution (QAP resolution, with the new capacities, that optimizes the objective metric) is produced. After a maximum number of iterations without any improvement, the algorithm stops. At the end of the execution, the solution that provides the best compromise between the performance and the objective metric (e.g., power consumption) is chosen as the best solution.

On-Chip Multiprocessor (OCM) component assignment is a critical and challenging issue for system design. This assignment aims to place components (processors and memories) to specific locations on the chip such that some cost metrics are optimized. This constrained optimization problem has been extensively studied for decades. Methods based on graph theory, integer linear programming, heuristic and stochastic approaches are proposed in the literature. However, most of these methods focus on fine-grain (net list) assignment and do not consider the link capacity aspect. With the trends in three dimension integrated circuit (3D-IC) technology and high functional density of chips, new methods are needed ; these methods must take into consideration the assignment problem at the beginning of the design process (application specification level). In this paper, we propose a coarse-grain assignment method at the application specification level ; this method consists of assigning (1) processors and memories to the nodes of OCM and (2) capacities to the links between each pair of neighboring nodes. To realize the proposed method, we transform the assignment problem of components into a problem belonging to the family of the known quadratic assignment problem (QAP) and we use a perturbation-based approach for link capacity assignment. The objective is to provide a decision support method for engineers for an efficient On-Chip Multiprocessor design. A simulation program using C++ in Linux environment is developed to evaluate our proposed method.

5.1.3 Contribution

Our contributions, in this paper, can be summarized as follows :

1. We propose a novel multi-phase modeling solution that transforms the component assignment problem into the well known quadratic assignment problem (QAP). More specifically, the proposed modeling solution transforms the interaction matrix (IA), which represents the input to the assignment problem, into a flow matrix which can be used as the input to QAP. Thus, using existing QAP resolution methods, we are able to assign components (processors and memories) to On-Chip Multiprocessor (OCM) system.

2. We propose a new link capacity assignment method that assigns best capacities to the edges of OCM configuration ; the proposed method is based on a perturbation approach derived from perturbation theory [61]. The objective is to produce a configuration of OCM that optimizes a given metric (e.g., power consumption).

3. We implement in C++ the proposed models to demonstrate the effectiveness of our solution for the assignment of the components to the nodes, and for the assignment of the link capacities to the edges of 3D-OCM.

5.1.4 Paper Organization

This paper is organized as follows. Section 5.2 presents related work. Section 5.3 presents a brief description of On-Chip Multiprocessor systems to highlight the structure of processors and memories. Section 5.4 introduces the decomposition of an application specification into components. Section 5.5 describes 3D-OMC model we use. Section 5.6 presents the proposed multi-phase modeling solution for the assignment of components. Section 5.7 presents the proposed link capacity assignment method. Section 5.8 presents empirical results for the assignment of components and link capacities. Section 5.9 concludes the paper and presents future work.

5.2 Related Work

The assignment problem in integrated circuit design has been extensively studied in different ways in the literature. Four groups of methods have been identified : (1) Graph theoretical methods [24], [33], [117], [38], [57] and [97] ; (2) Integer linear programming methods [83] and [169] ; (3) heuristic methods [159], [4], [36], [74] ; and (4) stochastic methods [99], [113], [159]. In the case of graph theory, the assignment problem is solved by finding the minimum cost of the inter-node communication ; it then applies a decomposition algorithm on the task graph to produce task sets where each set is run by a single processor. Graph theory-based methods are limited by the high computation time when the number of nodes increases in the allocation process. The optimal solution of the linear programming methods is hard to obtain because it is an exhaustive enumeration of all possible solutions ; it is also CPU and memory hungry and its requirement for computational resources increases exponentially. Indeed, it is adequate for only small size problem instances where tools, such as CPLEX [88], are generally used. The heuristic methods are used to produce near optimal solutions with an acceptable response time even for large size problem instances. The stochastic methods are used as statistical models to predict component assignment in complex integrated circuits and to obtain cost estimation of routing channels in terms of the number of components, the average number of interconnections between components, and the average interconnection wire length.

In the case of On-Chip Multiprocessor design, the assignment problem is studied as the placement of structural component blocks onto On-Chip Multiprocessor nodes. Extensive research has been performed in this domain and presented in the literature. Lo *et al.* [115] present OREGAMI, an application placement tool, aiming to assign parallel computations to message-passing architectures ; the placement algorithm is based on temporal exploration of the task graph represented by the inter-node communication graph. With the OREGAMI tool, the solution is not fully automated ; the user needs to guide the tool and checks the effectiveness of the mapping decision. The approach used in OREGAMI could lead easily to errors if the graph size is large. Hu et al. [84] propose

a branch and bound algorithm to solve energy-aware optimization models ; the algorithm is used to automatically place the components onto the nodes of a regular configuration network. The objective is to minimize the power consumption by minimizing the inter-core communication. Lei et al. [109] present a tool with two-step genetic algorithm used to place a parallelized application task graph on OCM with the objective of minimizing the overall execution time. Murali et al. [124] present an algorithm called NMAP, which maps a task graph onto OCM with the objective to minimize the communication delay under traffic bandwidth constraints. Existing methods [84], [109], [115], [124] have been mostly used for VLSI design. They concentrate on two dimensional integrated circuits and on fine-grained assignment at RTL. In the era of gigascale On-Chip Multiprocessor, new placement approaches are needed. In this paper, we propose a new method of coarse-grain assignment of processors (IP, RISC, MIPS, DSP) and memories (Flash Memory, ROM, RAM) on 3D integrated circuit. We believe that a coarse-grain near optimal assignment planned at application specification level can yield good OCM performance.

5.3 On-Chip Multiprocessor Structure

Our target architecture in this paper is Three Dimension On-Chip Multiprocessor (3D-OCM) structure (e.g., Figure 5.2.a) with multiple different components. 3D-OCM is accompanied by network-on-chip (NoC). We consider network-on-chip (NoC) as the interconnect communication platform methodology of OCM ; it consists of NoC Interface Units (NIU) and the switches [102], [21], [79]. The NIU can be processors, Digital Signal Processors, Digital/Analog Signal cores, Application Specific Integrated Circuits (ASICs), IPs or memories (ROM, RAM, Flash memory). A space of OCM that hosts a NIU is called a tile. The switches connect the NIUs via communication links or channels. Let us recall that the most used representation of OCM is mesh configuration. In the following, we use, as the assignment target structure, 3D-OCM illustrated by 3D mesh configuration shown in Figure 5.1 ; each node of the structure represents a network tile with its switch and each edge represents a link. Without loss of generality, a 3D mesh NoC is defined as a superposition of several 2D NoC (see Figure 5.2.b).

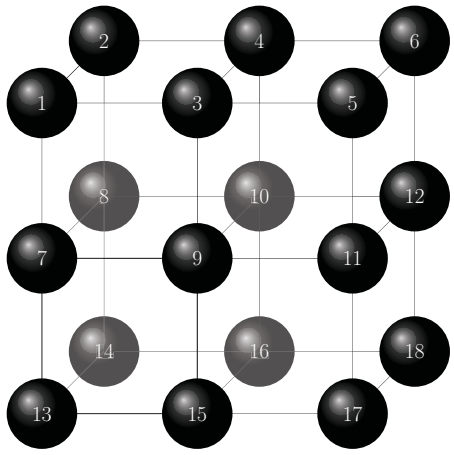

Figure 5.1 – Target Architecture Structure

5.4 Decomposition of Application Specification for On-Chip Multiprocessor

Usually, an application that runs on On-Chip Multiprocessor (OCM) systems is very complex and parallelizable ; otherwise, a simple mono-processor system is sufficient. Such an application needs to be decomposed (broken down) into a number of components (processes and data sets) to take full benefit of OCM ; thus, the different components of an application can be executed by the different components (processors and memories) of OCM. Indeed, the assignment problem, we consider in this paper, is to assign the processes and data sets to processors and to memories of OCM respectively. In other words, the result of the application decomposition procedure is the Interaction Matrix (IA) that represents the input to the proposed component assignment approach. In this section, we briefly present the details of a decomposition method [133] that can be used to produce the IA for a given application. The application decomposition problem is solved as a cell formation problem [133].

Let us assume that m processors pc_α, $\alpha = 1,2,\ldots,m$ and n memories du_β, $\beta = 1,2,\ldots,n$ have to be assigned to the tiles on a chip. Furthermore, let use assume that $a_{\alpha\beta}$ denote the number of accesses of pc_α to du_β. In [133], we solve a cell formation problem to determine K cells (C_k, Fam_k), $k = 1,2,\ldots,K$ where C_k and Fam_k are groups of processors and family of memories, respectively. The groups C_k, $k = 1,2,\ldots,K$ and

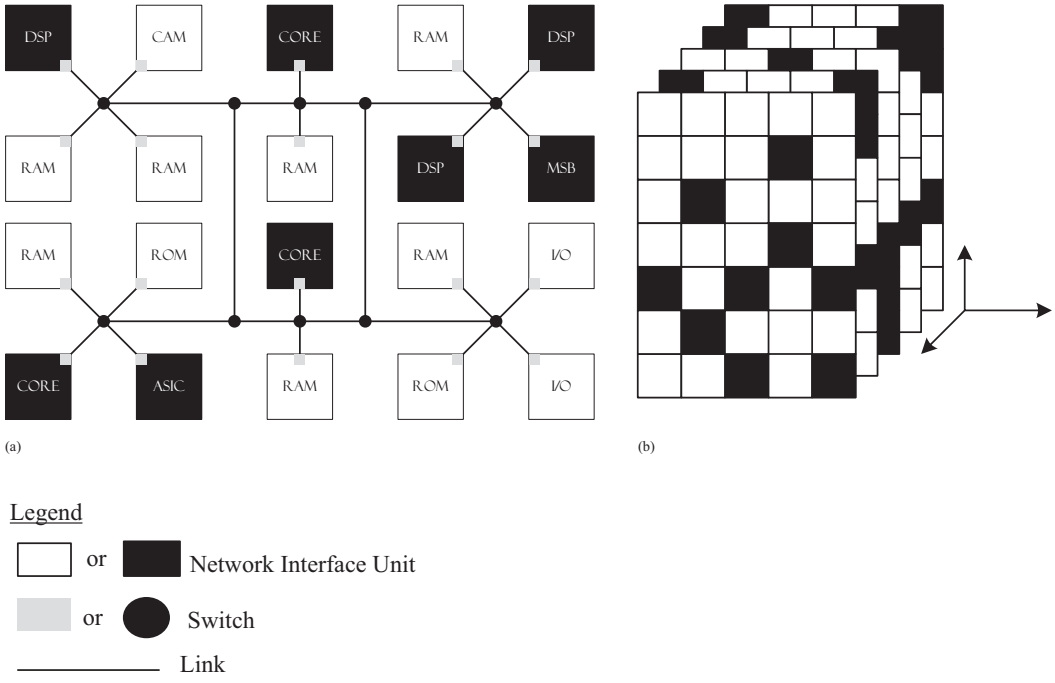

Figure 5.2 – (a) 2D and (b) 3D Switch Fabric Network Architecture

the families Fam_k, $k = 1,2,\ldots,K$, are partitions of the set of processors and of the set memories, respectively. The objective of the cell formation problem is to reduce the total number of accesses of the processors to memories belonging to other cells. Furthermore, we aim to have the same number of accesses by the processors to memories within each cell, where a cell is defined as a set of processors and memories that have large number of accesses among themselves. A square $K \times K$ interaction matrix $\mathrm{IA} = [\mathrm{IA}_{ij}]$ is associated with K cells (C_k, Fam_k), $k = 1,2,\ldots,K$ where

$$\mathrm{IA}_{ij} = \sum_{pc_\alpha \in C_i} \sum_{du_\beta \in Fam_j} a_{\alpha\beta}$$

Note that solving the cell formation problem infers in general that $\forall i = 1,\ldots,K$, $\mathrm{IA}_{ii} \gg \mathrm{IA}_{ij}$, $\forall j = 1,\ldots,K$, $j \neq i$ because the objective is to reduce the number of accesses of processors in C_i to data sets in Fam_j, $j \neq i$. Furthermore, we partition the processors and memories in order to have the value of IA_{ii} as similar as possible for all i.

5.5 The model of 3D On-Chip Multiprocessor

In this paper, the computer architecture is a $3D$ On-Chip Multiprocessor represented by a graph $\mathscr{N} = (V,E)$. Each vertex, also called a tile, corresponds to a node where a group of processes or a family of data is assigned. Hence, assuming that K groups and K families have to be assigned, then the number of vertices in V is equal to $2K$. The structure of the configuration of the computer architecture is illustrated in Figure 5.1. Typically, there is no diagonal edge. For instance, there is no edge between vertices 1 and 2. A routing path between two nodes v_1 and v_2 is identified by a list of links. Furthermore, there exist several routing paths between any pair of nodes. Link may support different capacities of transaction depending on the architecture configuration. Different capacities cause different connection times of access between the two connected nodes. It follows that the routing path has a connection time equals to the sum of the connection times of links that compose the path. Let $\bar{D} = [\bar{d}_{pq}]$ be the connection time matrix where $\bar{d}_{pq}$ is the shortest connection time to connect the pair of nodes p and q. Note that we use the Floyd-Warshall algorithm [27] to determine the shortest path between each pair of nodes.

5.6 Assignment of Groups and Families to the 3D On-Chip Multiprocessor Nodes

The rationale behind the proposed assignment of components on OCM is to place processors and memories to specific locations on the chip such that the total cost of interactions between processors and memories is minimized (optimized). This assignment problem is a complex problem ; its difficulty lies in the fact that it tries to place each component to a position on OCM, with the objective to minimize any interaction cost between two pair of components. The assignment problem is NP-Hard ; therefore, it is impossible to solve in polynomial time, even for small instances [100]. The Quadratic Assignment Problem (QAP), a class of assignment problems, is an alternative approach that yields near-optimal solutions. Thus, the proposed assignment of processors (groups) and memories (families) to 3D OCM can be determined by solving a Quadratic Assignment Problem.

5.6.1 Quadratic Assignment Problem

A quadratic assignment problem (QAP) is a NP-Hard Integer Programming problem. It was introduced in 1957 by Koopmans and Beckmann [100]. QAP aims to minimize the cost of assigning r facilities to specific locations, considering the flow of goods (information or materials) between the facilities and the distance (costs) between the locations. Hence, two $r \times r$ matrices are required : F, the matrix of flows between the locations and D the matrix of distances between locations.

The Quadratic Assignment Problem can be formulated as follows :

$$\underset{\pi \in \Pi(r)}{\text{Min}} \; z(\pi) = \sum_{i=1}^{r} \sum_{j=1}^{r} f_{ij} d_{\pi(i)\pi(j)} \tag{5.1}$$

where π is a permutation vector of $\{1,2,\ldots r\}$, Π the set of all $r-vector$ permutations, and $\pi(i)$ is the location where facility i is assigned. The objective is to determine the assignment minimizing the total cost of moving the goods. Even though the primary application of QAP is for facilities location, the problem has been used extensively and applied in different research areas like industry, technology and fundamental science domains [31]. As in various NP-Hard programming problems, exact methods are proposed for relatively small problems while metaheuristic approaches are proposed for larger ones. Among the metaheuristic approaches proposed for QAP we mention the most notable ones like Hopefield neural networks [26], simulated annealing [42], threshold accepting [174], genetic algorithm [163], [64], [41], Tabu search [161], [151], the evolution strategies [131] and the ant colony optimization [119], [156], [66].

In this paper, we use Taillard Tabu search. The idea of the Taillard approach is based on an efficient and robust Tabu search. It can solve problems having more than 20 facilities. Its functionality is based on : (1) a random variation of the Tabu list length ; (2) the prohibition of an exchange between two nodes if they had exchanged their locations recently ; and (3) the execution of a move if it is not Tabu. We select Taillard algorithm to solve our QAP because it is efficient, simple to implement and its neighborhood exploration time is in $O(n^2)$ instead of $O(n^3)$ as it is the case for most other algorithms [161].

5.6.2 QAP Formulation of Assigning Groups and Families to 3D On-Chip Multiprocessor

To formulate our assignment problem as a QAP, the 3D network nodes correspond to the locations, also called tiles. The connection time matrix $\bar{D}$ specified in section 5.5 is used as the distance matrix in QAP. Recall that $\bar{D} = [\bar{d}_{pq}]$ where $\bar{d}_{pq}$ is the shortest connection time among all routing paths linking nodes p and q. Furthermore, since the edges in the 3D network can be used in both ways (*i.e.*, the edges are not oriented), it follows that the matrix $\bar{D}$ is symmetric (*i.e.*, $\bar{d}_{pq} = \bar{d}_{qp}$). To specify the flow matrix F (square matrix of order $2K$), we refer to the solution of the cell formation problem, as stated earlier, where K processors $\{C_1 \dots C_K\}$ and K memories $\{Fam_1 \dots Fam_K\}$ are generated together with the square interaction matrix IA. These processors and memories correspond to the $2K$ facilities, and they are ordered as follows : $\{C_1 \dots C_K, Fam_1 \dots Fam_K\}$.

Different formulations of the matrix F are used in order to generate different configurations according to the computed application under consideration (i.e., to be executed on OCM). Three different cases are analyzed in the following : (1) free configuration assignment, (2) interleaved configuration assignment and (3) Memory intensive communication configuration assignment. However, many other formulations can be proposed. The rationale behind the choice of the three different configurations is :

1. The free configuration assignment (also called performance critical assignment) is proposed for high-performance design when there are no physical or technological design limitations (e.g., mixed-signal design, processors and memories built of the same IC technology). Processors and memories are randomly distributed on OCM. This configuration is used for general purpose processing and non regular memory accesses. The primary goal is to maximize OCM performance (e.g., time-closure).

2. The interleaved configuration assignment is proposed for OCM design when processors and memories are built on the same 3D IC die. This configuration is effective for autonomous, repetitive or sub-data processing like polynomial evaluation or matrix manipulation in image processing where each sub matrix can be computed separately by different processors and where the shared data between the

processors can be decomposed ; for example, this configuration is suitable to execute image registration where the application has to locate reference images within a larger one. In this case, data distribution is very limited (*less distribution and synchronization in the network*) ; these characteristics make the interleave (eliminate the proximity of processors and memories) assignment suitable. In the interleaved configuration assignment, each processor can simultaneously be connected to six different memories and each memory is connected to six different processors. In both cases, four in horizontal connection and two in vertical connection.

3. The memory intensive communication configuration assignment is proposed for OCM with vertical packaging of memory arrays and processors arrays. This configuration is proposed for different technologies or IPs integration and where vertical packaging is critical for space management (i.e., handheld processors design).

5.6.2.1 Free Topology Assignment

In this case, the designer requires assigning the processors and the data to the 3D network nodes in order to reduce the total access time. Since processors are not required to have access to each other, then

$$F^1_{\mu\nu} = 0 \quad \text{if } \mu \text{ and } \nu \in \{1,2,\ldots,K\}$$

Similarly, since the memories are not required to have access to each other, then

$$F^1_{\mu\nu} = 0 \quad \text{if } \mu \text{ and } \nu \in \{K+1,\ldots,2K\}$$

For the other elements of F^1 involving groups and families, we refer to the interaction matrix IA introduced in section 5.4. Hence these elements correspond to the access bet-

ween groups and families :

$$\begin{aligned} \mathrm{IA}_{\mu(\nu-K)} \quad & \text{if } \mu \in \{1,2,\ldots,K\} \text{ and} \\ & \nu \in \{K+1,\ldots,2K\} \\ & (\textit{i.e. } \mu \text{ is a processor group and} \\ & \nu \text{ is a memory family)} \end{aligned}$$

$$\begin{aligned} \mathrm{IA}_{(\mu-K)\nu} \quad & \text{if } \mu \in \{K+1\ldots,2K\} \text{ and} \\ & \nu \in \{1,2,\ldots,K\} \\ & (\textit{i.e. } \mu \text{ is a memory family and} \\ & \nu \text{ is a processor group)} \end{aligned}$$

A matrix F^1 is shown in Figure 5.3, and a typical 3D network associated with F^1 generated with Taillard's Tabu search procedure is illustrated in Figure 5.10.

$F^1 =$

		Processor Groups				Data Unit Families			
		C_1	C_2	...	C_K	Fam_1	Fam_2	...	Fam_K
Processor Groups	C_1	0	0	...	0	IA_{11}	IA_{12}	...	IA_{1K}
	C_2	0	0	...	0	IA_{21}	IA_{22}	...	IA_{2K}
	⋮	⋮				⋮			
	C_K	0	0	...	0	IA_{K1}	IA_{K2}	...	IA_{KK}
Data Units Families	Fam_1	IA_{11}	IA_{12}	...	IA_{1K}	0	0	...	0
	Fam_2	IA_{21}	IA_{22}	...	IA_{2K}	0	0	...	0
	⋮	⋮				⋮			
	Fam_K	IA_{K1}	IA_{K2}	...	IA_{KK}	0	0	...	0

Figure 5.3 – Free Topology Assignment Model

5.6.2.2 Eliminate Proximity of Processors and Proximity of Memories

In the second case, we intend to eliminate proximity of processors and proximity of memories ; i.e., two processors or two memories should not be assigned to adjacent network nodes. Then, the entries of F^2 related to the access between two processor

groups or between two memory families are specified as follows :

$$\begin{cases} F^2_{\mu\nu} = -\infty & \text{if } \mu \text{ and } \nu \in \{1,\ldots,K\},\ \mu \neq \nu \text{ or} \\ & \text{if } \mu \text{ and } \nu \in \{K+1,\ldots,2K\},\ \mu \neq \nu \\ F^2_{\mu\mu} = 0 & \text{if } \mu \in \{1,\ldots,K,K+1,\ldots,2K\} \end{cases} \tag{5.2}$$

The other entries of F^2 are specified as in F^1. A matrix F^2 is illustrated in Figure 5.4 and a typical 3D network associated with F^2, generated with Taillard's Tabu search procedure, is illustrated in Figures 5.12 and 5.13.

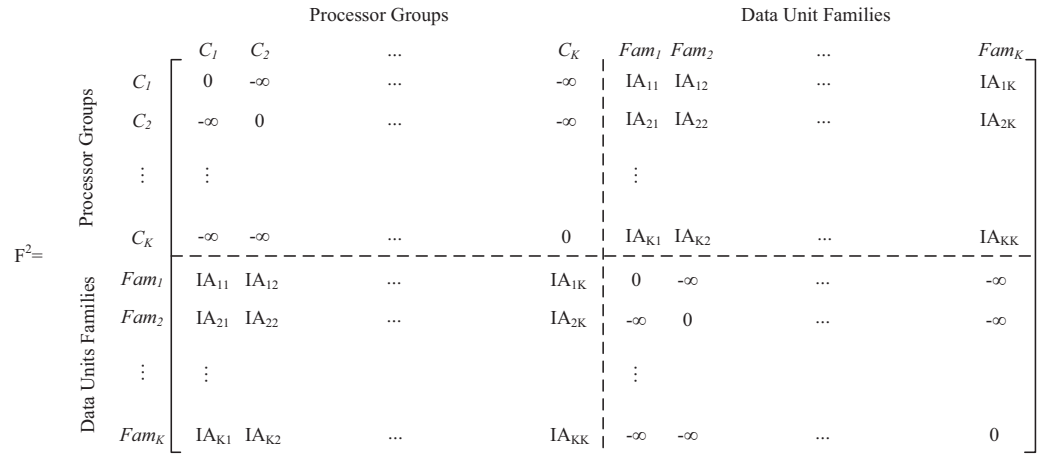

Figure 5.4 – Eliminate Proximity of Groups and Proximity of Families Model

5.6.2.3 Memory Intensive Communication

Referring to Figure 5.5, we observe that for each cell k $(k = 1,\ldots,K)$, the group C_k and the family Fam_k are assigned to adjacent nodes of the network. This is a consequence of the objective of the cell formation problem to induce a large number of accesses between C_k and Fam_k $(k = 1,\ldots,K)$, as compared to the number of access between C_{k^1} and Fam_{k^2} when $k^1 \neq k^2$. It follows that the edges linking nodes C_k and Fam_k should have a large capacity of transactions. As a consequence, the routing path linking C_k to any other $Fam_{k'}$, $k \neq k'$, should include the link between C_k and Fam_k. Consequently, we can generate a flow matrix F^3 accounting for this observation by simulating the accesses

from processors in C_k to any memory in $Fam_{k'}$, $k \neq k'$ as if it is accessed from Fam_k to memory in $Fam_{k'}$. Thus the flow matrix is specified as illustrated by Figure 5.5 and a typical 3D matrix network associated with F^3, generated with Taillard's Tabu search procedure is illustrated in Figure 5.12.

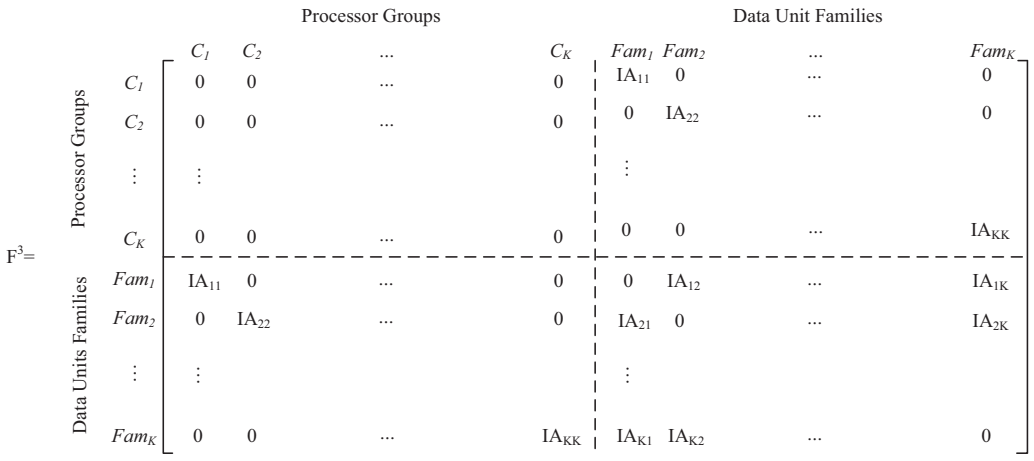

Figure 5.5 – Memory Intensive Communication Model

5.6.3 Assigning the Processors and Memories to the Nodes of 3D On-Chip Multiprocessor

Let us consider the connection time matrix $\bar{D}$ introduced in Section 5.6.2 as the distance matrix to be used in the definition of the quadratic assignment problem. We then have three different problems associated with each of the three flow matrices F^1, F^2 and F^3 introduced before. Solving these quadratic assignment problems with Taillard's Tabu search procedure, four different types of multi-core computer architectures are illustrated in Figures 5.10 - 5.13 (Figures 5.12 and 5.13 represent different executions of the flow F^3), when $K = 9$ (i.e., 9 processors and 9 memories). In all four computer architectures, we note that for each cell k, $(k = 1, \ldots, K)$ the group C_k and the family Fam_k are located at neighboring nodes (i.e., nodes linked with an edge) of the network. This follows directly from solving the cell formation problem generating the groups and the families to reduce the number of accesses from processors in the cell k to memories in

any other cell k' with $k \neq k'$. Furthermore the distance increases between nodes when interaction between processor in cell k and memory in cell k' decrease i.e., the number of accesses from processor in cell k to memory in cell k' ($IA_{kk'}$) is smaller. The procedure is summarized in Algorithm 5.

Algorithm 5 *QAP Formulation of Assigning Groups and Families to the 3D Networks Nodes*

Require: $IA = [IA_{ij}]$ and $\mathscr{N}$.

1: Generate $\bar{D}$ from $\mathscr{N}$. {A}n entry of $\bar{D}$ is 1 if there is an edge e between two nodes p and q and 0 otherwise.
2: Choose a target topology.
3: Generate flow matrix as follows :
4: **if** Free Topology Assignment **then**

$$\begin{cases} F^1_{\mu\nu} = 0 & \text{if } \mu \text{ and } \nu \in \{1,2,\ldots,K\} \\ F^1_{\mu\nu} = 0 & \text{if } \mu \text{ and } \nu \in \{K+1,\ldots,2K\} \\ F^1_{\mu(\nu-K)} = IA_{\mu(\nu-K)} & \text{if } \mu \in \{1,2,\ldots,K\} \text{ and } \nu \in \{K+1,\ldots,2K\} \\ F^1_{(\mu-K)\nu} = IA_{(\mu-K)\nu} & \text{if } \mu \in \{K+1\ldots,2K\} \text{ and } \nu \in \{1,2,\ldots,K\} \end{cases}$$

5: **end if**
6: **if** Eliminate Proximity of Groups and Proximity of Families **then**

$$\begin{cases} F^2_{\mu\nu} = -\infty & \text{if } \mu \text{ and } \nu \in \{1,\ldots,K\},\ \mu \neq \nu \text{ or if } \mu \text{ and } \nu \in \{K+1,\ldots,2K\},\ \mu \neq \nu \\ F^2_{\mu\mu} = 0 & \text{if } \mu \in \{1,\ldots,K,K+1,\ldots,2K\} \end{cases}$$

7: **end if**
8: **if** Memory Intensive Communication **then**
9: Refer to Section 5.6.2.3
10: **end if**
11: $\mathscr{N} \leftarrow QAP_{Taillard}(F^*, \bar{D})$ [161]

5.7 Link Capacity Assignment to 3D On-Chip Multiprocessor

The characteristics of the links in OCM design depend strongly on the application specification under consideration. With the dramatic decrease of transistor size and the increasing of the density of IC chips, parallel links cannot guarantee the signal integrity due to cross-coupling noise. In OCM, effective communication link design is critical for

the overall performance. In order to have the best trade-off between performance, power, energy dissipation and area, the design of links need to be planned ahead during the application specification and the architecture phases. The performance of links must be included in the overall performance analysis of the system. Oswens *et al.* [136] show that enormous power is used to move and store data across OCM. Many contributions have been proposed in the literature to improve data transfer in OCM. Most of them focus on the buffer design optimization (e.g., [84] and [129]). More recently, few proposals concentrate on the 3D packaging or IC design not only to reduce the crossbar switch power consumption but also to maximize the interconnect performance of OCM components. Due to the fact that the most part of power consumed by OCM is used to move and store data, we can assume that reducing data transfer between OCM components will increase its performance.

As stated earlier, solving the component assignment problem is NP-hard. The link capacity assignment to OCM in this context is adding another level of complexity that makes our problem even harder to solve in polynomial time. In this section, we propose an effective approach for link capacity assignment to 3D OCM ; we use perturbation based approach to find the best set of link capacities for OCM ; more specifically, the assignment of link capacities between processors and memories in 3D OCM will be determined by perturbing the value of each link capacity between neighbouring nodes. The set of values that gives the best QAP resolution cost is considered the best solution of the link capacity assignment.

This Section is organized as follows. Section 5.7.1 introduces the perturbation problem. Section 5.7.2 presents the formulation of the link capacity assignment. Section 5.7.3 describes how the capacities are adjusted between neighboring nodes for near-optimal solution. Section 5.7.4 presents the link capacity assignment procedure.

5.7.1 Perturbation Problem

The perturbation approach we use is derived from the perturbation theory. The perturbation theory is an approximation resolution method used in applied mathematics to find solutions to complex problems which cannot be solved otherwise [61]. The basic

idea of the resolution is to find an exact solution to a related problem and change this solution by adding or subtracting a small term to the variables of the problem to obtain the desired solution. As mentioned in [61], this method is mainly used in theoretical physics and chemistry.

In our proposed link capacity assignment problem, we use a perturbation approach to find the desired solution. We consider as input for this problem the solution returned by the first QAP resolution (See Section 5.6.2). This solution, returned by QAP, is set to be the known solution. We then add a small number (this number depends on the objective of OCM designer) to the distance matrix at each iteration of the resolution. The best solution is a compromise between the best cost of QAP resolution and the best approximation to the computed metric value.

5.7.2 Formulation of Assigning Link Capacities to the 3D Network Edges

To formulate the link assignment as a perturbation problem, the network $\mathcal{N}$ edges are variables. Let us recall that $\bar{D} = [\bar{d}_{pq}]$ is symmetric and that $\mathcal{N}$ is not oriented. Furthermore, let us assume that all edges have the same capacity for the solution returned by the first resolution of QAP (See Section 5.6.2). For a given metric constraint (die surface, wire length, heat dissipation, power consumption ...), we perturb the cost matrix $\bar{D}$ by adding a small number $\alpha_{\bar{d}}$ to $\bar{d}_i$ to reduce the capacity of the link. Because $\bar{d}_i$ represents the cost to transfer data from one node to another, a small $\bar{d}_i$ means a big link capacity and a big $\bar{d}_i$ means a small capacity. A new QAP is resolved any time link capacities change. The best QAP result that approximates both the first QAP result and the desired metric is considered to be the best solution. The link capacity assignment solution consists the new set of link capacities $\{\bar{d}_1, \ldots, \bar{d}_{|E|}\}$ of OCM that corresponds to the best solution.

5.7.3 Links of the 3D Network : Capacity Adjustment

Recall that to formulate the Quadratic Assignment Problem, the capacity of each link of 3D OCM has been specified in order to generate the connection time matrix $\bar{D} = [\bar{d}_{pq}]$.

Once the processors and the memories have been assigned to the 3D network nodes, we can adjust the capacity of each link allowing the proper level of accesses between the network nodes in order to reduce the total capacity cost of the network.

When using the Floyd-Warshall algorithm to determine the matrix $\bar{D}$, the procedure also generates the additional information to identify the links of the shortest path for each pair of nodes p and q. Consequently, we can determine the level of access $L(e)$ going through each link e of the network as follows. Denote

$$OD(e) = \{(k,k') : \begin{array}{l}\text{link is on the shortest path linking} \\ \text{the pair of nodes where group } k \\ \text{and family } k' \text{ are assigned} \\ \text{respectively}\}\end{array}$$

$$\text{Then} \quad L(e) = \sum_{(k,k') \in OD(e)} IA_{kk'}$$

Furthermore, denote by $cap(e)$ the minimal bus capacity such that $L(e) \leq cap(e)$, and let $\$(e)$ be the cost of a $cap(e)$ bus capacity. It follows that the total capacity cost CC is

$$CC = \sum_{e \in E} \$(e)$$

5.7.4 Link Capacity Perturbation Procedure

In this section, we propose an iterative procedure that can be used by the designer to analyze the total capacity cost for fixing the capacity of the links versus the access flow cost generated by solving the quadratic assignment problem. The procedure is summarized in Algorithm 6. Note that a way to complete the initialization is assigning the largest capacity value to each edge. An execution of the procedure is illustrated in Figure 5.16.

5.8 Experiments

In this section, we use the algorithms we proposed in Section 5.6 and Section 5.7 to solve the assignment of the components of an application, represented by interaction

Algorithm 6 *Perturbation Procedure*

1: Determine an initial capacity for each link $e \in V$.
2: Determine the corresponding connection time associated with the capacity for each link $v \in V$.
Determine the matrix $\bar{D}$ with the Floyd-Warshall algorithm.
Select one of the flow matrix F^1, F^2, or F^3.
3: Solve QAP using Taillard's Tabu search, to obtain access flow cost $z(\pi)$.
4: Adjust the link capacities as in section 5.7.3 in order to determine the total capacity cost CC.
5: Analyze and compare $z(\pi)$ versus CC.
6: **if** analysis is conclusive **then**
7: The procedure terminate
8: **else**
9: Modify the capacity of some link $e \in V$
10: Goto 1.
11: **end if**

matrix IA (Figure 5.6), to OCM. The objective is to illustrate the effectiveness of the proposed component and link capacity assignment models. This section is organized in two parts. The first one focuses on the assignment of components for different configurations and the second one focuses on the assignment of link capacities. In the numerical application, the axis z represents the cost metrics we aim to optimize (i.e., chip size, wire length, latency, power consumption, and heat dissipation).

5.8.1 Simulation Setup

We consider an example where the number of processors and the number of the memories are both equal to 9 respectively. The 9×9 square interaction matrix IA is shown in Figure 5.6. Note that the elements on the diagonal of IA dominate the other elements of the matrix since they correspond to the level of accesses within the cells generated by solving the cell formation problem [132]. The 9 groups and the 9 families of IA must be assigned to 3D mesh network $(3 \times 3 \times 2)$, $\mathcal{N} = (V, E)$ having $|V| = 18$ and $|E| = 33$. The procedures have been implemented in C++ and the numerical experiments have been run using a Linux-based machine (AMD Sempron (t) Processor 3100, clock 1GHz, 256KB of cache, and 1GB of RAM).

IA =

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
F_0	713	10	170	0	0	7	233	10	147
F_1	0	627	8	251	5	35	6	0	0
F_2	18	13	440	6	98	18	16	22	7
F_3	22	409	7	875	0	93	28	111	15
F_4	0	0	275	9	1025	11	10	20	98
F_5	0	0	0	2	0	440	0	0	0
F_6	121	18	0	9	87	5	867	0	78
F_7	153	6	1	267	0	91	0	1246	0
F_8	376	8	0	0	50	52	84	26	1064

Figure 5.6 – IA configuration example

F^1 =

	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
F_0	0	0	0	0	0	0	0	0	0	713	10	170	0	0	7	233	10	147
F_1	0	0	0	0	0	0	0	0	0	0	627	8	251	5	35	6	0	0
F_2	0	0	0	0	0	0	0	0	0	18	13	440	6	98	18	16	22	7
F_3	0	0	0	0	0	0	0	0	0	22	409	7	875	0	93	28	111	15
F_4	0	0	0	0	0	0	0	0	0	0	0	275	9	1025	11	10	20	98
F_5	0	0	0	0	0	0	0	0	0	0	0	0	2	0	440	0	0	0
F_6	0	0	0	0	0	0	0	0	0	121	18	0	9	87	5	867	0	78
F_7	0	0	0	0	0	0	0	0	0	153	6	1	267	0	91	0	1246	0
F_8	0	0	0	0	0	0	0	0	0	376	8	0	0	50	52	84	26	1064
M_0	713	0	18	22	0	0	121	153	376	0	0	0	0	0	0	0	0	0
M_1	10	627	13	409	0	0	18	6	8	0	0	0	0	0	0	0	0	0
M_2	170	8	440	7	275	0	0	1	0	0	0	0	0	0	0	0	0	0
M_3	0	251	6	875	9	2	9	267	0	0	0	0	0	0	0	0	0	0
M_4	0	5	98	0	1025	0	87	0	50	0	0	0	0	0	0	0	0	0
M_5	7	35	18	93	11	440	5	91	52	0	0	0	0	0	0	0	0	0
M_6	233	6	16	28	10	0	867	0	84	0	0	0	0	0	0	0	0	0
M_7	10	0	22	111	20	0	0	1246	26	0	0	0	0	0	0	0	0	0
M_8	147	0	7	15	98	0	78	0	1064	0	0	0	0	0	0	0	0	0

Figure 5.7 – F^1 configuration example

5.8.2 Assignment of Components : Configurations associated with the different flow matrices F^1, F^2, and F^3

The following results illustrate the different configurations obtained with the three different flow matrices F^1, F^2, and F^3 generated using the interaction matrix IA. These matrices are shown in Figures 5.6-5.9 respectively. In this case, we assume that all the links of V have the same maximal capacity and thus the cost to move data from one node to another in OCM equal to 1. The matrix shown in Figure 5.7 shows the free configuration flow matrix F^1 associated with interaction matrix IA. The assignment is performed by solving QAP without any particular instruction. Figure 5.10 illustrates the

F^2 =

	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
F_0	0	-2500	-2500	-2500	-2500	-2500	-2500	-2500	-2500	713	10	170	0	0	7	233	10	147
F_1	-2500	0	-2500	-2500	-2500	-2500	-2500	-2500	-2500	0	627	8	251	5	35	6	0	0
F_2	-2500	-2500	0	-2500	-2500	-2500	-2500	-2500	-2500	18	13	440	6	98	18	16	22	7
F_3	-2500	-2500	-2500	0	-2500	-2500	-2500	-2500	-2500	22	409	7	875	0	93	28	111	15
F_4	-2500	-2500	-2500	-2500	0	-2500	-2500	-2500	-2500	0	0	275	9	1025	11	10	20	98
F_5	-2500	-2500	-2500	-2500	-2500	0	-2500	-2500	-2500	0	0	0	2	0	440	0	0	0
F_6	-2500	-2500	-2500	-2500	-2500	-2500	0	-2500	-2500	121	18	0	9	87	5	867	0	78
F_7	-2500	-2500	-2500	-2500	-2500	-2500	-2500	0	-2500	153	6	1	267	0	91	0	1246	0
F_8	-2500	-2500	-2500	-2500	-2500	-2500	-2500	-2500	0	376	8	0	0	50	52	84	26	1064
M_0	713	0	18	22	0	0	121	153	376	0	-2500	-2500	-2500	-2500	-2500	-2500	-2500	-2500
M_1	10	627	13	409	0	0	18	6	8	-2500	0	-2500	-2500	-2500	-2500	-2500	-2500	-2500
M_2	170	8	440	7	275	0	0	1	0	-2500	-2500	0	-2500	-2500	-2500	-2500	-2500	-2500
M_3	0	251	6	875	9	2	9	267	0	-2500	-2500	-2500	0	-2500	-2500	-2500	-2500	-2500
M_4	0	5	98	0	1025	0	87	0	50	-2500	-2500	-2500	-2500	0	-2500	-2500	-2500	-2500
M_5	7	35	18	93	11	440	5	91	52	-2500	-2500	-2500	-2500	-2500	0	-2500	-2500	-2500
M_6	233	6	16	28	10	0	867	0	84	-2500	-2500	-2500	-2500	-2500	-2500	0	-2500	-2500
M_7	10	0	22	111	20	0	0	1246	26	-2500	-2500	-2500	-2500	-2500	-2500	-2500	0	-2500
M_8	147	0	7	15	98	0	78	0	1064	-2500	-2500	-2500	-2500	-2500	-2500	-2500	-2500	0

Figure 5.8 – F^2 configuration example

assignment of processors in black and of memories in white.

To eliminate the proximity of processors and memories, we generated the flow matrix F^2 shown in Figure 5.8. The generated topology is illustrated in Figure 5.11 ; the processors are isolated from each other and the memories are also isolated from each other. This configuration is proposed for autonomous, repetitive or sub-data processing like polynomial evaluation or matrix manipulation in image processing where each sub matrix can be computed separately by different processors and where the shared data between the processors can be decomposed ; for example, this configuration is suitable to execute image registration where the application has to locate reference images within a larger one. In this case, data distribution is very limited (*less distribution and synchronization in the network*) ; these characteristics make the intersperse (eliminate the proximity of processors and memories) assignment suitable.

Matrix F^3 shown in Figure 5.9 corresponds to the memory intensive communication case. Recall that in this case any connection between a processor p_k and any memory $m_{k'}$ is completed by going though the memory m_k belonging to the same cell k. The configuration is shown in Figures 5.12-5.13. Both Figures are illustrations for the assignment of F^3 on OCM. The fact that F^3 can have two configurations shows the complexity (NP-hard) of our assignment problem ; we observe that all the memories are connected and

$F^3 =$

	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
F_0	0	0	0	0	0	0	0	0	0	713	0	0	0	0	0	0	0	0
F_1	0	0	0	0	0	0	0	0	0	0	627	0	0	0	0	0	0	0
F_2	0	0	0	0	0	0	0	0	0	0	0	440	0	0	0	0	0	0
F_3	0	0	0	0	0	0	0	0	0	0	0	0	875	0	0	0	0	0
F_4	0	0	0	0	0	0	0	0	0	0	0	0	0	1025	0	0	0	0
F_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	440	0	0	0
F_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	867	0	0
F_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1246	0
F_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1064
M_0	713	0	0	0	0	0	0	0	0	0	10	188	22	0	7	354	163	523
M_1	0	627	0	0	0	0	0	0	0	10	0	21	660	5	35	24	6	8
M_2	0	0	440	0	0	0	0	0	0	188	21	0	13	373	18	16	23	7
M_3	0	0	0	875	0	0	0	0	0	22	660	13	0	9	95	37	378	15
M_4	0	0	0	0	1025	0	0	0	0	0	5	373	9	0	11	97	20	148
M_5	0	0	0	0	0	440	0	0	0	7	35	18	95	11	0	5	91	52
M_6	0	0	0	0	0	0	867	0	0	354	24	16	37	97	5	0	0	162
M_7	0	0	0	0	0	0	0	1246	0	163	6	23	378	20	91	0	0	26
M_8	0	0	0	0	0	0	0	0	1064	523	8	7	15	148	52	162	26	0

Figure 5.9 – F^3 configuration example

share the same communication space (Highlighted in Figures 5.12-5.13). The advantage of a such configuration is that it can implement shared, distributed, or distributed-shared memories. In the case of shared memories, the memories connected to processors are seen as different pages of a single memory with different and scalable address spaces, for example, the space $0x00000000$ to $0x0000FFFF$ for M_0, $0x00010000$ to $0x0001FFFF$ for M_1 and $0x00020000$ to $0x0002FFFF$ for M_2. To access a memory location, a simple read or write instruction to the memory by its connected processor will find the data location based on the address table. Indeed, a table containing address space is implemented in each memory interface (MI). A routing protocol is implemented to move data between memories. Routing protocol, synchronization primitive and data coherence of share memories are out of scope of this paper [81]. Unlike a conventional shared memory where processors access the centralized memory via a bus, our shared memory is based on memory network, where all the memories of OCM are connected through a 3D mesh based network. Hence, a processor can access directly its connected memory. For each request made by a processor, if the data is not in its connected memory, the MI (Memory Interface) of this memory searches for the data in other memories of the network ; thus, the behavior of the memory network is transparent to the processor. In the case of distributed memory, each processor p_k considers its connected memory m_k as the extension of its

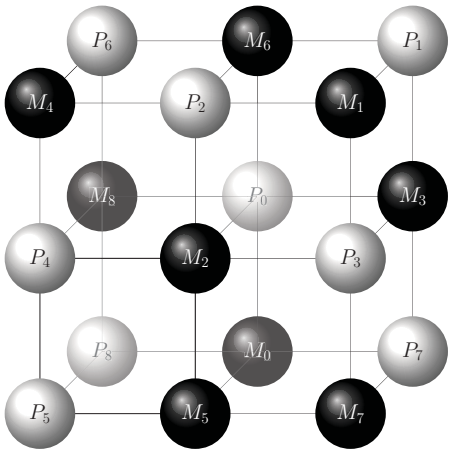

Figure 5.10 – Free Topology Assignment Model for F^1

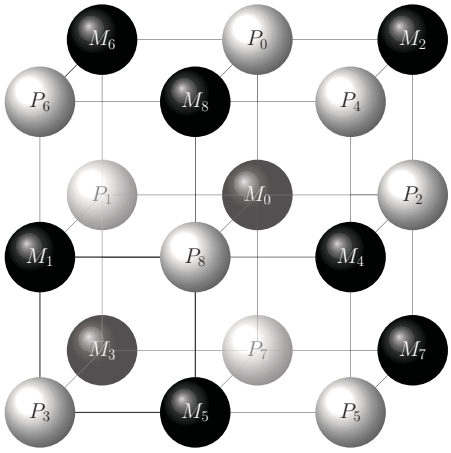

Figure 5.11 – Eliminate Proximity of Groups and Proximity of Families Model for F^2

memory space. The traditional cache coherence and synchronization primitives are used to keep all the memories of the network up to date [81]. The intensive communication assignment creates a straightforward connection for distributed-shared memory organization. Each memory is considered as an autonomous element of the network but can be accessed by any other element. In this context, algorithms must be provided to share data across the system (*distribution*) and to preserve the data coherence of all shared memory space (*synchronization*). As the main target of the system design is performance, the coherence and distribution algorithms must minimize access latency and overhead of synchronization management respectively. Many algorithms are provided across the

literature [112], [1] and [135] to solve the coherence and the synchronization problems.

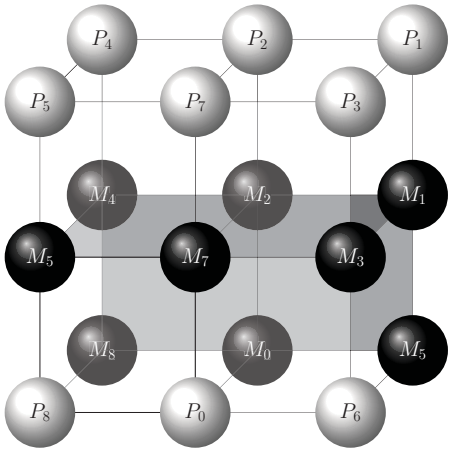

Figure 5.12 – First Memory Intensive Communication Model for F^3

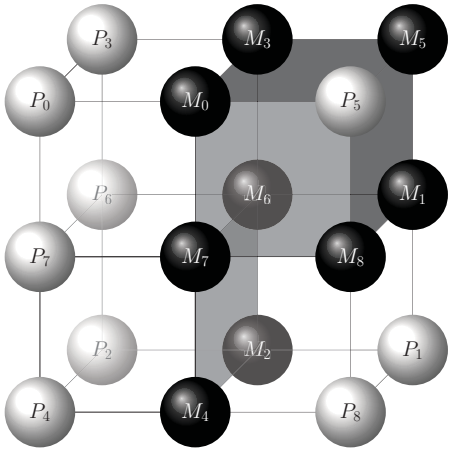

Figure 5.13 – Second Memory Intensive Communication Model for F^3

5.8.3 Link Perturbation Assignment

In the following, we illustrate the perturbation model proposed in Section 5.7. We use the flow matrix F^1 to perform QAP resolution for the perturbation model (See Section 5.6.2). In Figures 5.14-5.17, the *QAP resolution cost* axis represents the assignment cost of QAP every time link capacities are perturbed. The *Quantitative Metric* axis represents a metric that is desired by the designer. The *Link Capacity Assignment Iteration*

axis represents iterations when the link capacities change. Each iteration is associated with a set of link capacities. In Figures 5.14, 5.15 and 5.17, a change of abscissa (i.e., Link Capacity Assignment Iteration axis) indicates when all the link capacities change from one penalty to another (i.e., in 2, all the 33 links capacities of OCM are penalized by 2 while at 3, they are penalized by 3). Between these abscissas, we perturb some percentage of the link capacities of OCM.

In Figures 5.14 and 5.15, we perturb at each iteration 20% of the link capacities of OCM. The new values are changed from 1 to 2, then from 2 to 3. The last iteration changes the link capacities from 4 to 5. Figure 5.14 shows the result of the simulation and shows a parabolic behavior of QAP resolution cost (i.e., cost of the corresponding QAP solution). We can conclude that up to 60% of the link capacities can be changed and keep a reasonable QAP resolution cost while increasing the performance of the computation. Figure 5.15 illustrates the behavior of a random variation in $\{1,2,3,4,5\}$ of link capacities. We observe that the behavior of the plot is almost linear and close to the $y = 24718 \times x$, which is the multiplication of QAP resolution cost by the percentage of the link capacities that have changed. When all link capacities change from 1 to 5, QAP resolution cost changes from 24718 (0% changed) to 123590 (100%). Figure 5.16 shows QAP resolution cost of link capacity perturbation on the set $\{1,5\}$. We change the 33 links of the 3D network shown in Figure 5.1 one by one (representing 3.33% of the total links in the network) as follow : $\underbrace{1,1,1,\ldots,1,1}_{33}$ for the first iteration, $\underbrace{1,1,1,\ldots,1,5}_{33}$ for the second iteration, $\underbrace{1,1,1,\ldots,5,5}_{33}$ for the third iteration and so on up to $\underbrace{5,5,5,\ldots,5,5}_{33}$ for the 33^{rd} iteration. The result of the assignments shows a polynomial distribution from 1 to 5 and we can partially conclude that, like in the case of Figure5.14, that up to 90% of the value of the links can change while keeping a reasonable QAP resolution cost change and increasing the performance of the system. The Figure 5.17 illustrates the feasible solution by the integration of Figure 5.14, 5.15, 5.16 and 5.16. The projection of the desired Quantitative Metric to the plots of Figures 5.14-5.17 gives QAP resolution cost for this solution and the corresponding link capacities set on the Link Capacity Assignment Iteration axis. The link capacities associated with this iteration are considered

to be the best solution.

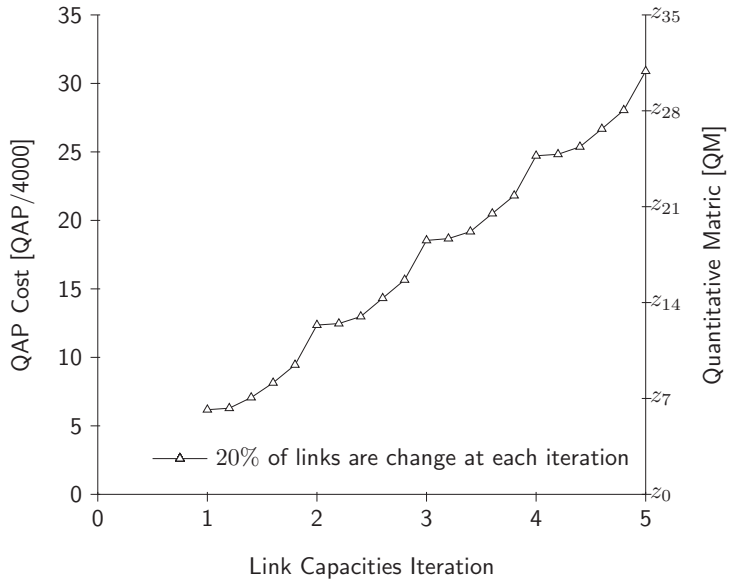

Figure 5.14 – Assignment Cost vs One Step Link Penalty

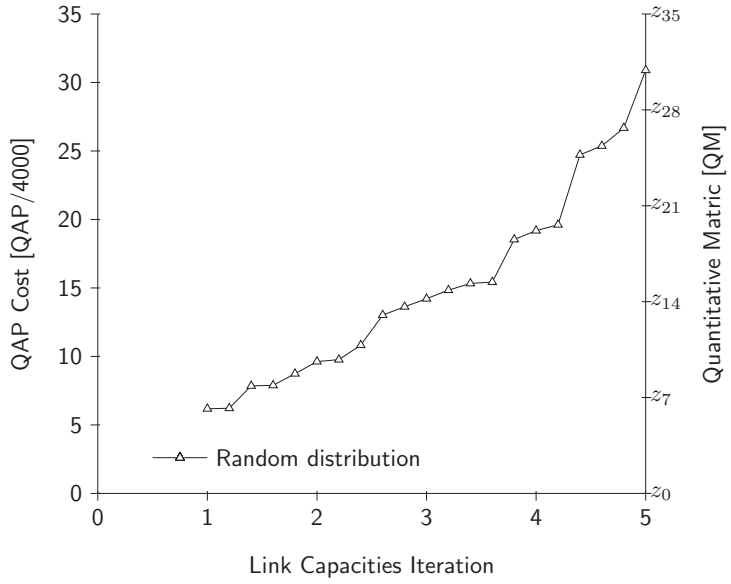

Figure 5.15 – Assignment Cost vs Random Link Penalty in {1,2,3,4,5}

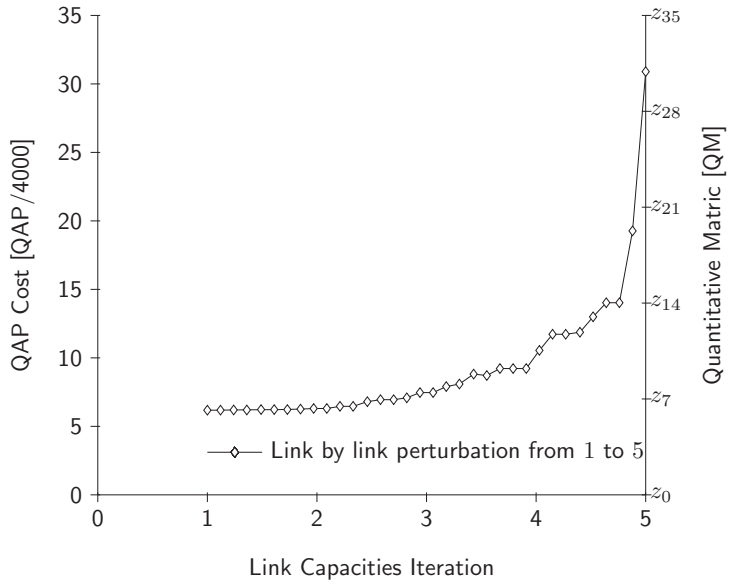

Figure 5.16 – Assignment Cost vs One Link Penalty by factor 5

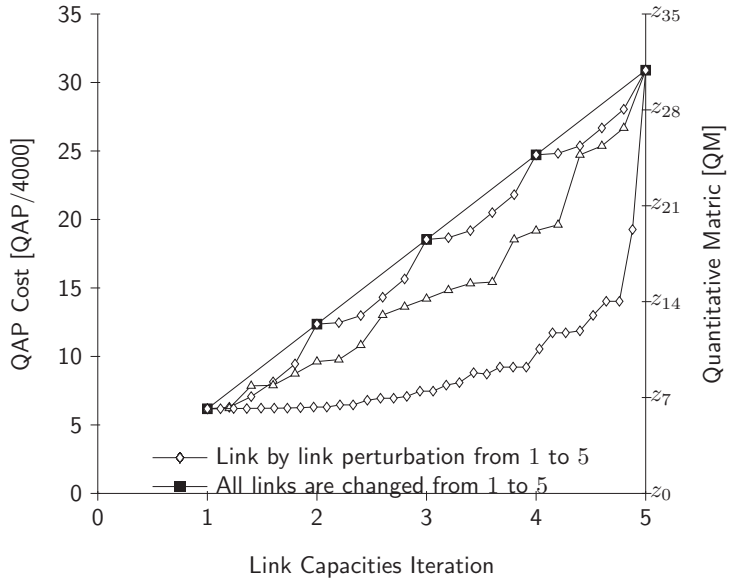

Figure 5.17 – Feasible Region for Assignment Cost vs Link Penalty

5.9 Conclusion

In this paper, we address two complementary On-Chip Multiprocessor design problems : (1) the problem of efficient assignment of components to provide the best performance by reducing to the minimum inter component communication ; and (2) the problem of link capacity assignment that assigns to the edges of OCM the best communication capacities between any two neighboring components. For the first problem, we proposed a novel multi-phase modeling solution that transforms the component assignment problem into QAP. More specifically, the proposed modeling solution transforms the interaction matrix (IA), which represents the input to the assignment problem, into a flow matrix which can be used as the input to QAP. Thus, using existing QAP resolution methods, we are able to assign components (processors and memories) to OCM. For the second problem, we proposed a new link capacity assignment method that assigns best capacities to the edges of OCM configuration ; the proposed method is based on a perturbation approach derived from perturbation theory [61]. The objective is to produce OCM configuration that optimizes a given metric (e.g., power consumption). The implementation in C++ and the experiments we performed show that, with the assignment of components, any configuration of OCM can be designed through the modeling of the flow matrix at a high level of abstraction (specification or system architecture level). The combination of component and link capacity assignment does not only provide a scalable and flexible OCM design environment but also guarantees an effective performance for the designed OCM.

In the future, we plan to develop a Simulink based library to provide different configurations for OCM design ; we also plan to conduct FPGA or ASIC based simulations to determine link capacities for specific performance metrics. Another research avenue includes the design of verification tools to verify cell interfaces based on specification and the results of our proposed solutions. Since the proposed assignment method is done at application specification level, future work may also include code generation for simulation and OCM synthesis.

Acknowledgment

The authors would like to thank Jonathan Bellemare for his precious help in implementing the proposed algorithms.

CHAPITRE 6

CONCLUSION

Dans le but d'optimiser les performances des systèmes multiprocesseurs sur puce électronique (On-Chip Multiprocessor [OCM]), une architecture est proposée dans cette thèse. Des problèmes de leur conception dès la spécification fonctionnelle ou l'architecture sont abordés. Parmi ces problèmes, nous avons la proximité des données, la décomposition d'une spécification d'application et la recherche du nombre optimal de n œuds dans un système OCM, et l'affectation efficace de n œuds dans les réseaux de processeurs. Les résultats de quelques réponses à ces problèmes sont résumés dans la section 6.1 tandis que nous discutons dans la section 6.2 de certaines pistes de recherches futures pouvant améliorer les solutions proposées.

6.1 Récapitulation des travaux

Nous avons proposé dans cette thèse trois nouvelles approches permettant d'aborder respectivement les trois problèmes posés ci-dessus. Dans la première contribution présentée dans le chapitre trois, nous nous sommes penchés sur la performance des systèmes OCM en fonction de l'organisation de leurs composantes. Avec la réduction de la taille des transistors et l'augmentation de la densité fonctionnelle des circuits intégrés, les systèmes OCM sont devenus le paradigme dans la conception des circuits intégrés. Plusieurs processeurs et mémoires sont ainsi embarqués sur la même puce électronique pour la création d'un système performance. Cependant, l'accès aux données reste le maillon faible et le plus grand défi de la recherche de cette performance. Selon Dally et Towles [46], la solution à ce problème ne passe pas par la surenchère de la performance des processeurs mais par l'interconnexion de ces composantes (processeurs et mémoires). Ainsi, le défi majeur de l'industrie et la communauté académique est de trouver de meilleures façons d'interconnecter les composantes des OCM.

Dans notre contribution, nous avons proposé une approche d'organisation des pro-

cesseurs et des mémoires, appelée architecture isométrique des systèmes multiprocesseurs sur puce (Isomectric On-Chip Multiprocessor Architecture [ICMA]), basée sur la structure cristalline du chlorure du sodium ($NaCl$). Cette structure est une superposition tridimensionnelle de plusieurs couches dans lesquelles les processeurs (identifiés aux sodiums [Na^+]) et les mémoires (identifiées aux chlores [Cl^-]) sont connectés les uns aux autres en trois dimensions dont ces derniers forment les sommets et de façon entrelacée. Cette approche permet de connecter six processeurs à une mémoire et six mémoires à un processeur. L'idée dernière cette structure est de former de petits groupes de processeurs autour des mémoires et vice-versa sans rendre ces dernières exclusives aux processeurs comme dans le cas des architectures à mémoires distribuées conventionnelles. Les données sont ainsi rapprochées des processeurs selon les dépendances. Cette architecture réduit les goulots d'étranglement dans les calculs. En effet, la combinaison des méthodes spéculatives des processeurs et l'organisation que nous proposons dans cette contribution élimineraient les délais d'accès aux données dans les mémoires. Un autre avantage de cette structure est le prototypage rapide. Nous considérons les cubes ou les couches comme des éléments de base de l'architecture (Commodity Nodes) [138] sur lesquels les emplacements des processeurs et des mémoires sont remplis en fonction de la configuration souhaitée lors de la conception du système OCM. Ce qui facilite le prototypage ou l'émulation d'un système avant son développement. Les résultats des simulations et certaines analyses ont montré les performances de l'approche proposée.

La deuxième contribution porte sur la décomposition de la spécification d'une application (spécification ou architecture) pour le ICMA. Plus spécifiquement, comment prendre une application, la décomposer afin de l'exécuter sur une architecture ICMA, ou comment prendre une spécification fonctionnelle, la décomposer afin de créer une architecture ICMA pour un système dédié. En effet, l'un des aspects les plus importants des traitements parallèles et distribués, ou de la conception d'un système multiprocesseur est la partition de l'application qui est exécutée sur ce système ou la partition de la spécification du dit système en vue de sa conception. Dans cette deuxième contribution, nous avons proposé une méthode qui est basée sur le problème de la formation des cellules (Cell Formation Problem [CFP]) de la technologie des groupes. Nous avons

proposé un modèle modifié du CFP qui généralise la version $zro-un$ du problème à une variante de nombres entiers. Ces nombres modélisent les interactions entre les processeurs et les mémoires d'un système OCM. En effet, nous transformons une application ou une spécification en matrice d'incidence où les lignes représentent les processus de cette application et les colonnes les mémoires. Le modèle proposé réorganise les lignes et les colonnes selon une règle de similarité, préalablement définie, de telle sorte que les processus et les données qui ont de très fortes interactions se retrouvent dans une même cellule. Notre méthode propose un modèle à objectifs multiples basé sur des contraintes comme la taille des cellules, le nombre de cellules, les poids des éléments exceptionnels dans le résultat, et l'équilibrage des charges dans les cellules. Vue la complexité du problème, nous avons proposé un algorithme heuristique à trois étapes où deux méthodes de résolution méta-heuristiques sont utilisées pour raffiner la solution initiale. Dans une expérience numérique, nous avons utilisé notre modèle de décomposition pour choisir un nombre optimal de n œuds dans la conception d'un système multiprocesseur sur puce (OCM) étant donné une application. Ce modèle peut par ailleurs être utilisé pour la décomposition d'une application écrite pour un processeur conventionnel en vue de son exécution sur un système parallèle ou distribué.

La troisième contribution porte sur l'affectation des composantes (processeurs et mémoires) aux n œuds d'un réseau de systèmes multiprocesseurs sur puce OCM d'une part, et sur l'affectation des capacités des liens entre ces n œuds, d'autre part. Rappelons que le problème d'affectation des liens est un problème rarement abordé dans la littérature. Dans le cas de l'affectation des composantes nous avons utilisé le problème d'affectation quadratique (Quadratique Assignment Problem [QAP]). Dans le cas de l'affectation des capacités des liens, nous avons utilisé une méthode de perturbation progressive des liens qui permet d'approximer une solution souhaitée pour la performance du système. Dans l'affectation des composantes, une approche est proposée pour modéliser les flots entre les composantes selon différentes topologies. La matrice d'interaction générée par le problème de la formation des cellules dans la deuxième contribution est utilisée comme entrée de cette approche. L'affectation des liens quant à elle utilise le résultat de l'affectation des composantes et change progressivement les capacités des liens entre ces

composantes. Le but étant de trouver les meilleures capacités des liens qui conjuguent une architecture performante et la prise en compte de métriques comme par exemple l'énergie consommée par le système, la surface occupée ou la dissipation de chaleur. Les aspects de l'affectation proposée dans cette contribution ont été appliqués sur une expérience numérique, qui a montré la puissance et la flexibilité de notre méthode dans la prise de décision lors de l'architecture d'un système multiprocesseur sur puce. Outre la conception des systèmes multiprocesseurs, les approches de nos deux dernières contributions peuvent être utilisées pour l'affectation dynamique des tâches et la gestion des flots dans les réseaux déjà existants.

6.2 Discussion et travaux futurs

Nous devons admettre que les travaux de ce cette thèse n'ont permis que de faire un petit pas dans la direction d'une architecture optimale des systèmes sur puce et des méthodologies qui leur sont associées. Nous savons que plusieurs améliorations sont à envisager dans le futur. Nos contributions ont ouvert plusieurs pistes de recherches pour des travaux futurs.

Pour compléter l'architecture proposée dans la première contribution, nous pensons que la création d'une librairie d'interfaces peut faciliter l'intégration des processeurs et des mémoires, et accélérer le prototypage des OCM dès la phase de l'architecture. Ainsi, un environnement paramétrique permettra de générer en très peu de temps un modèle du système OCM souhaité dont les configurations vont permettre d'étudier ses caractéristiques avant de se lancer dans la conception définitive du système. Ceci permettra de réduire considérablement le temps de la conception du système.

La création d'un compilateur et d'un environnement matériel qui permettent des décompositions et des affectations automatiques des applications conventionnelles peut être envisagée parmi les travaux futurs. Ces derniers permettront de transformer des applications déjà existantes en des ensembles de cellules selon les caractéristiques de l'ICMA. Bien que nous ayons fait des expériences numériques sur l'architecture que nous avons proposée, nous avons utilisé des applications comme la multiplication ma-

tricielle ou les systèmes masses-ressorts où les décompositions et les affectations sont plus ou moins évidentes. L'étape suivante serait d'explorer l'architecture avec des applications dont les données sont moins régulières que les manipulations matricielles, e.g., l'apprentissage machine ou les problèmes de convolution avec des filtre très petits, afin d'établir la performances de l'architecture proposée. En général, nos résultats indiquent que la réduction de flots de données dans le système entre les composantes est la clé de la performance.

Dans le cas de la décomposition, nous pouvons envisager d'ajouter à nos méthodes un générateur de codes de langages de description matérielle qui permet de générer les modules synthétisables de telle sorte que le passage de l'analyse au développement du système OCM se fasse automatiquement. Aussi, l'introduction de contraintes temporelles d'une part, et des dépendances entre les données d'autre part dans la formation des cellules peut être une avenue intéressante à envisager dans des recherches futures.

Dans le cas de l'affectation des capacités des liens, nous avons utilisé une approche empirique qui consiste à faire des expériences successives et à tirer des conclusions selon les solutions recherchées. Nous pensons qu'une formulation du problème de l'affectation quadratique (QAP) avec les éléments de la matrice de coût considérés comme des variables plutôt que des constantes peut être envisagée. Nous pensons aussi qu'un formalisme de l'architecture proposée ici peut être envisagé selon une approche similaire au " Hierarchical Annotated Action Diagrams (HAAD) " proposée par Cerny *et al.*, [32].

BIBLIOGRAPHIE

[1] Agarwal, A. *et al.* 1995. *The MIT Alewife Machine : Architecture and Performance*. Proceedings of the 22nd Annual International Symposium on Computer Architecture, (ISCA '95), 2-13, June 22-24 1995, Santa Margherita Ligure, Italy.

[2] Aghaghi, Y., *et al.* 2001. *lmdundant Address BUS Encoding for Low Power*. In Proceedings of the International Symposium on Low Power Electronics and Design. 182-187.

[3] Agresti, A. 2002. *Categorical Data Analysis*. Wiley Series in Probability and Statistics, 2 edition, July 22, 2002, Hoboken, NY.

[4] Ahmad, I., and Dhodhi, M.K. 1995. *Task assignment using problem-space genetic algorithm*. Concurrency : Practice and Experience, vol. 7, no. 5, 411-428, August 1995.

[5] Aljaber, N., *et al.* 1997. *A tabu search approach to the cell formation problem*. Comput Ind Eng 32, 169-185.

[6] ALTERA Inc. 2010. *Nios II Processor Reference Handbook*. Version 10.1, http ://www.altera.com, December 2010, San Jose, CA.

[7] Andrés, C. and Lozano, S. 2006. *A particle swarm optimization algorithm for part-machine grouping*. Robotics and Computer-Integrated Manufacturing 22, 468-474.

[8] Arabie, P. and Hubert, L.J. 1990. *The bond energy algorithm revisited*. Systems, Man and Cybernetics, IEEE Transactions on, vol. 20, no. 1, 268-274.

[9] Arabie, P., *et al.* 1988. *Marketing applications of sequencing and partitioning of nonsymmetric and/or two-mode matrices*, in Data Analysis, Decision Support, and Expert Knowledge Representation in Marketing, W. Gaul and M. Schader, Eds. Berlin : Springer-Verlag, 215-224.

[10] Arató, P., *et al.* 2005. *Algorithmic aspects of hardware/software partitioning*. ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 10, no. 1, January 2005.

[11] ARC. 2002. *Customizing a Soft Microprocessor Core*. http ://www.arc.com.

[12] Artieri, A., *et al.* 2003. *NomadikTM Open Multimedia Platform for Next-generation Mobile Devices*. STMicroelectronics Technical Article TA305, disponible au http ://www.st.com.

[13] Asanovc, K., *et al.* 2006. *The landscape of parallel computing research : a view from berkeley*. Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, University of California at Berkeley, December 2006.

[14] Baer J-L., 2010. *Microprocessor Architecture : From Simple Pipelines to Chip Multiprocessors*. 1 edition, Cambridge University Press, NYC, NY.

[15] Balarin, F., *et al.* 1997. *Hardware-software co-design of embedded systems : the POLIS approach*. Kluwer Academic Publishers, Norwell, MA.

[16] Ballakur, A. and Steudel, H.J. 1987. *A within cell utilization based heuristic for designing cellular manufacturing systems*. In International Journal of Production Research, 25, 639-655.

[17] Banakar, R., *et al.* 2002. *Scratchpad Memory : Design Alternative for Cache on-Chip Memory in Embedded Systems*. In Proceedings of CODES.

[18] Banerjee K., *et al.*. 2001. *3-D ICs : a novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration* ; In Proceedings of the IEEE ; vol. 89, no. 5, May 2001 ; pp : 602-633

[19] Bell, S., *et al.* 2008. *TILE64 processor : A 64-core SoC with mesh interconnect*. In Proceedings of the International Solid-State Circuits Conference, IEEE, Los Alamitos, CA.

[20] Ben Miled Z., *et al.*. 1996. *Hierarchical Processors-and-Memory Architecture for High Performance Computing*. In Proceedings. of the 6th Symposium on the Frontiers of Massively Parallel Computation ; October 1996 ; pp. 355-362 ; Annapolis, MD.

[21] Benini, L., and De Micheli, G. 2002. *Networks on chips : A new SoC paradigm*. IEEE Computer magazine, vol. 35, 70-78, January 2002.

[22] Benini, L., *et al.* 2005. *Architectural, system level and protocol level techniques for power optimization for networked embedded systems*. VLSI Design. 18th International Conference, January 3-7 2005, 18.

[23] Ben Miled, Z., *et al.* 1996. *Hierarchical Processors-and-Memory Architecture for High Performance Computing*. In Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation, Annapolis.

[24] Bokhari, S.H. 1987. *Assignment Problems in Parallel and Distributed Computing*. Kluwer Academic Publishers, Boston, MA, 1987.

[25] Bononi, L., *et al.* 2007. *NoC topologies exploration based on mapping and simulation models*. In Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, DSD, August 2007, 543-546.

[26] Bousono-Calzon, C., and Manning, M.R.W. 1995. *The hopfield neural network applied to the quadratic assignment problem*. Neural Computing & Applications vol. 3, no. 2, 64-72, June 1995.

[27] Brassard, G. and Bratley, P. 1995. *Fundamentals of Algorithmics*. Prentice Hall ; First Edition, August 20 1995, Eaglewood Cliffs, NJ.

[28] Bultan, T., and Aykanat, C. 1992. *A new mapping heuristic based on mean field annealing*. Journal of Parallel and Distributed Computing, vol. 16, no. 4, 292-305, December 1992.

[29] Burbidge, J.L. 1997. *Production Flow Analysis for Planning Group Technology*. Clarendon Press, 1997.

[30] Burger, D., *et al.* 2004. *Scaling to the end of silicon with edge architectures*. Computer, vol. 37, no. 7, 44-55.

[31] Cela, E. 1998. *The Quadratic Assignment Problem : Theory and Algorithms*. Kluwer Academic Publishers Group, December 1998, Boston, MA.

[32] Cerny, E., *et al.* 1998. *Hierarchical Annotated Action Diagrams : An Interface-Oriented Specification and Verification Method*. Kluwer Academic Publisher Springer, 1st edition, October 31 1998, Norwell, MA.

[33] Chaudhary, V., and Aggarwal, J.K. 1993. *A generalized scheme for mapping parallel algorithms*. IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 3, 328-346, March 1993.

[34] Chen, T.C., *et al.* 2006. *A high-quality mixed-size analytical placer considering preplaced blocks and density constraints*. In IEEE/ACM Proceedings of International Conference on Computer-Aided Design (ICCAD'06), 187-192, November 5-9 2006, San Jose, CA.

[35] Chen, T.-C., *et al.* 2005. *NTUplace : a ratio partitioning based placement algorithm for large-scale mixedsize designs*. In Proceedings of ISPD, 236-238.

[36] Chockalingam, T., and Arunkumar, S. 1992. *A randomized heuristics for the mapping problem : the genetic approach*. Parallel Computing, vol. 18, no. 10, 1157-1165, October 1992.

[37] Chou, C., and Marculescu, R. 2008. *Contention-aware application mapping for networkon-chip communication architectures*. In Proceedings of the International Conference on Computer Design (ICCD), Lake Tahoe, CA, October 2008.

[38] Chu, W.W., 1969. *Optimal File Allocation in Multiple Computing System*. IEEE Transaction on Computer, vol. 18, no. 10, 885-889, October 1969, Washington, DC.

[39] Chu, C.H., 1995. *Recent advances in mathematical programming for cell formation*. Manufacturing Research and Technology, vol. 24, 3-46.

[40] Churchman C.W., *et al.*, 1957. *Introduction to Operatons Research*. John Wiley & Sons Inc, December 1957, New York, NY.

[41] Cohoon, J.P., and Paris, W.D., 1987. *Genetic placement*. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. CAD-6, no. 6, 956-964, November 1987.

[42] Connolly, D.T., 1990. *An improved annealing scheme for the QAP*. Eur. J. Oper. Res., vol. 46, no. 1, 93-100, May 1990.

[43] Courtois, P.J., 1977. *Decomposability : queueing and computer system applications*. New York : Academic Press.

[44] Dally, W.J., and Seitz, C.L., 1986. *The torus routing chip*. J. Distribution. Computer, vol. 1, no. 3,187-196.

[45] Dally, W.J., and Towles, B., 2001. *Route packets, not wires : on-chip interconnection networks*. In Proceedings of DAC, June 2001.

[46] Dally, W.J., and Towles, B., 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA.

[47] Dees, W.A. and Karger, P.G., 1982. *Automated rip-up and reroute techniques*. In Proc. 19th Design Automation Conf., 432-439.

[48] Del Cuvillo, J., *et al.*, 2006. *Towards a Software Infrastructure for Cyclops-64 Cellular Architecture*. In HPCS 2006, St. John's, Newfoundland, Canada, May 14-17 2006.

[49] Dobkin, R., *et al.*, 2006. *Fast Asynchronous Shift Register for Bit-Serial Communication*. In Proceedings of ASYNC'06.

[50] Dobkin, R., *et al.*, 2006. *High-Speed Serial Interconnect for NoC*. NoC Workshop, DATE'06.

[51] Dobkin, R., *et al.*, 2008. *Parallel vs. serial on-chip communication*. In Proceedings of the 2008 international workshop on System level interconnect prediction (SLIP '08). ACM, New York, NY, 43-50.

[52] Dongkook Park, S.E., *et al.*, 2008 *Mira : A multi-layered on-chip interconnect for router architecture*. In Proceedings of the International Symposium on Computer Architecture (ISCA), June 2008, 251-261.

[53] Douglas, B. E., and Ho, S-M. 2006. *Structure and Chemistry of Crystalline Solids*. Springer, New York, NY, Springer

[54] Dutta, S., *et al.*, 2001. *Viper : A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems*. IEEE Design & Test, vol. 18, no. 5, September 2001, 21 - 31.

[55] Dyer, M., *et al.*, 2009. *Design and Analysis of System on a Chip Encoder for JPEG2000*. Circuits and Systems for Video Technology, IEEE Transactions on. vol. 19, no.2, February 2009, 215-225.

[56] Eigenmann R. and Hassanzadeh S., 1996. *Benchmarking with real industrial applications : the SPEC High-Performance Group*. IEEE Computing in Science & Engineering ; Vol. 3, no. 1 ; Spring 1996 ; pp. 18-23

[57] El-Dessouki, O.I., and Huan, W.H. 1980. *Distributed Enumeration on Network Computer*. IEEE Trans. Comput., vol. C-29, 1068-1079, December 1980.

[58] Erbas, C., *et al.* 2006. *Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design*. IEEE Transactions on Evolutionary Computation, vol. 10, no. 3.

[59] Ernst, R., Henkel J., and Benner, T. 1993. *Hardware-Software Cosynthesis for Microcontrollers*. IEEE Design & Test, vol. 10, no 4, 64-75, 1993.

[60] Fei, S., *et al.* 2005. *Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors*. VLSI Design, 18th International Conference on. 551-556.

[61] Fernandez, F.M. 2000. *Introduction to Perturbation Theory in Quantum Mechanics*. CRC Press, 1 edition, September 19 2000, Boca Raton, FL.

[62] Fienberg, S.E. 2007. *The Analysis of Cross-Classified Categorical Data*. Springer ; 2nd edition, July 30, 2007, Pittsburgh, PA.

[63] Flachs, B. *et al.* 2006. *The microarchitecture of the streaming processor for a Cell processor*. IEEE Journal of Solid-State Circuits, vol. 41, no. 1, January 2006, 63-70.

[64] Fleurent C. and Ferland, J.A. 1993. *Genetic Hybrids for the Quadratic Assignment Problem*. DIMACS Series in Mathematics and Theoretical Computer Science, American Mathematical Society, 173-187.

[65] Flynn M.J. 1995. *The role of representation in computer architecture*. Massively Parallel Processing Using Optical Interconnections, 1995., In Proceedings of the Second International Conference on, 23-24 October 1995, 36-41, San Antonio, TX

[66] Gambardella, L. M., *et al.* 1999. *Ant colonies for the quadratic assignment problem* J. Oper. Res. Soc., vol. 50, no. 2, 167-176, Febuary 1999.

[67] Gass, S. 1958. *Linear Programming*, McGraw-Hill, NY, 1958.

[68] Ghosh, A., *et al.* 1992. *Sequential Logic Testing and Verification*. Kluwer Academic Publishers, Norwell, MA.

[69] Glass, C.J., and Ni, L.M. 1992. *The turn model for adaptive routing*. In Proceedings of the International Sympodium on Computer Architecture (ISCA), May 1992, 278-287.

[70] Glover, F., and Laguna, M., 1997. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

[71] Guarini, K. W., *et al.* 2002. *Electrical Integrity of State-of-the-Art 0.13 μm SOI CMOS Devices and Circuits Transferred for Three-Dimensional (3D) Integrated Circuit (IC) Fabrication*. IEDM Tech. Digest, 943.

[72] Guerrier, P., and Greiner, A. 2002. *A Generic Architecture for On-Chip Packet-Switched Interconnections*. In Proceedings of DATE 2000, Pans, France, March 2002, 250-256.

[73] Gupta, R. K., and Michelli, G. D. 1993. *Hardware-Software Cosynthesis for Digital Systems*. IEEE Design and Test. Comput., vol. 10, no. 3, September 1993, 29-41.

[74] Gylys V.B. and J.A. Edward. 1976. *Optimal Partitioning of Workload for Distributed Systems*. In Dig. COMPCON 1976, 353-357, Fall 1976.

[75] Ha, S., *et al.* 2007. *PeaCE : a hardware-software codesign environment for multimedia embedded systems*. ACM Transactions on Design Automaton of Electronic Systems (TODAES), 1-25.

[76] Haddad, E. 1995. *Optimal load distribution for asynchronously scheduled homogeneous multiprocessor and distributed systems*. HICSS, 28th Hawaii International Conference on System Sciences (HICSS'95), 183.

[77] Held, J., *et al.* 2006. *From a few cores to many : A tera-scale computing research overview*. Intel White Paper.

[78] Hemani A., et al. 2000. *Network on a chip : an architecture for billion transistor era*. Proceedings of the 18th IEEE Norchip Conference ; November 5-6, 2000 ; Turku, Finland

[79] Henkel, J., *et al.* 2004. *On-chip networks : a scalable, communication-centric embedded system design paradigm*. VLSI Design, In Proceedings of the 17th International Conference on. January 5-9 2004, Mumbai, India.

[80] Henkel, J., *et al.* 2005. *H. 264 HDTV Decoder Using Application-Specific Networks-On-Chip*. Multimedia and Expo, ICME 2005. IEEE International Conference on. July 6 2005, 1508-1511.

[81] Hennessy, J., and Gross, T. 1983. *Postpass code optimization of pipeline constraints*. ACM Transactions on Programming Languages and Systems 5 (3), July 1983, 422-448.

[82] Hennessy, J., and Patterson, D.A. 2003. *Computer Architecture : a Quantitative Approach*. Morgan Kaufmann Publishers ; 3rd Edition, San Francisco, CA.

[83] Chu, W.W., Holloway, L.Y., Lan, M.T., and Efe, K. 1980. *Task allocation in distributed data processing*. IEEE Computer 13, November 1980, 57-69.

[84] Hu, J., and Marculescu, R. 2003. *Energy-aware mapping for tilebased NoC architectures under performance constraints*. In ASPDAC : ACM, In Proceedings of the 2003 conference on Asia South Pacific design automation, New York, NY, 233-239.

[85] Hu, J., and Marculescu, R. 2005. *Communication and task scheduling of application-specific networks-on-chip Computers and Digital Techniques*. IEE Proceedings-vol. 152, no. 5, September 9 2005, 643 - 651.

[86] Huerta, P., *et al.* 2005. *A MicroBlaze Based Multiprocessor SoC*. WSEAS Transactions on Circuits and Systems, vol. 4, no. 5, Grece, 423-430.

[87] Ienne, P., and Leupers, R. (EDS.) 2006. *Customizable Embedded Processors*. Morgan Kauffman, San Francisco.

[88] ILOG. 2006 *CPLEX 10.0. User's Manual*

[89] INTEL. 2008. *Intel Tera-scale Computing Research Program*. http ://www.intel.com.

[90] Jantsch, A., and Tenhunen H. 2003. *Networks on Chip*. Kluwer Academic Publishers, February 2003.

[91] Jensen, M.T. 2003. *Reducing the run-time complexity of multiobjective EAs : the NSGA-II and other algorithms*. IEEE Transactions on Evolutionary Computation, vol. 7, no. 5, 503-515.

[92] Jiang, X., *et al.* 2005. *A methodology for design, modeling, and analysis of networks-on-chip*. Circuits and System, ISCAS 2005. IEEE International Symposium on. vol. 2, May 23-26, 1778 - 1781.

[93] Jiang, Z.-W., *et al.* 2006. *NTUplace2 : A hybrid placer using partitioning and analytical techniques*. In Proceedings of ISPD, 215-217.

[94] Johnson, D.B. 1977. *Efficient Algorithms for Shortest Paths in Sparse Networks*. Journal of the Association for Computing Machinery, Vol 24, No 1, January 1977, 1-13.

[95] Jose, A.P., *et al.* 2006. *Pulsed Current-Mode Signaling for Nearly Speed-of-Light Intrachip Communication*. Journal of Solid-State Circuits, vol. 41, no. 4, 772-780.

[96] Kalavade, A., and Lee E.A. 2002. *The Extended Partitioning Problem : Hardware/Software Mapping, Scheduling and Implementation-bin Selection*. In Hardware/Software co-design, Kluwer Academic Publishers, 2002.

[97] Kafil, M., and Ahmad, I. 1998. *Optimal task assignment in heterogeneous distributed computing systems*, IEEE Concurrency 6 July-September 1998, 42-51.

[98] Kangmin, L., *et al.* 2004. *SILENT : serialized low energy transmission coding for on-chip interconnection networks*. ICCAD, International Conference on Computer Aided Design (ICCAD'04), 448-451.

[99] Kirkpatrik, S., *et al.* 1983. *Optimization by simulated annealing*. Science 220, May 1983, 671-680.

[100] KoopmansT., and Beckmann, M. 1957. *Assignment problems and the location of economic activities Econometrica*. vol. 25, no. 1, 53-76.

[101] Kumar, S. and Chandrasekharan M.P. 1990. *Grouping Efficacy : A Quantitative Criterion for Goodness of Block Diagonal Forms of Binary Matrices In Group Technology*. International Journal of Production Research 28, 233-243.

[102] Kumar, S., *et al.* 2002. *A network on chip architecture and design methodology*. IEEE Computer Society Annual Symposium on VLSI, 117-124, April 25-26 2002, Pittsburgh, PA.

[103] Kumar, R., *et al.* 2004. *Single-ISA heterogeneous multi-core architectures for multithreaded workload performance*. Computer Architecture, In Proceedings of the 31st Annual International Symposium on. June 19-23 2004, 64-75.

[104] Kunito, T., *et al.* 1989. *Three-Dimensional IC's Having Four Stacked Active Device Layers*. IEDM Tech. Digest, 837.

[105] Kurtzberg, J.M. 1962. *On Approximation Methods for the Assignment Problem*. J. ACM 9, 4, 419-439, October 1962.

[106] Lahiri, K., *et al.* 2004. *Design space exploration for optimizing onchip communication architectures*. IEEE Transactions on Computer-Aided Design on Integrated Circuits and Systems 23 (6).

[107] LambrechtsA., *et al.* 2005. *Power Breakdown Analysis for a Heterogeneous NoC Platform Running a Video Application"; Application-Specific Systems, Architecture Processors*. ASAP 2005 ; 16th IEEE International Conference on July 23-25 2005, 179-184.

[108] Lee F.E. 1993 *A Scalable Computer Architecture for Lattice Gas Simulations*. PhD thesis, Department of Electrical Engineering, Stanford University. May 1993

[109] Lei, T., and Kumar, S. 2003. *A two-step genetic algorithm for mapping task graphs to a network on chip architecture*. In Proceedings of the Euromicro Symposium on Digital System Design, 180-189.

[110] Lengauer, T. 1990. *Combinatorial algorithms for integrated circuit layout*. John Wiley & Sons, Inc. New York, NY.

[111] Li, K. 1988. *Ivy : A Shared Virtual Memory System for Parallel Computing*. In Proceedings of the 1988 International Conference on Parallel Processing, 94-101.

[112] Li T., and Wolf, W. H. 1999. *Hardware/Software Co-Synthesis with Memory Hierarchies*. IEEE Transactions on Computer, vol. 18, no. 10, November 1999, 1405-1417.

[113] Lin, F.T. and Hsu, C.C. 1990. *Task assignment scheduling by simulated annealing*. IEEE Region 10 Conference on Computer and Communication Systems, 279-283, September 24-27 1990.

[114] Lin, T.L., *et al.* 1996. *A heuristic-based procedure for the weighted production-cell formation problem*. IIE Trans. 26, 579-589.

[115] Lo V.M., *et al.* 1991. *OREGAMI : Tools for Mapping Parallel Computations to Parallel Architectures*. Intl Journal of Parallel Programming, vol. 20, no. 3, 237-270.

[116] López-Vallejo, M. and López, J.C. 2003. *On the Hardware-Software Partitioning Problem : System Modeling and Partitioning Techniques*. ACM Transactions on Design Automation of Electronic Systems, vol. 8, no. 3, pp. 269-297, 2003.

[117] Ma, P.R., *et al.* 1982. *A task allocation model for distributed computing systems*. IEEE Transactions on Computers, 41-47, January 31 1982.

[118] Mak, R.H. 2002. *Design and performance analysis of buffers : a constructive approach*. In Proceedings of ASYNC, 137-148.

[119] Maniezzo, V., and Colorni, A. 1999. *The ant system applied to the quadratic assignment problem*. IEEE Trans. Knowl. Data Eng., vol. 11, no. 5, 769-778, Sep./Oct. 1999.

[120] Marchioro, G. F., *et al.* 1997. *Transformational Partitioning for Co-Design of Multiprocessor Systems*. In Proceedings of the 17th ICCAD'97, 508-515.

[121] McAuley, J. 1972. *Machine grouping for efficient production*. The Production Engineer, 53-57.

[122] McCormick Jr, W.T., *et al.* 1972. *Problem decomposition and data reorganization by a clustering technique*. Operations Research, vol. 20 no. 5, 993-1009.

[123] Moore, S.K. ; , *Multicore is bad news for supercomputers*. Spectrum, IEEE , vol.45, no.11, pp.15, November 2008

[124] Murali, S., and De Micheli, G. 2004. *Bandwidth-constrained mapping of cores onto NoC architectures*. In DATE, IEEE Computer Society, 896-903, 2004.

[125] Murphy, J., and Klask, K. 2008. *Designing an interactive GUI*. EDA Tech Forum, vol. 5, no. 5, December 2008, 22-26.

[126] Nayfeh, B. A. 1998. *The Case for a Single-Chip Multiprocessor*. PhD Dissertation, Computer Systems Laboratory, Stanford University, September 1998, Stanford, CA.

[127] Ng, S.M. 1993. *Worst-case analysis of an algorithm for cellular manufacturing*. European Journal of Operational Research, 69, 384-398, 1993.

[128] Ni, L.M., and Mckinley, P.K. 1993. *A survey of wormhole routing techniques in direct networks*. Computer, 26, 62-76.

[129] Nicopoulos, C.A., *et al.* 2006. *Vichar : A dynamic virtual channel regulator for networkon-chip routers*. In Proceedings of the 39th Annual International Symposium on Microarchitecture (MICRO), Orlando, FL, December 9-13 2006, 333-344.

[130] Niemann, R., and Marweldel, P. 1998. *Hardware/software co-design for data flow dominated embedded systems*. Kluwer Academic Publisher Springer, 1st edition, October 31 1998, Norwell, MA.

[131] Nissen, V. and Paul, H. 1995. *A modification of threshold accepting and its application to the quadratic assignment problem*. OR Spektrum, vol. 17, no. 2/3, 205-210, June 1995.

[132] Ogoubi, E., *et al.* 2007. *The Isometric on-Chip Multiprocessor Architecture*. In 14th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Morocco, 2007.

[133] Ogoubi, E., *et al.* 2006. *The Isometric on Chip Multiprocessor Architecture*. Rapport Technique No. 1287, Université de Montréal, Montréal, Québec.

[134] E. Ogoubi, *et al.* "A New Multi-Phase Cell Formation Problem Method for Application Decomposition for On-Chip Multiprocessor System Design". Submitted to IEEE Transaction on Parallel and Distributed Systems.

[135] Ojima, Y., *et al.* 2005. *Design of software distributed shared memory system using MPI communication layer*. In Proc. The 4th International Workshop on OpenMP : Experiences and Implementations WOMPEI, January 18-25 2005, Tsukuba, Japan.

[136] Owens, J.D., *et al.* 2007. *Research challenges for on-chip interconnection networks*. IEEE Micro, vol. 27, no. 5, September-October 2007, 96-108.

[137] Papaioannou, G., and Wilson, J.M. 2010. *The evolution of cell formation problem methodologies based on recent studies (1997-2008) : Review and directions for future research*. European Journal of Operational Research, vol. 206, no. 3, November 1 2010, 509-521.

[138] Parhami, B. 1999. *Introduction to Parallel Processing : Algorithms and Architectures*. Springer Publisher, 1ère edition, Plenum Press, New York, NY, January 31 1999.

[139] Pham, D.C., *et al.* 2006. *Overview of the Architecture, Circuit Design, and Physical Implementation of a First-Generation Cell Processor*. IEEE Journal of Solid-State Circuits, vol. 41, No. 1, January 2006.

[140] Pisinger, D. and Ropke, S. 2009. *Large neighborhood search Handbook of Metaheuristics*. Forthcoming, M. Gendreau and J.-Y. Potvin(eds), 2nd edition.

[141] Preas, B.T., and Lorenzetti, M. 1988. *Physical Design Automation of Vlsi Systems*. Benjamin-Cummings Pub Co, July 1988.

[142] Rojas, W. *et al.* 2004. *An Efficient Genetic Algorithm to Solve the manufacturing Cell Formation Problem*. Adaptive Computing in Desing and Manufacture VI, Springer-Verlag(1), 173-188.

[143] Roy, J., *et al.* 2006. *Satisfying whitespace requirements in top-down placement*. In Proceedings of ISPD, 206-208.

[144] Saastamoinen, I., *et al.* 2001. *Interconnect IP for Gigascale System-on-chip*. In Proceedings of the European Conference Circuit Theory and Design (ECCTD), 281-284.

[145] Sahni, S., and Bhatt, A. 1980. *The complexity of design automation problems*. Annual ACM IEEE Design Automation Conference, Proceedings of the 17th Design Automation Conference, 402-411, Minneapolis, Minnesota.

[146] Salminen, E., *et al.* 2005. *HIBI-based multiprocessor SoC on FPGA*. Circuits and Systems. ISCAS 2005. IEEE International Symposium on. vol. 4, May 23-26 2005, 3351-3354.

[147] Sarker, B., and Khan, M. 2001. *A Comparison of Existing Grouping Efficiency Measures and a New Weighted Grouping Efficiency Measure*. IIE Transactions 33, 11-27.

[148] Schrimpf, G., *et al.* 2000. *Record breaking optimization results using the ruin and recreate principle*. Journal of Computational Physics, 159(2), 139-171.

[149] Sgroi, M., *et al.* 2001. *Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design*. In Proceedings of the Design Automation Conference 2001, Las Vegas, June 2001.

[150] Sitl, G., *et al.* 1991. *Analytical placement : A linear or a quadratic objective function*. In 28th ACM/IEEE Design Automation Conference (DAC), 427-432, June 21 1991, San Francisco, CA.

[151] Skorin-Kapov, J. 1990. *Tabu search applied to the quadratic assignment problem*. ORSA J. Comput., vol. 2, no. 1, 33-45.

[152] Spindler, P., and Johannes, F. M. 2006. *Fast and robust quadratic placement combined with an exact linear net model*. In Proceedings of ICCAD. November 5-9 2006, San Jose, CA.

[153] Srinivasan, V., Radhakrshnan, S., and Vemuri, R., *Hardware Software Partitioning with Integrated Hardware Design Space Exploration*. Proc. Design, Automation, and Test in Europe (DATE), 28-35, 1998.

[154] Stan, M. R., *et al.* 1997. *Law-Power Encoding for Global Communication in CMOS VLSI*. Transactions an VLSI Systems, vol. 5, no.4, June 1997.

[155] Stravers, P., and Hoogerbrugge, J. 2001. *Homogeneous multiprocessing and the future of silicon design paradigms*. VLSI Technology, Systems, and Applications. In Proceedings of Technical Papers. 2001 International Symposium on. April 18-20 2001, 184-187.

[156] Stützle, T. and Dorigo, M. 1999. *ACO algorithms for the quadratic assignment problem*. in New Ideas for Optimization, D. Corne, M. Dorigo, and F. Glover, Eds. New York : McGraw-Hill, 33-50, New-York, NY.

[157] Susan, X., and Pollitt-Smith, H. 2008. *A Multi-MicroBlaze Based SOC System : From SystemC Modeling to FPGA Prototyping*. The 19th IEEE/IFIP International Symposium on Rapid System Prototyping, 121-127.

[158] Suutari, T., *et al.* 2001. *High-speed Serial Communication with Error Correction Using 0.25 μm CMOS Technology*. In Proceedings of ISCAS, vol. 4, 618-621.

[159] Tabi, E.G. and Muntean, T. 1993. *Hill-climbing, Simulated Annealing and Genetic Algorithms : a Comparative Study and Application to the Mapping Problem*. IEEE 26th Hawaii International Conference System Sciences, 565-573.

[160] Taghavi, T., *et al.* 2006. *Dragon2006 : Blockage-aware congestion-controlling mixed-size placer*. In Proceedings of ISPD, 209-211.

[161] Taillard, E.D. 1991. *Robust taboo search for the quadratic assignment problem*. Parrallel Computing, vol. 17, 1991.

[162] Tamhankar, R. R., *et al.* 2005. *Performance driven reliable link design for networks on chips*. In Proceedings of the 2005 Asia and South Pacific Design Automation Conference (ASP-DAC '05). ACM, New York, NY, 749-754.

[163] Tate, D.M. and Smith, A.E. 1995. *A genetic approach to the quadratic assignment problem*. Comput. Oper. Res., vol. 22, no. 1, 73-83, January 1995.

[164] Teifel, J., and Manohar, R. 2003. *A High-Speed Clockless Serial Link Transceiver*. In Proceedings of ASYNC'03, 151-161.

[165] Topol, A. W., *et al.* 2006. *Three-dimensional integrated circuits*. Journal of Research and Development, IBM J. RES. & DEV. vol. 50, No. 4/5, July/September 2006, 491-506.

[166] Tunnukij, T. and Hicks, C. 2009. *An Enhanced Grouping Genetic Algorithm for solving the cell formation problem*. International Journal of Production Research, vol. 47, no. 7, 1989-2007, January 2009.

[167] Viswanathan, N., *et al.* 2006. *FastPlace 2.0 : An efficient analytical placer for mixed-mode designs*. In Proceedings of ASPDAC, 195-200.

[168] Viswanathan, N. and Chu, C.C.N. . 2005. *Fastplace : Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model*. IEEE Trarasactions on Computer-Aided Desig of Circuits and Systems, vol. 24, no. 5, 722-733, May 2005.

[169] Von Neumann, J. 1953. *A certain-zero-sum two-person game equivalent to the optimal assignment problem*. In H. Kuhn and A. Tucker (eds.), Contribution to the Theory of Games II (Ann. Math. Study No. 28), 5-12, Princeton University Press, 1953. Princeton, NJ.

[170] Wall, D.W. 1993. *Limits of instruction-level parallelism*. Technical Report WRL 93/6, Digital Western Research Laboratory, November 1993, Palo Alto, CA.

[171] Walrand, J., and Varaiya, P. 2000. *High-Performance Communication Networks*. Morgan Kaufman.

[172] Wang, H.S., *et al.* 2003. "Powerdriven design of router micro-architectures in on-chip networks," in Proceedings of the 36th Annual ACM/IEEE International Symposium on Micro architecture, 105-116, December 03-05 2003, Washington DC.

[173] Wentzlaff, D., *et al.* 2007. *On-chip interconnection architecture of the tile processor*. Micro, IEEE, vol. 27, no. 5, September 2007, 15-31.

[174] Wilhelm, M.R., and Ward, T.L. 1987. *Solving quadratic assignment problems by simulated annealing*. IEEE Trans., vol. 19, no. 1, 107-119.

[175] Wolf, W. 2004. *The future of multiprocessor systems-on-chips*. In Proceedings of the 41st annual conference on Design automation. San Diego, CA, 681-685.

[176] Wolkotte, P.T., *et al.* 2005. *An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip*. Parallel and Distributed Processing. In Proceedings of the 19th IEEE International Symposium on. April 4-8 2005, 155a-155a.

[177] XILINX CORP. 2009. *Xilinx MicroBlaze and PowerPC Processor Embedded Kit -Virtex-5 FX70T FPGA Edition, Getting Started Guide*. UG699 (v1.0.1), July 30 2009, San Jose, CA.

[178] XILINX CORP. 2008. *MicroBlaze Processor Reference Guide : Embedded Development Kit EDK 13.1*. UG081 (v12.0), March 2011, San Jose, CA.

[179] Xu, W., *et al.* 2008. *A Quantitative Study of the On-Chip Network and Memory Hierarchy Design for Many-Core Processor*. Parallel and Distributed Systems. ICPADS '08. 14th IEEE International Conference on. December 8-10 2008, 689-696.

[180] Yamada, J., *et al.* 2006. *High-speed interconnect for a multiprocessor server using over 1tb/s crossbar*. in IEEE ISSCC Digest Technical Papers, 343-352, February 2006.

[181] Yang, Y., *et al.* 2005. *High-performance systolic arrays for band matrix multiplication*. Circuits and Systems. ISCAS 2005. IEEE International Symposium on. May 23-26 2005, vol. 2, 1130-1133.

[182] Yin Y., and Yasuda K. 2006. *Similarity coefficient methods applied to the cell formation problem : A taxonomy and review*. International Journal of Production Economics, vol. 101, no. 2, June 2006. 329-352.

[183] Zhang, W.-T., *et al.* 2007. *Design of heterogeneous MPSoC on FPGA*. ASIC, 2007. ASICON '07. 7th International Conference on. October 22-25 2007, 102-105.

[184] Zipf, P., *et al.* 2004. *A switch architecture and signal synchronization for GALS system-on-chips*. Integrated Circuits and Systems Design. SBCCI 2004. 17th Symposium on. September 7-11 2004. 210-215.