N° d'ordre :

THESE

pour obtenir le grade de :

DOCTEUR DE L'UNIVERSITE DE BRETAGNE SUD

Spécialité : Sciences de l'ingénieur
Mention : Electronique et Informatique Industrielle

ISSAM MAALEJ

Exploration haut niveau des architectures multiprocesseurs : analyse et métriques

Soutenue publiquement le 16/10/2007

COMPOSITION DU JURY

Michel AUGUIN	Directeur de recherche CNRS, I3S, Nice	Examinateur
Frederic ROUSSEAU	Maître de conférence HDR TIMA, Grenoble	Rapporteur
Adel ALIMI	Professeur, REGIM, Sfax	Rapporteur
Jean Luc PHILIPPE	Professeur, LESTER, Lorient	Directeur de thèse
Mohamed ABID	CES, Ecole Nationale d'Ingénieur de Sfax	Directeur de thèse
Guy GOGNIAT	Maître de conférence, LESTER, Lorient	Examinateur

Laboratoire d'Electronique des Systèmes Temps Réel (LESTER, FRE2734)
Université de Bretagne Sud

Abstract

Architecture exploration is a fundamental step in the design flow of embedded systems since the decisions made at this level have a significant impact on the final performance of the system. Applications and architectures have evolved and are still evolving, which increases the complexity of architecture exploration approaches. Indeed, these approaches have reached their own limits and are less efficient in the processing of applications which include a huge number of tasks and multiprocessor applications with an increasing number of processors.

In this PhD, we discuss about the issue caused by multiprocessor architecture exploration with a high number of processors for applications that include many tasks. A reflection was necessary in order to identify and analyse difficulties caused by these applications. A new architecture exploration approach was implemented in order to deal with these new difficulties and with the ones caused by architecture exploration methods traditionally used. A multi-PACM architecture template was established to represent both architectures specification and its new parameters (proximity, parallelism and software diversity). This approach is divided in two steps. The exploration step is preceded by a pre-exploration step which takes place at a higher level (operational) in order to reduce architecture space as well as exploration costs and complexity. Pre-exploration step consists in distributing tasks into the architecture's PACMs. The distribution of tasks among the PACMs is call "partition". Pre-exploration step is a multi-objective optimisation aiming at maximising six metrics. The purpose of these metrics consists in optimising the distribution of data exchanges, data sharing, throughput constraints and partitions in order to optimise system time, surface and consumption. Replacing time, surface and consumption space used in traditional methods with the six metrics-based space allows to reduce exploration costs since metrics are less dependent on technology and represent a significantly less expensive information collection.

The suggested approach allows to deal with the increasing parallelism and the huge number of software and hardware resources. Furthermore, it allows decreasing exploration costs by reducing the number of performance estimations. This approach is flexible and helps the designer to enrich and guide the exploration.

UMTS transmitter, AC3 signal encoding and ICAM object tracking applications image are used to validate the metrics and their analysis through genetic algorithm, exploration approach and its ability to face with an extended design space, both in an application and architecture point of view.

Résumé

L'étape d'exploration architecturale est une étape critique dans le flot de conception des systèmes embarqués dans la mesure où les décisions prises à ce niveau impactent très fortement les performances finales du système. Les applications et les architectures ont évolué et continuent d'évoluer compliquant la tâche des approches d'exploration d'architecture. En effet, ces dernières ont atteint leurs limites et sont moins efficaces pour traiter des applications avec un nombre important de tâches et des architectures multiprocesseurs dont le nombre de processeurs croît continuellement.

Dans le cadre de cette thèse, le problème adressé est celui de l'exploration d'architecture multiprocesseur avec un nombre de processeurs élevé pour des applications ayant un nombre de tâches élevé. Une réflexion a été nécessaire afin d'identifier et analyser les nouvelles difficultés imposées par ces architectures. Une nouvelle approche d'exploration a été mise en œuvre pour faire face à la fois à ces nouvelles difficultés et à celles de l'exploration des architectures classiquement utilisées. Un modèle d'architecture multi-PACM a été élaboré pour représenter à la fois les architectures et les nouveaux paramètres (la proximité, le parallélisme et la diversité logicielle). Cette approche se base sur deux étapes. Nous précédons l'étape d'exploration par une étape de pré-exploration à un niveau supérieur (fonctionnel) afin de réduire l'espace des architectures et réduire par conséquent les coûts et la complexité de l'exploration. L'étape de pré-exploration consiste à distribuer les tâches entre les PACM de l'architecture. Une distribution des tâches entre les PACM est nommée une partition. L'étape de pré-exploration consiste donc en une optimisation multi-objectif visant à maximiser six métriques. Ces métriques ont pour but d'optimiser la distribution des échanges des données, des partages de données, de la distribution des contraintes de débit et des partitions afin d'optimiser le temps, la surface et la consommation du système. Remplacer l'espace temps, surface et consommation des approches actuelles par l'espace basé sue les 6 métriques a pour intérêt de réduire les coûts de l'exploration dans la mesure où les métriques sont moins dépendantes de la technologie et représentent donc un ensemble d'informations considérablement moins coûteux.

L'approche proposée permet de faire face au parallélisme croissant et au nombre de ressources logicielles et matérielles élevé. Elle permet aussi de réduire les coûts d'exploration en réduisant le nombre d'estimations des performances. L'approche est flexible et permet au concepteur d'enrichir et de guider l'exploration.

Les applications émetteur UMTS, codage des signaux AC3 et suivi d'objets dans une image ICAM ont été utilisées pour valider les métriques et leur analyse par l'algorithme génétique, l'approche d'exploration et sa capacité à appréhender un espace de conception étendu aussi bien du point de vue de l'application que de l'architecture

Remerciements

Ce document présente les travaux de thèse que j'ai effectués en cotutelle entre le laboratoire LESTER à Lorient en France et le laboratoire CES à Sfax en Tunisie. Je remercie les membres des deux laboratoires qui m'ont bien accueilli parmi eux et qui m'ont permis de réaliser mes travaux.

Je remercie M. Frédéric Rousseau et M. Adel Alimi d'avoir accepté d'être les rapporteurs de cette thèse et de faire partie du jury lors de la soutenance. Je remercie M. Michel Auguin qui accepté de présider le jury. Je les remercie pour l'intérêt qu'ils ont porté à mes travaux et pour l'aide qu'ils m'ont apporté.

Je remercie mon directeur de thèse en France, M. Jean-Luc Philippe, pour sa disponibilité et les discussions enrichissantes autour de mes travaux. Je remercie également M. Abid Mohamed, mon directeur de thèse en Tunisie, pour ses remarques enrichissantes. Je tiens à remercier mon encadrant Guy Gogniat pour son soutien et son aide tout au long de la thèse.

Je remercie les personnes que j'ai eu l'occasion de rencontrer au cours de mes années de thèse, étudiants et collaborateurs, pour leur aide, leur soutien et leurs conseils.

Je tiens aussi remercier chaleureusement ma famille, mes amis et toutes les personnes qui m'ont soutenu pendant la thèse et m'ont poussé à avancer durant la thèse. Je remercie spécialement Magali pour m'avoir soutenu même durant les phases difficiles de la thèse.

TABLE DES MATIERES

I. INTRODUCTION 8

1. CONTEXTE DE LA THESE 9
2. MOTIVATIONS 10
3. OBJECTIFS ET CONTRIBUTION 13
4. PLAN DE LA THESE 14

II. ETAT DE L'ART 16

1. INTRODUCTION 17
2. APPLICATIONS ET ARCHITECTURE MULTIPROCESSEUR 18
2.1. Evolution des architectures selon INTEL 19
2.2. Multiprocesseur Cell de IBM 20
2.3. Multiprocesseur CT3400 de CRADLE 20
2.4. Les architectures du futur 21
3. PERFORMANCES DES ARCHITECTURES 21
3.1. Temps d'exécution 21
3.2. Consommation 23
3.3. Surface 24
3.4. Résumé 25
4. COMPLEXITE DE L'EXPLORATION DES ARCHITECTURES 25
5. QUELQUES APPROCHES D'EXPLORATION 27
5.1. Exploration manuelle 27
5.2. Exploration directe d'architecture 28
5.3. Division de l'exploration en plusieurs parties 30
5.4. Plateformes 35
5.5. Analyse de haut niveau 36
5.6. Résumé 37
6. CONCLUSION 39

III. FLOT D'EXPLORATION D'ARCHITECTURES 42

1. INTRODUCTION 43
2. ARCHITECTURE MULTIPROCESSEUR ET NOUVEAUX DEFIS 45
2.1. Les ressources logicielles 46

2.2. Les ressources matérielles ... 47
2.3. Les ressources de communication ... 47
2.4. Les ressources de mémoires ... 48
2.5. Nouveaux défis des architectures ... 48
3. NOUVELLE APPROCHE D'EXPLORATION ... 49
4. MODELE D'ARCHITECTURE MULTI-PACM ... 52
4.1. Définition de l'architecture multi-PACM ... 52
4.2. Dimension spatiale ... 53
4.3. Parallélisme ... 54
4.4. Diversité logicielle ... 54
4.5. Généricité du modèle ... 55
5. ANALYSE DE HAUT NIVEAU ... 56
5.1. Analyse des tâches ... 56
5.2. Analyse du concepteur ... 57
5.3. Graphe de spécification ... 57
6. PRE-EXPLORATION ... 61
6.1. Réduction de l'espace d'architectures ... 61
6.2. Modèle d'analyse ... 62
6.3. Exploration de l'espace des partitions et d'architectures ... 64
7. CONCLUSION ... 65

IV. METRIQUES ET EXPLORATION ... 68

1. INTRODUCTION ... 69
2. METRIQUES ... 69
2.1. Les performances d'exécution des tâches ... 70
2.2. Les mémoires ... 74
2.3. Les communications ... 77
2.4. Parallélisme ... 83
2.5. L'espace des métriques ... 83
3. ALGORITHME GENETIQUE ... 84
3.1. Définitions ... 84
3.2. Formulation du problème ... 84
3.3. L'optimisation multi-objectif ... 86
3.4. Algorithme génétique ... 87
4. OUTIL GAMA² ET DECISIONS DU CONCEPTEUR ... 92
4.1. L'outil GAMA² ... 92
4.2. La décision du concepteur ... 93
5. CONCLUSION ... 94

V. APPLICATIONS ET RESULTATS ... 96

1. INTRODUCTION ... 97
2. EMETTEUR UMTS ... 97
2.1. Présentation de l'application ... 98
2.2. Graphe de spécification ... 99
2.3. Analyse et metriques ... 102
2.4. Exploration ... 111
2.5. Implémentation de l'émetteur UMTS ... 115
3. SYSTEME DE DECODAGE DE SIGNAUX AC3 ... 117
3.1. Présentation de l'application ... 118
3.2. Graphe de spécification ... 118
3.3. Exploration pour une architecture en deux PACM ... 120

3.4. Exploration par CODEF 121
3.5. Pré-exploration par GAMA² et CODEF 122
3.6. Comparaison des résultats des deux approches 125
4. APPLICATION ICAM 126
4.1. Présentation de l'application 127
4.2. Graphe de spécification 129
4.3. ICAM quatre flots 129
4.4. ICAM huit flots 134
4.5. Conclusion 135
5. CONCLUSION 136

VI. CONCLUSIONS ET PERSPECTIVES 137

1. CONCLUSION 138
1.1. Réponse à la problèmatique et travail réalisé 138
1.2. Résultats 140
1.3. Analyses critiques 140
2. PERSPECTIVES 140
2.1. Perspectives à court terme 141
2.2. Perspectives à moyen terme 141
2.3. Perspectives à long terme 142

LISTE DES FIGURES

Figure 1 : Courbe d'exploration d'architectures pour une application 11
Figure 2 : (a) Evolution de la courbe performance de l'architecture trouvée en fonction du coût d'exploration (b) Méthode pour augmenter le σ de l'exploration 13
Figure 3 : Chaîne de conception d'un circuit embarqué 17
Figure 4 : Evolution des processeurs selon INTEL 19
Figure 5 : Architecture de processeur Cell 20
Figure 6 : Temps d'exécution 22
Figure 7 : La complexité de l'exploration 26
Figure 8 : Flot de l'exploration manuelle 28
Figure 9 : Flot d'exploration CODEF 29
Figure 10 : Description des entrées et sorties de l'outil CODEF 29
Figure 11 : Approche d'exploration séparant partitionnement et communication 31
Figure 12 : Exploration de l'architecture par COSYN 32
Figure 13 : Approche d'exploration d'architectures par MOCSYN 34
Figure 14 : Modèle d'architecture MFSAM 35
Figure 15 : Flot proposé dans [Brandolese 2006] 37
Figure 16 : Approche classique d'exploration d'architectures 43
Figure 17 : Nouvelle approche d'exploration d'architectures 44
Figure 18 : Un exemple d'architecture multiprocesseur 45
Figure 19 : Influence de la position sur les communications 48
Figure 20 : Elévation du niveau de l'exploration 50
Figure 21 : Nouvelle approche d'exploration 51
Figure 22 : Architecture multi-PACM 52
Figure 23 : Exemple de proximité de l'architecture multi-PACM 53
Figure 24 : Généricité de l'architecture multi-PACM 55
Figure 25 : Exemple du graphe de spécification 58
Figure 26 : Pré-exploration et exploration 62
Figure 27 : Espace des métriques 64
Figure 28 : Exploration de l'espace d'architectures 65
Figure 29 : Distribution des données mémorisées 76
Figure 30 : Exemple d'architecture multiprocesseur 78
Figure 31 : Exemple de distribution des données 80
Figure 32 : Influence des métriques 83
Figure 33 : Classification générale des méthodes d'optimisation 86
Figure 34 : Boucle générationnelle de l'algorithme génétique 88
Figure 35 : Exemple d'un individu 89
Figure 36 : Code de la sélection par tournois 90
Figure 37 : Croisement 91

Figure 38 : Mutation ____ **91**
Figure 39 : Description des entrées et sorties de l'outil GAMA² ____ **92**
Figure 40 : Paramètres de l'exploration de GAMA² ____ **93**
Figure 41 : Exemple de quelques espace des métriques ____ **94**
Figure 1 : Figure 42 : Schéma de spécification de l'émetteur UMTS ____ **98**
Figure 43 : Exemple de spécification des tâches SRC et CRC de l'émetteur UMTS ____ **99**
Figure 44 : Exemple de spécification de deux arcs de l'émetteur UMTS ____ **100**
Figure 45 : Exemple d'un individu de l'émetteur UMTS ____ **102**
Figure 46 : Métriques de communication pour quelques exemples d'implémentation de l'émetteur UMTS en deux PACM ____ **103**
Figure 47 : Le vecteur de classification, la distribution des données et les métriques de 4 exemples ____ **104**
Figure 48 : (a) Les exemples de débits utilisés (b) Métriques contraintes pour l'émetteur UMTS avec les contraintes modifiées ____ **107**
Figure 49 : (a) Les métriques des exemples analysés (b) L'exemple 1 et son espace des métriques (c) L'exemple 2 et son espace des métriques (d) L'exemple 3 et son espace des métriques **110**
Figure 50 : (a) Contraintes et données à gérer pour une exploration classique (b) Contraintes et données à gérer pour une exploration de deux PACM ____ **111**
Figure 51 : Les espaces des métriques des différentes solutions ____ **112**
Figure 52 : Comparaison de l'exploration manuelle avec l'algorithme génétique ____ **113**
Figure 53 : (a) Contraintes et données à gérer pour une exploration classique (b) Contraintes et données à gérer pour une exploration de trois PACM ____ **114**
Figure 54 : Les espaces des métriques des différentes solutions ____ **115**
Figure 55 : Schéma de spécification de l'émetteur UMTS ____ **116**
Figure 56 : L'espace des métriques de la solution trouvée ____ **116**
Figure 57 : Schéma de spécification du décodeur AC3 ____ **118**
Figure 58 : Expérimentations pour l'application AC3 ____ **121**
Figure 59 : Architectures générées par CODEF ____ **122**
Figure 60 : Partition en deux PACM de l'application AC3 ____ **123**
Figure 61 : Architectures générées par CODEF précédé par GAMA² ____ **125**
Figure 62 : Les architectures générées par les deux approches ____ **126**
Figure 63 : Schéma fonctionnel de l'application Icam ____ **128**
Figure 64 : Graphe de spécification de l'application ICAM4f ____ **130**
Figure 65 : Solutions pour la partition en deux PACM ____ **131**
Figure 66 : Exemple de la répartition des données et de la mémoire en deux PACM ____ **132**
Figure 67 : Solution pour une partition en trois PACM ____ **132**
Figure 68 : Exemple de la répartition des données et de la mémoire en 3 PACM ____ **133**
Figure 69 : Solution pour une partition en quatre PACM ____ **133**
Figure 70 : Exemple de la répartition des données et de la mémoire en 4 PACM ____ **134**
Figure 71 : Exploration en 7 PACM de ICAM8f ____ **135**

LISTE DES TABLEAUX

Tableau 1 : Les paramètres influant sur les performances 25
Tableau 2 : Résumé des approches d'exploration d'architectures 38
Tableau 3 : Les paramètres influant sur les performances 63
Tableau 4 : Analogie architecture application..... 70
Tableau 5 : Résumé de spécification de l'émetteur UMTS..... 101
Tableau 6 : Métriques contraintes de l'émetteur UMTS..... 106
Tableau 7 : Les solutions de la pré-exploration de l'émetteur UMTS en 2 PACM..... 112
Tableau 8 : Les solutions de la pré-exploration de l'émetteur UMTS en 3 PACM..... 114
Tableau 9 : Possibilités d'exécution des tâches 121
Tableau 10 : Possibilités d'exécution des tâches après analyse par GAMA^2 124

I. INTRODUCTION

I. INTRODUCTION 8

1. CONTEXTE DE LA THESE 9
2. MOTIVATIONS 10
3. OBJECTIFS ET CONTRIBUTION 13
4. PLAN DE LA THESE 14

1. CONTEXTE DE LA THESE

Depuis plusieurs années, nous assistons à une évolution croissante et importante dans le domaine de la conception des systèmes sur puces (System on Chip : SoC). En effet, suite à la présence de plus en plus fréquente des SoC dans la vie de tous les jours (téléviseur, appareil photo, etc.) et dans des domaines exigeants (militaires, spatial, etc.), la conception de ces systèmes a focalisé l'intérêt d'un grand nombre d'industriels et de chercheurs. Ainsi ces systèmes sont maintenant composés d'un nombre important de composants logiciels et matériels communicant via de nombreux supports de communication. A titre d'exemple, un système automobile qui dure de nombreuses années peut comporter jusqu'à 1000 ECU (ECU, Electronic control unit), chacune d'entre elles devant être disponible pour le développement du logiciel dans le cas où un problème de compatibilité avec un matériel traditionnel surviendrait, ou pour la mise à jour avec de nouvelles technologies lorsque de nouveaux modèles de véhicules sont introduits.

L'intérêt croissant pour les systèmes numériques dans le monde industriel a permis aux applications numériques de se répandre partout dans notre vie quotidienne. En effet, les téléphones portables, les Ipods, les lecteurs MP3 et DVD, et d'autres systèmes numériques sont devenus des éléments courants de notre quotidien. Par exemple, 660 millions de téléphones mobiles ont été vendus en 2004 dont 80 millions avec appareil photo intégré [Diguet 2005]. Cette évolution a suscité l'intérêt de beaucoup d'industriels partout dans le monde et le processus de conception de ces applications a subi une grande pression au niveau du temps de mise sur le marché et du coût de fabrication.

En effet, le coût des parties numériques représente une part importante dans le coût global du système (1/3 du prix d'une automobile en 2006 [Diguet 2005]). Le coût de conception en temps et en argent d'un produit vendu dans le commerce est l'accumulation des coûts des différentes étapes nécessaires pour le développement du produit, depuis le cahier des charges jusqu'à la production massive et la mise en la vente dans les commerces.

i. Coûts de définition du problème et vérification système : En analysant le cahier des charges, il s'agit de définir l'application qui répond à l'exigence du cahier des charges. Ensuite, il s'agit de vérifier manuellement ou à l'aide d'outils de vérification si l'application définie répond aux besoins décrits dans le cahier des charges.

ii. Coûts de l'exploration de l'architecture : Une fois l'application définie, il s'agit ensuite d'analyser l'application et ses contraintes afin de trouver l'architecture la mieux adaptée à l'application. Cette exploration consiste à trouver une des meilleures partitions logicielles/matérielles et architectures de communication en considérant un certain nombre de contraintes à respecter et de performances à atteindre. Nous pouvons considérer la contrainte temps réel, le coût de réalisation, la consommation, etc.

iii. Coûts de développement/d'achat des codes et implémentation des tâches : Une fois l'architecture définie, il s'agit de l'implémenter. Donc, il faut coder ou acheter l'implémentation de chaque tâche sur le support d'exécution qui lui est attribué après l'exploration, implémenter l'architecture de communication et ajouter toutes les parties de gestion et d'ordonnancement de l'application (noyau temps réel, tâches de synchronisation et de communication).

iv. Coûts de simulation/émulation : Une fois l'architecture mise en œuvre, le fonctionnement est validé par simulation, émulation ou les deux (généralement les deux). Ensuite, les dernières finitions sont mises en œuvre par le/les concepteur(s) afin de préparer l'architecture à la production.

v. Coûts de production massive et de test : Une fois l'architecture prête, la production en masse commence. Ce genre de production est généralement suivi par des tests des circuits électroniques. En effet, lors de la production massive des circuits électroniques, quelques circuits sont produits avec des défaillances. Les tests permettent de détecter la quasi-totalité de ces circuits défaillants et de les écarter du marché.

La concurrence entre les industriels fait que chaque produit a un coût de conception global maximal au delà duquel le produit n'est plus commercialisable. Ainsi chacune des étapes de conception a un coût maximal à ne pas dépasser. Ce coût correspond soit au temps que les concepteurs passent à élaborer chaque étape, soit aux achats de spécifications ou d'informations nécessaires, soit aux achats de composants, frais d'implémentation et autres frais liés à la conception du produit.

2. MOTIVATIONS

L'étape d'exploration architecturale est une étape critique dans le flot de mise en œuvre du produit. En effet, pour une application donnée, il est possible d'avoir une multitude d'architectures délimitant un espace d'architectures. Cet espace contient des architectures non

réalisables, d'autres qui ne satisfont pas les contraintes, d'autres qui les satisfont et des architectures qui ont des performances optimales. Il incombe à l'étape d'exploration de trouver pour l'application une architecture adéquate qui satisfait les contraintes et qui optimise au mieux ses performances. Une élaboration manuelle rapide de l'architecture, qui consisterait à rassembler les différents éléments de l'application sur les supports d'exécution mis à disposition du concepteur, correspond à une exploration de l'architecture à un petit coût. Toutefois, en général, l'architecture trouvée ne répond pas aux contraintes.

En général, plus le concepteur investit en coût d'exploration, plus les performances de l'architecture trouvée sont élevées. Nous résumons l'évolution des performances de l'architecture en fonction du coût de l'exploration dans la Figure 1. Une exploration rapide manuelle conduit à une architecture qui ne satisfait pas les contraintes (croix rouge 1). Un minimum de raffinement, de récolte d'informations et d'automatisation de l'exploration est nécessaire pour trouver une architecture avec des performances qui satisfont les contraintes (croix rouge 2). Ce coût minimal dépend de la complexité de l'application (nombre de tâches, complexité des échanges des données, etc.), de la complexité du modèle d'architecture à réaliser (nombre de processeurs, architectures de communication, etc.). Plus l'investissement est élevé, plus la qualité de l'architecture est accrue jusqu'à avoir une architecture avec des performances très proches des performances optimales (croix rouge 3). A ce niveau, raffiner encore plus l'exploration coûte cher pour un gain en performances très restreint (croix rouge 4).

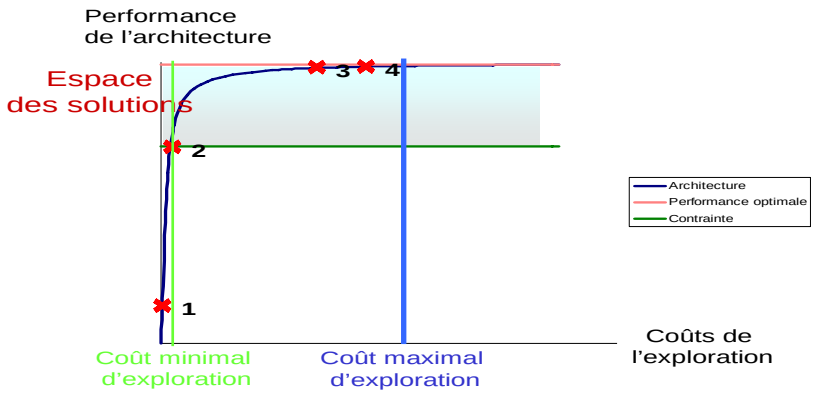


Figure 1 : Courbe d'exploration d'architectures pour une application

Nous estimons la courbe des performances des architectures en fonction du coût de l'exploration à une exponentielle dont l'équation est : $PO \times \exp\left(\frac{-\sigma}{coût}\right)$. PO est la performance optimale pour une application. Plus l'application est complexe, plus σ est grand. Plus le modèle d'architecture est complexe, plus σ est grand. σ varie aussi d'une méthode

d'exploration à une autre. Nous considérons que la courbe représente la meilleure méthode d'exploration. Au fil du temps, les applications sont de plus en plus compliquées et les possibilités technologiques sont plus importantes (nombre de processeurs de plus en plus élevé, une variété de ressources logicielles de plus en plus riche). Par conséquent, le σ n'a pas cessé d'augmenter. Le coût d'exploration devient de plus en plus élevé.

Il y a une dizaine d'années, les applications étaient composées de quelques tâches et étaient implémentées sur des architectures monoprocesseurs. L'architecture qui satisfaisait les contraintes était trouvée avec un coût d'exploration beaucoup plus petit que le coût d'exploration maximale (Figure 2.a). Les ambitions étaient d'atteindre une architecture avec des performances optimales. Les applications ont évolué au fil des années. Elles contiennent de plus en plus de tâches. De même, les architectures sont constituées d'un nombre croissant de composants logiciels, matériels et de communication. Par conséquent, l'exploration devient plus coûteuse et le **σ** de la courbe des performances en fonction du coût de l'exploration est de plus en plus élevé (Figure 2.a). Le coût minimum d'exploration permettant de trouver une architecture avec des performances qui satisfont les contraintes est de plus en plus élevé. En effet, le coût minimum d'exploration qui permet de trouver une architecture qui satisfait les contraintes est $cout\_\min = \frac{\sigma}{Ln\frac{Po}{Pc}}$. **Po** est la performance optimale. **Pc** est la performance contrainte. Le coût minimum d'exploration **cout_min** est proportionnel à **σ**. Plus **σ** augmente, plus le coût minimum augmente. Pour les applications actuelles, le coût minimum est de plus en plus proche du coût maximal d'exploration (Figure 2.a). La question qui se pose est de savoir si dans quelques années, il sera encore possible de trouver une architecture qui satisfera les contraintes sans dépasser le coût maximal d'exploration. Etant donné que dans quelques années, les architectures seront composées d'une dizaine de processeurs différents ou plus et que les applications comporteront plus d'une centaine de tâches, il est sûr qu'un jour les méthodes d'exploration actuelles dépasseront le coût maximal d'exploration.

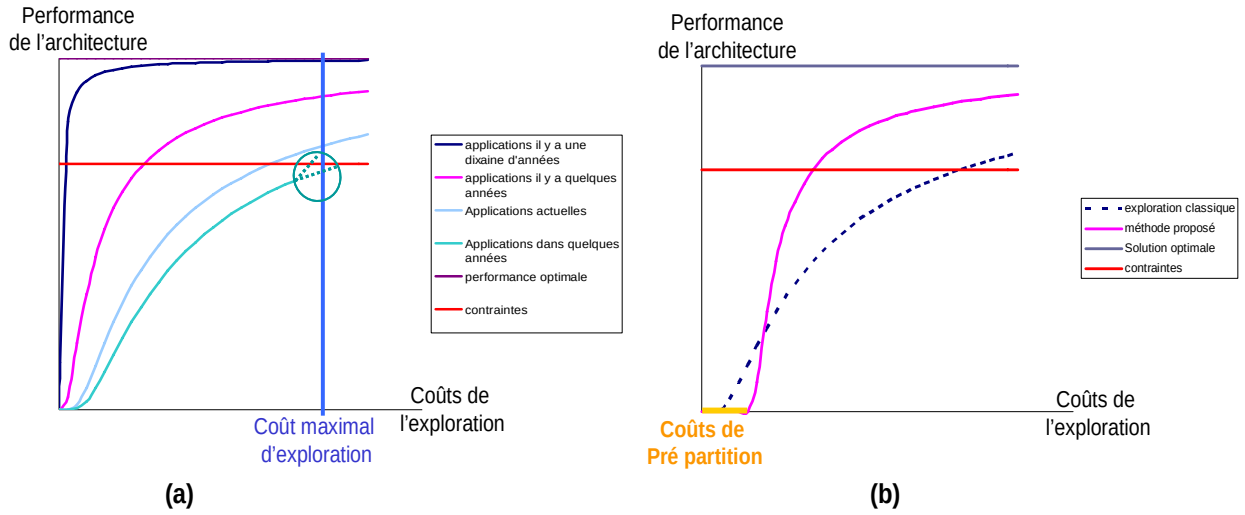


Figure 2 : (a) Evolution de la courbe performance de l'architecture trouvée en fonction du coût d'exploration (b) Méthode pour augmenter le σ de l'exploration

3. OBJECTIFS ET CONTRIBUTION

Il est donc indispensable de trouver un moyen pour diminuer les coûts d'exploration afin de pouvoir trouver une architecture adéquate aux applications à venir. Plusieurs moyens d'y arriver sont possibles. Certains sont financiers et d'autres sont techniques. Il nous est impossible d'intervenir sur les moyens financiers. Nous allons donc nous intéresser aux moyens techniques :

- ✓ Solution financière, la délocalisation : "Changement d'implantation géographique de tout ou partie des activités d'une entreprise, notamment pour réduire les coûts de production" [Grand dictionnaire]. Ceci permettra d'avoir une exploration de moindre coût. Par conséquent, la marge entre le coût d'exploration pour avoir une architecture qui satisfasse les contraintes et le coût maximum d'exploration est plus large.
- ✓ Evolution de la méthodologie d'exploration : Il faut avoir des méthodologies qui permettent d'avoir un σ plus élevé. D'un côté, σ dépend de la complexité des applications. La complexité n'arrêtera pas de croître au fil du temps. D'un autre côté, σ dépend de la complexité des architectures adéquates pour les applications. Plus le nombre de processeurs et de ressources de communication est grand, plus σ est grand. Donc, si le nombre d'architectures possibles pour l'application diminue, σ diminue aussi. Par conséquent, la solution est d'avoir une étape avant l'exploration, qui a un coût

faible et qui permet de réduire le nombre des architectures possibles pour l'application afin de diminuer σ de la courbe de l'exploration (Figure 2.b).

Nous proposons, dans cette thèse, une approche de conception qui commence l'exploration à un haut niveau afin de réduire l'espace d'architectures pour l'exploration traditionnelle et réduire en conséquence les coûts de l'estimation des performances et de l'exploration. La réduction de l'espace d'architectures n'est pas fait au détriment de la qualité de l'exploration. En effet, l'analyse préliminaire tient compte de suffisamment de paramètres pour permettre d'éliminer les architectures les moins performantes et guider l'exploration vers un espace réduit contenant la plupart des architectures les plus performantes. Cette approche considère à haut niveau des nouveaux paramètres des architectures multiprocesseurs modernes influant sur les performances. Ces paramètres sont la diversité logicielle, un parallélisme accentué et une notion de proximité entre les composants. Ils sont détaillés et discutés dans le troisième chapitre.

4. PLAN DE LA THESE

Le chapitre II présente une étude de l'exploration des architectures et résume les travaux existants. En premier lieu, nous détaillons les différentes parties de l'exploration de l'architecture qui sont la communication, les mémoires, l'ordonnancement et l'affectation du support d'exécution de chaque tâche. Ensuite, nous étudions l'influence des différents éléments de l'architecture qui influent sur les performances du système (temps, surface, consommation). Enfin, nous présentons les différentes méthodologies d'exploration d'architectures.

Dans le chapitre III, nous présentons les difficultés actuelles et futures de l'exploration des architectures. Nous présentons notre approche pour faire face à ces nouvelles difficultés. Elle consiste à faire précéder les méthodes d'exploration traditionnelles par une étape de pré-exploration. Cette dernière consiste à analyser à haut niveau l'espace d'architectures afin de réduire la complexité et le coût de l'exploration. Cette analyse se base sur l'analyse de plusieurs critères de l'application : les communications, les mémoires, le parallélisme et les performances d'exécution des tâches.

Le chapitre IV détaille l'analyse de haut niveau que nous proposons. Nous définissons des métriques comme modèle d'analyse. Ces métriques forment un espace ou l'analyse doit chercher la solution qui couvre le plus cet espace. Pour cela, nous détaillons l'algorithme génétique multi-objectif utilisé et décrivons l'outil GAMA^2 mis en œuvre.

Trois applications sont étudiées dans le chapitre V afin de valider la méthodologie que nous proposons ainsi que les différents modèles que nous proposons (le graphe de spécification et les métriques).

Enfin, nous concluons le manuscrit dans le chapitre VI et nous présentons les perspectives du travail présenté.

II. ETAT DE L'ART

1. INTRODUCTION 17
2. APPLICATIONS ET ARCHITECTURE MULTIPROCESSEUR 18
2.1. EVOLUTION DES ARCHITECTURES SELON INTEL 19
2.2. MULTIPROCESSEUR CELL DE IBM 20
2.3. MULTIPROCESSEUR CT3400 DE CRADLE 20
2.4. LES ARCHITECTURES DU FUTUR 21
3. PERFORMANCES DES ARCHITECTURES 21
3.1. TEMPS D'EXECUTION 21
3.2. CONSOMMATION 23
3.3. SURFACE 24
3.4. RESUME 25
4. COMPLEXITE DE L'EXPLORATION DES ARCHITECTURES 25
5. QUELQUES APPROCHES D'EXPLORATIONS 27
5.1. EXPLORATION MANUELLE 27
5.2. EXPLORATION DIRECTE D'ARCHITECTURE 28
5.3. DIVISION DE L'EXPLORATION EN PLUSIEURS PARTIES 30
5.4. PLATEFORMES 35
5.5. ANALYSE DE HAUT NIVEAU 36
5.6. RESUME 37
6. CONCLUSION 39

1. INTRODUCTION

Nous nous intéressons dans ce travail à la conception des architectures multiprocesseurs monopuces. Ceci consiste à choisir, définir ou construire une architecture adéquate à l'application traitée. Afin de cerner les difficultés et les défis qu'il faut surpasser, en premier lieu, nous situons cette étape par rapport à la chaîne de conception d'un système embarqué (Figure 3). Pour qu'une application passe de son état cahier des charges (a) à son état circuit (d), elle passe par trois phases principales (Figure 3) :

- ✓ Mise en œuvre : pour répondre aux besoins du cahier des charges (a), le concepteur étudie les besoins du marché et les besoins de l'application pour définir une spécification adéquate. Par exemple, Motorola [Motorola] classe le téléphone mobile à concevoir parmi trois classes (bon marché, classe moyenne et le top de la technologie) pour définir la spécification du téléphone.
- ✓ Exploration de l'architecture : l'architecture en plus de répondre aux besoins de la spécification, doit satisfaire aussi des contraintes technologiques (temps réel, autonomie d'énergie et limite de surface). L'exploration consiste à parcourir l'espace d'architectures (l'espace qui regroupe l'ensemble des architectures possibles pour l'application) et à choisir celle qui satisfait les contraintes et qui optimise au mieux les performances.
- ✓ L'implémentation : la phase d'intégration est le passage d'une description haut niveau de l'architecture et de ses composants (niveau fonctionnel, comportemental, etc.) à une description de bas niveau (niveau transfert de registre, porte, etc.).

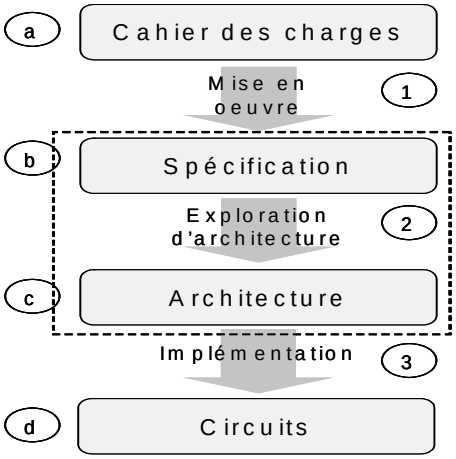


Figure 3 : Chaîne de conception d'un circuit embarqué

L'exploration de l'architecture est une étape clé de la chaîne de conception d'un circuit. Elle est le maillon de la chaîne qui permet le passage du concept (la spécification fonctionnelle) à la technologie (spécification architecturale). En effet, au niveau architecture les contraintes et les performances technologiques seront fixées et ne bougeront pas ou peu lors de l'implémentation en circuit. Donc, l'exploration est l'étape responsable de l'optimisation des performances de l'application.

Ci-après, nous présentons quelques exemples illustrant l'évolution des applications et des architectures. Ensuite, nous étudions le modèle d'évaluation des architectures. Enfin, nous étudions les approches d'exploration d'architectures.

2. APPLICATIONS ET ARCHITECTURE MULTIPROCESSEUR

Les applications sur le marché ne cessent d'évoluer en augmentant leur complexité. Nous pouvons classer l'évolution de ces architectures en deux catégories :

Les systèmes qui contiennent une seule application et qui demandent de plus en plus de performances. Par exemple, pour la console de jeux PS3 (Play Station 3), les concepteurs ont jugé nécessaire d'augmenter les performances du système de 100 fois par rapport à ceux de la PS2 (Play Station 2) [Khale 2005]. Ce type d'application demande un parallélisme pour alléger des calculs lourds. Les architectures monoprocesseurs comme celles des PC (Personal Computer) sont devenues insuffisantes pour de telles applications.

Les systèmes qui contiennent de plus en plus d'applications : les téléphones portables, les Ipods sont des exemples assez populaires pour leurs contenances de plusieurs applications. Par exemple, les PDA [Apple 2007a], [Qtek 2007] contiennent maintenant un lecteur vidéo, un lecteur de musique et un diaporama d'images, en plus d'un ensemble de logiciels qui constituent différents utilitaires comme l'agenda, le carnet d'adresse, les jeux, etc. Malgré toutes ces fonctionnalités, ils sont de plus en plus légers (entre 150 et 200 grammes), ils ont une capacité de stockage importante (jusqu'à 80 Go) et ils assurent une bonne autonomie (jusqu'à 20 heures de musique et 6,5 heures de vidéo). Les téléphones portables [Iphone 2007], [N95 2006] aussi contiennent de plus en plus d'applications : des jeux, des utilitaires (alarme, mémo, agenda, etc.), des appareils photo haute définition (jusqu'à 5 Méga pixels pour le N95), un GPS, des lecteurs/enregistreurs audio, des caméras et même des accès Internet à des débits assez élevés. Le nombre important de tâches dans un seul système demande de plus en plus d'architectures assurant un parallélisme important.

Les applications comportant de plus en plus de tâches, ou des tâches dont le calcul est assez lourd, nécessitent des architectures de plus en plus performantes pour leurs calculs. Les architectures monoprocesseurs traditionnelles, bien qu'elles aient apporté une évolution importante dans le monde du numérique, ont de plus en plus de difficultés à assurer les performances nécessaires pour les applications du présent et du futur. Dans le futur, cette évolution ne s'arrêtera pas. Les systèmes comporteront de plus en plus d'applications, comporteront plus de tâches et susciteront un niveau supérieur de parallélisme.

Les architectures ont elles aussi évolué pour suivre l'évolution des applications et répondre à leurs besoins en termes de performances. Nous décrivons ci-après quelques architectures pour illustrer l'évolution des architectures.

2.1. EVOLUTION DES ARCHITECTURES SELON INTEL

Intel, [Borkar 2007] établit une analyse de l'évolution des processeurs. L'augmentation de fréquence, bien que permettant d'augmenter les performances des processeurs, cause une augmentation plus conséquente de la consommation (Figure 4.a). Les architectures multiprocesseurs quant à elles présentent une solution pour une importante réduction de la consommation [Marcuello 1998]. Par exemple, pour les processeurs bi-cœur d'Intel (Figure 4.b), une baisse de fréquence de 15% par rapport aux processeurs mono-cœur a permis une nette augmentation des performances sans augmenter la consommation.

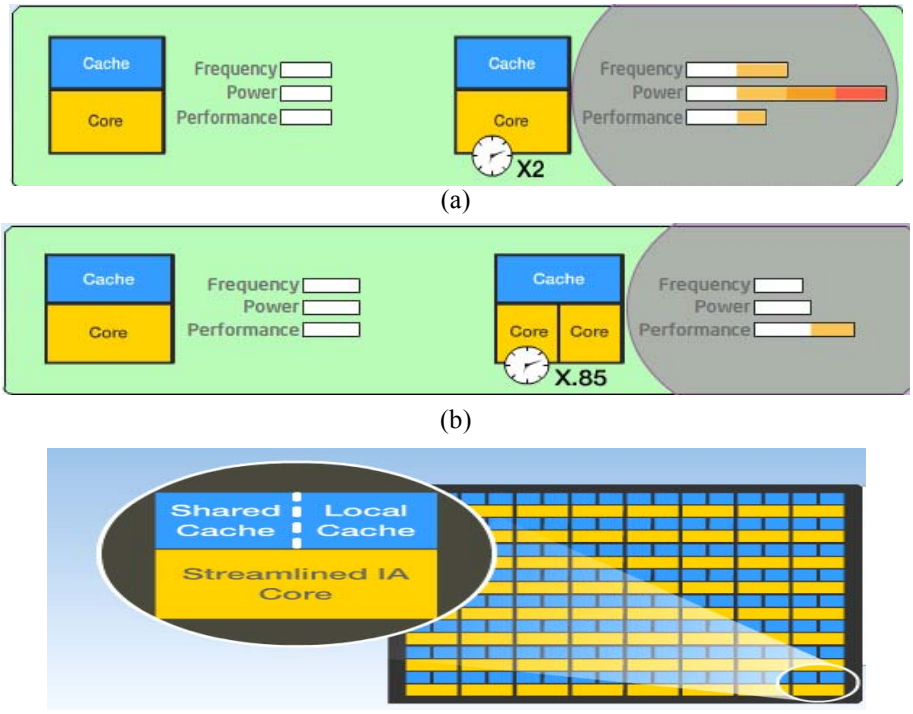


Figure 4 : Evolution des processeurs selon INTEL

Cette analyse laisse penser aux chercheurs d'Intel que le futur est aux systèmes multiprocesseurs composés de plusieurs cœurs munis chacun d'un cache local et d'un cache partagé (Figure 4.c).

2.2. MULTIPROCESSEUR CELL DE IBM

Au début de l'année 2001, le centre de conception STI, qui est le fruit de la collaboration de trois géants des système numériques **S**CEI (Sony Computer Entertainment Incorporated), **T**oshiba et **I**BM, a été créé pour l'élaboration d'une architecture cent fois plus performante que celle de la PS2 (Play Station 2) [Kunimatsu 2000]. Cette architecture (Figure 5) nommée Cell [Khale 2005] est utilisée pour la PS3. Il s'agit d'une architecture combinant un processeur "*Power architecture*" [IBM] 64 bits et plusieurs SPE (Synergitic Processor Element). Un SPE correspond à un processeur vectoriel indépendant possédant 128 registres 128 bits, 4 unités de calcul en virgule flottante double précision et 4 unités de calcul entier. Ce processeur effectue deux instructions par cycle d'horloge. Il inclut une mémoire locale de 256 Ko de type SRAM haute vitesse. Sony et Toshiba ont, d'ailleurs, mis au point un prototype de station de travail incluant plusieurs processeurs Cell et caractérisé par une puissance de calcul de 16 Tflops. Le processeur Cell mettra en œuvre un bus interne à 6,4 GHz. Il sera fabriqué en technologie Cmos SOI 90 nm dans un premier temps.

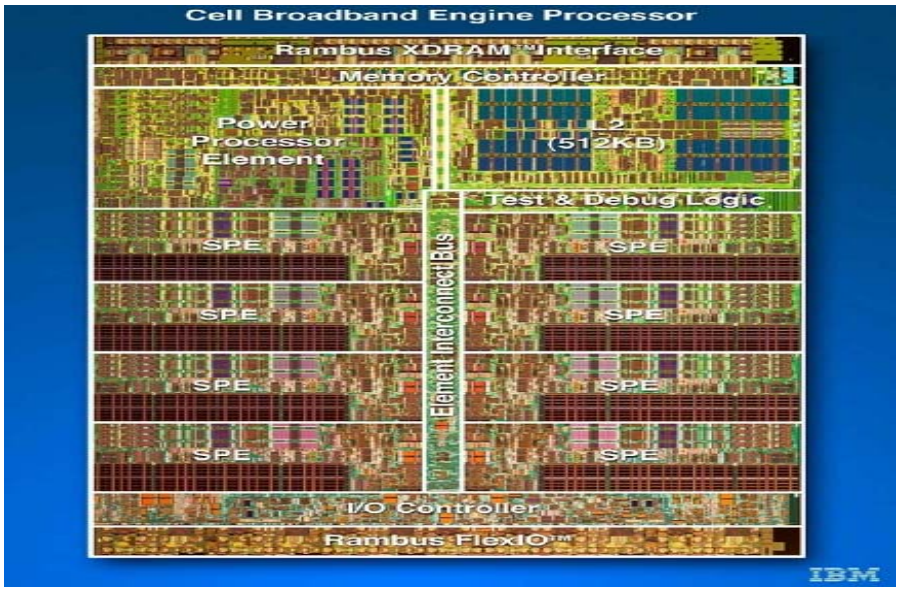


Figure 5 : Architecture de processeur Cell

2.3. MULTIPROCESSEUR CT3400 DE CRADLE

Le CT3400 [cradle 2007] est un système multiprocesseur dédié pour la vidéo, l'image et le calcul intensif. Il est constitué de 8 DSP et 4 processeurs RISC. Les DSP sont de type

SIMD [Flynn 1966] (*Singel Instruction Multiple Data*). Ils intègrent un ensemble d'instructions supportant les calculs intensifs et les opérations multidimensionnelles utilisées pour le son et la vidéo. Les huit DSP constituent un bloc de calcul noté '*DSP engine*'. Ce bloc est capable d'effectuer 2,9 billions de MAC (*Multiply-ACccumulte*) par seconde pour des données de taille 8 bits et une fréquence de 230 Mhz. En même temps, le bloc de processeur ('*RISC engine*') effectue les opérations de contrôle, des calculs supplémentaires et de contrôle de l'exécution des tâches sur les DSP.

2.4. LES ARCHITECTURES DU FUTUR

Les systèmes multiprocesseurs s'imposent de plus en plus comme des architectures plus performantes. Les industriels proposent de plus en plus des architectures et des plateformes multiprocesseurs (Cell, CT3400, etc.). Dans le futur, comme pour les architectures monoprocesseurs, les utilisateurs concevront leur propre architecture multiprocesseur spécifique à chaque application. Les architectures du futur seront composées d'un nombre assez élevé de processeurs, d'une multitude d'accélérateurs matériels, de mémoires et de plusieurs ressources de communication. Dans nos travaux, nous nous intéressons à la conception de ces architectures multiprocesseurs hétérogènes spécifiques à une application.

3. PERFORMANCES DES ARCHITECTURES

Les architectures sont évaluées suivant trois critères de performance : le temps, la surface et la consommation. Ces performances dépendent des performances de chacune des parties de l'architecture décrites précédemment. Nous étudierons ci-après chacune des trois critères de performance.

3.1. TEMPS D'EXECUTION

Pour la vidéo, un débit d'image minimum est indispensable pour avoir une visualisation correcte de la vidéo. Pour l'automobile, il est indispensable que l'électronique réagisse dans un délai minimum (par exemple, arrêter l'action du régulateur automatique de vitesse pour ne pas mettre en danger le conducteur). Quelques travaux se sont intéressés à l'estimation du temps d'exécution [Calvez 1996], [Ekerling 1996], [Gajski 1998], [Suzuki 1996] et [Ye 1995]. L'estimation consiste à évaluer le temps d'exécution des tâches soit par émulation sur carte soit en analysant à haut niveau la structure de la tâche. Afin de mieux appréhender

l'impact des décisions architecturales sur ce critère, nous nous intéressons à l'étude de l'influence des composants de l'architecture sur son temps d'exécution.

La performance temporelle traduit le temps nécessaire à l'application pour une exécution complète. Ce temps est une fonction qui dépend du temps d'exécution des tâches, des communications, des différentes latences et du parallélisme. La Figure 6 représente un exemple du temps d'exécution d'une application. Le temps de l'application **T** tient compte du temps d'exécution des tâches, des communications et du parallélisme de l'exécution des tâches.

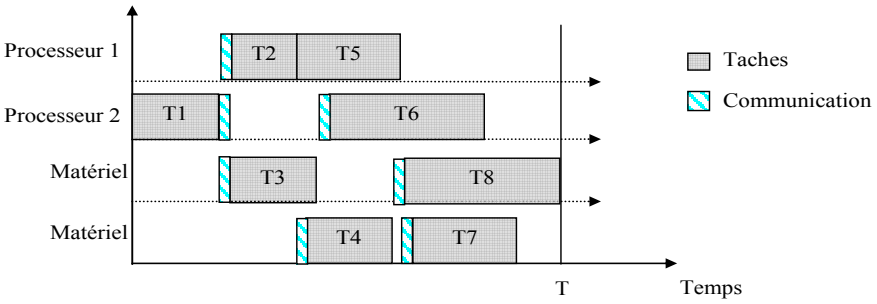


Figure 6 : Temps d'exécution

Le temps d'exécution de l'ensemble de l'architecture dépend des facteurs suivants :

- ✓ Temps d'exécution des tâches : plus le temps d'exécution des tâches est court, plus le temps d'exécution de l'ensemble de l'application est court. Le temps d'exécution d'une tâche de l'application dépend de son support d'exécution.
- ✓ Parallélisme : l'exécution de deux tâches en parallèle dure le temps de la plus longue exécution des deux tâches alors que l'exécution successive des deux tâches dure la somme des temps d'exécution des deux tâches. Par conséquent le parallélisme optimise le temps d'exécution total de l'application.
- ✓ Disponibilité des mémoires : lorsque deux tâches sont exécutées en parallèle et ont besoin d'accéder à la mémoire simultanément, il faut prévoir une ressource de mémoires disponible pour chacune ou une mémoire multi-port. Dans le cas contraire, des temps de latence et d'attente d'accès à la mémoire s'ajoutent au temps total d'exécution.
- ✓ Communications : la quantité des données échangées dans l'application est de plus en plus importante. Ainsi, le temps nécessaire à l'échange des données représente une part de plus en plus conséquente du temps total d'exécution de l'application. Aussi, le parallélisme croissant impose un nombre important d'échanges simultanés des données. En conclusion, l'architecture de communication se doit de fournir les ressources

nécessaires pour assurer des échanges des données de plus en plus importants et simultanés.

L'amélioration de l'une de ces caractéristiques peut parfois engendrer la détérioration d'une ou de plusieurs des autres caractéristiques. Par exemple, exécuter deux tâches sur le même processeur qui les optimisent améliore la communication, mais détériore le parallélisme. Ainsi, optimiser le temps d'exécution de l'application consiste à trouver le compromis qui améliore les quatre caractéristiques en même temps.

3.2. CONSOMMATION

La consommation est un des paramètres importants lors de la conception des systèmes actuels. En effet, la portabilité des appareils électroniques impose que ces appareils aient une autonomie aussi élevée que possible. Par exemple, s'il faut recharger les appareils toutes les deux minutes, la portabilité aura moins d'utilité.

[Wenjie 2002], [Chandrakasan 1995] définissent 3 sources de consommation pour les circuits numériques :

- ✓ Puissance dynamique (***P*** $_{dynamique}$) : la puissance consommée par la partie en activité du circuit.
- ✓ Courant de court circuit (***I*** $_{court\_circuit}$) : le courant qui se crée lors de la transition de deux transistors NMOS et PMOS en même temps.
- ✓ Courant de fuite (***I*** $_{fuitet}$) : un courant parasite qui se crée entre les transistors et le substrat. Ce courant, négligé il y a quelques temps, devient de plus en plus important avec l'évolution de la technologie (90 nanomètres et moins).

En résumé, la puissance est la somme de la puissance dynamique et la puissance statique (équation 1). La puissance statique est due au courant de court-circuit et au courant de fuite.

$$P = P_{dynamique} + (I_{court-circuit} + I_{fuite}) \times V \qquad \text{(équation 1)}$$

La puissance dynamique [Curd 2007] est proportionnelle à la fréquence **f**, à la capacité dynamique **C** (*capacitance of node swtching*) et au carré de la tension d'alimentation **V** (équation 2)

$$P_{switching} = C \times f \times \mathrm{V}^2 \qquad \textbf{(équation 2)}$$

La capacité C est dépendante du taux d'activité du circuit [Landman 1995]. En d'autres termes, s'il y a une tâche en activité pendant l'exécution de l'application, la puissance

dynamique du circuit à ce moment là est celle de la tâche en cours d'exécution. Autrement dit, d'une façon plus globale, nous pouvons assimiler la puissance du circuit à un instant **t** à la somme de la puissance statique du circuit plus la puissance dynamique des tâches en cours d'exécution. Ainsi pour optimiser la puissance du circuit, il faut attribuer à chaque tâche le support d'exécution qui minimise le plus sa puissance. A l'exécution des tâches s'ajoute aussi le transfert des données entre les tâches. Par conséquent, l'optimisation de la puissance des communications permet d'optimiser également la puissance totale du système. L'augmentation de la fréquence a pour conséquence d'augmenter la consommation (équation 2). Il faut donc s'assurer d'utiliser les fréquences les moins élevées pour les ressources d'exécution et les ressources de communication.

Le parallélisme dans une application augmente le taux d'activité et donc la capacité dynamique. Toutefois, comme le parallélisme accélère les calculs, nous pouvons diminuer la fréquence **f** et la tension **V,** ce qui laisse penser à une diminution de la puissance consommée par le système (équation 2). Les travaux de [Laurent 2007] présentent un exemple dans lequel le parallélisme ne favorise pas toujours la consommation. Dans cet exemple, le parallélisme a augmenté le nombre de composants, d'où une nette augmentation de la puissance statique. Ainsi, nous pouvons dire que si nous favorisons le parallélisme sans augmenter le nombre de composants, par ordonnancement des tâches de l'application ou en implantant une architecture favorisant le parallélisme, nous pouvons améliorer l'optimisation de la puissance. Toutefois, nous admettons que cette supposition dépend des applications et il se peut qu'elle ne soit pas valable pour toutes les applications.

Plusieurs travaux se sont intéressés à l'étude de la consommation des composants de l'application. Des travaux étudient les ressources d'exécution : les processeurs [Pakdeepaiboonpol 2006], les DSP [Ktari 05], [Laurent 2004] et les FPGA [Garcia 2005]. D'autres ciblent les mémoires : [Marteil 2004] et [Elleouet 2006]. Enfin, [Lahiri 2004] et [Caldari 2003] étudient les bus de communication.

3.3. SURFACE

La surface constitue elle aussi une contrainte importante. En effet, les applications contiennent de plus en plus de composants qu'il faut intégrer dans des systèmes de plus en plus miniaturisés. Aussi, la surface influe sur la consommation. Plus la surface est grande, plus la consommation statique est importante. La surface d'une application est la somme des surfaces des supports d'exécution des tâches, des mémoires et des supports de communication.

3.4. RESUME

En résumé (Tableau 1), nous avons défini dans cette étude les éléments de l'architecture influant sur les trois critères de performance de l'architecture que sont le temps, la surface et la consommation. Pour chacun des critères, nous avons extrait l'influence de l'exécution des tâches, des communications, de la mémoire et du parallélisme sur les performances.

Performances	Tâches	Communication	Mémoire	Parallélisme
Temps	Optimiser le temps d'exécution des tâches	Minimiser les latences	- Minimiser le temps d'accès - Minimiser le temps d'attente	Favoriser le parallélisme
Surface	Optimiser la surface des supports d'exécution	Optimiser la surface des composants de communication		
Consommation	- Optimiser la consommation des ressources d'exécution - Diminuer la fréquence des ressources d'exécution	- Optimiser la consommation des ressources de communication - Diminuer la fréquence des ressources de communication	- Optimiser la consommation des mémoires	

Tableau 1 : Les paramètres influant sur les performances

La définition d'une architecture pour une application particulière consiste à explorer l'espace des architectures possibles pour l'application afin de déterminer l'architecture qui optimise au mieux les trois performances : le temps, la surface et la consommation. Ci-après, nous faisons état de l'art des différentes méthodologies d'exploration de l'architecture.

4. COMPLEXITE DE L'EXPLORATION DES ARCHITECTURES

Le problème de l'exploration de l'espace d'architectures peut être vu comme un problème classique d'exploration combinatoire. Il consiste à analyser toutes les combinaisons possibles d'architecture logicielle/matérielle pour toutes les fonctions de l'application à concevoir. Il s'agit d'un problème N-P complet. Les premières solutions étaient soit orientées logiciel, soit orientées matériel. L'approche orientée logiciel [Ernst 1996], [Osterling 1997] commence à partir d'une spécification entièrement logicielle et cherche à faire de la migration de code vers le matériel. Ce type d'approche favorise la flexibilité et assure l'implémentation logicielle des tâches de l'application. L'approche orientée matériel [Gupta 1996] part d'une spécification initiale entièrement en matériel. Par la suite, les parties non critiques sont identifiées afin de les affecter à une réalisation logicielle, ce qui permet de réduire le coût de

réalisation. Ce type d'approche vise à avoir des performances temporelles bien optimisées grâce aux performances que peut fournir l'implémentation matérielle. Toutefois, face à l'évolution des applications et des architectures, ces approches se sont trouvées assez limitées et ne permettent pas de trouver l'architecture la plus adéquate. Donc, la recherche met en pratique une heuristique pour analyser les combinaisons de logiciels, de matériels, d'architectures de communication et d'architectures de mémoire afin de trouver celle qui s'adapte aux contraintes.

L'exploration de l'architecture consiste à définir pour une application spécifique les ressources d'exécution logicielles et matérielles qui exécuteront ces différentes tâches. Il s'agit aussi d'établir l'architecture mémoire nécessaire pour l'optimisation de l'exécution des tâches et des différents transferts entre elles. Enfin, il faut aussi mettre en œuvre une architecture de communication capable d'accomplir les différents transferts des données tout en respectant les contraintes de l'application. Ces différents composants de l'architecture sont définis de façon à améliorer les performances de l'application (temps, surface, consommation).

La complexité de l'exploration réside dans le choix d'un composant de communication ou d'une ressource d'exécution pouvant influer sur les performances des autres composants. Un exemple simple, nous exécutons une tâche 1 par un processeur 1 pour optimiser les performances. Ensuite, si nous exécutons une tâche 2 sur un processeur 2 pour optimiser les performances, les performances de la tâche 1 peuvent être détériorées. En effet, si la tâche 1 échange des données avec la tâche 2 au cours de son exécution, le temps des communications s'ajoute au temps d'exécution de la tâche. Par conséquent, les performances de la tâche 1 sont moins optimisées.

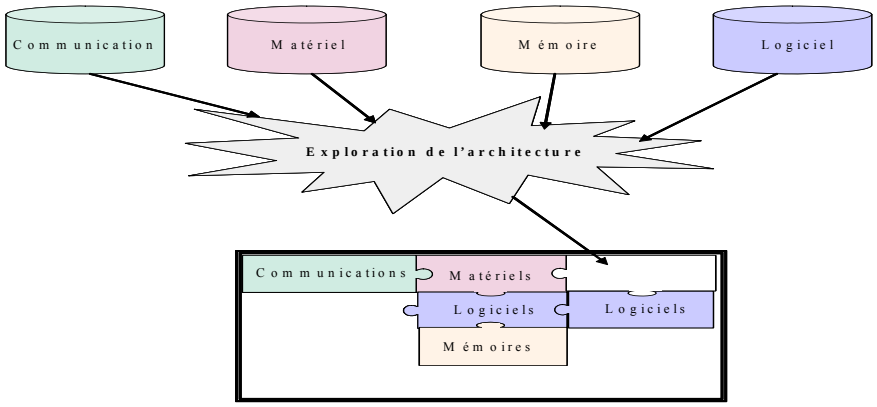


Figure 7 : La complexité de l'exploration

Pour résumer, nous pouvons considérer l'exploration de l'architecture comme la construction d'un puzzle (Figure 7). Les composants de ce puzzle sont dans les bibliothèques

de communication, de mémoires et de ressources d'exécution logicielles et matérielles. Le choix de chaque partie du puzzle influe sur la forme finale du puzzle.

Cette complexité a fait que la plupart des travaux utilisent des heuristiques pour explorer l'espace d'architectures [Barros 1994], [Jantsch 1994 [Dave 1998b], [Dick 1998], [Fei 2002], etc.

Ensuite, l'augmentation du nombre des tâches des applications, des ressources logicielles, des ressources de communication et du nombre de composants pouvant s'intégrer dans une même architecture ont augmenté considérablement la complexité de l'exploration. Nous présentons ci-après les principales approches existantes pour résoudre ce genre de problème.

5. QUELQUES APPROCHES D'EXPLORATION

Beaucoup de travaux se sont intéressés à l'exploration d'architectures. Nous étudions ci-après différentes approches d'exploration.

5.1. EXPLORATION MANUELLE

[Baghdadi 2002] présente une approche qui se base sur une estimation des performances à un niveau bas (RT : *Register Transfert*), des décisions architecturales manuelles et une simulation haut niveau pour estimer les performances de l'architecture (Figure 8). L'application est décrite en description haut niveau SDL (*System Description Langage*). L'approche d'exploration se compose de quatre étapes principales :

- ✓ Calcul des délais élémentaires : l'estimation des performances et des délais se fait au niveau RT en utilisant l'outil MUSIC [Jerraya 1999]. Ce dernier génère à partir de la description SDL des descriptions RT logicielles et matérielles des différentes tâches.
- ✓ Annotation : il s'agit d'instrumenter la spécification SDL avec les délais élémentaires obtenus dans l'étape précédente en utilisant les extensions de SDL introduites par ObjectGEODE.
- ✓ Choix architecturaux : cette étape permet d'introduire les choix des supports d'exécution des différentes tâches de l'application dans la description de l'application.
- ✓ Simulation : il s'agit d'exécuter la description SDL obtenue après les choix architecturaux par le simulateur *geodesim* de l'environnement ObjectGEODE.

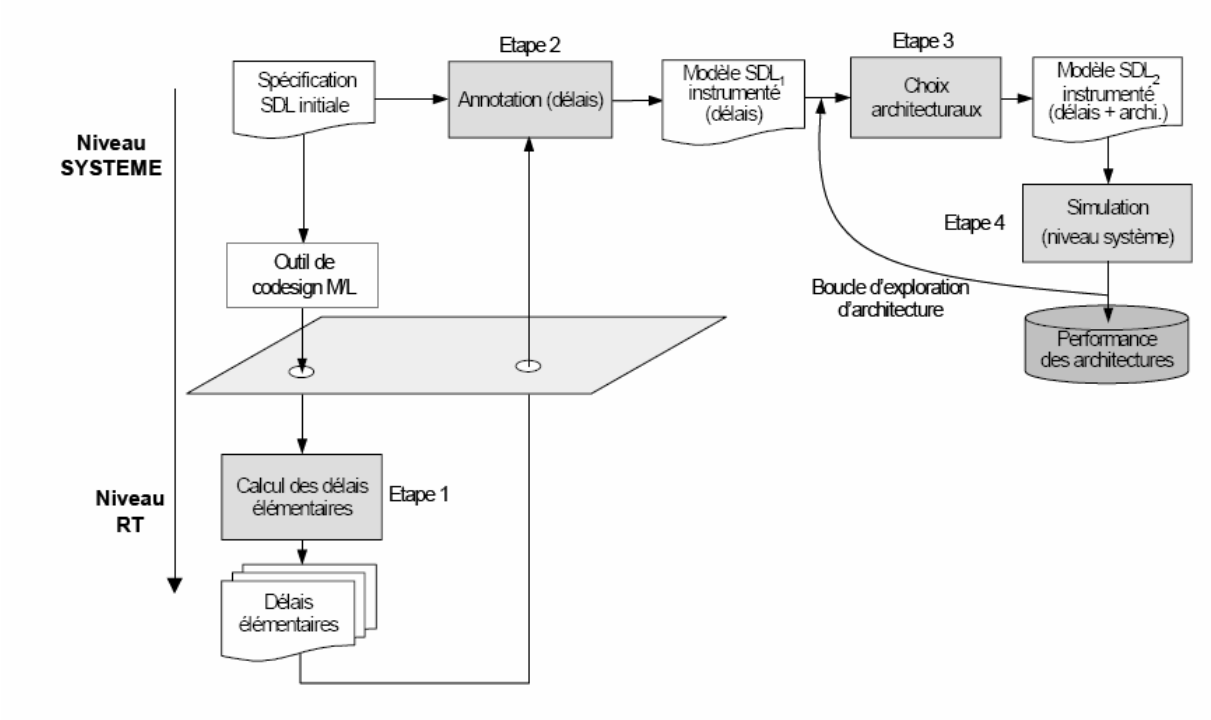


Figure 8 : Flot de l'exploration manuelle

L'étape de choix se fait manuellement en se basant sur l'expérience du concepteur. L'outil de simulation permet de donner rapidement des informations sur les performances de l'architecture choisie permettant de guider le concepteur lors de son exploration. Toutefois, face aux applications de plus en complexes, cette approche atteint ses limites. En effet, l'espace d'architectures étant de plus en plus important, l'exploration manuelle de cet espace sera lente avant de trouver une architecture adéquate.

5.2. EXPLORATION DIRECTE D'ARCHITECTURE

[Bianco 1999b] et [Auguin 2001] présentent un outil d'exploration d'architectures CODEF (Figure 9). L'algorithme d'exploration est divisé en deux parties :

- ✓ Une partie statique : elle comprend la lecture des bibliothèques, du graphe de flots de données, du système prédéfini (s'il existe), des contraintes temporelles et matérielles. Elle comprend également le calcul de valeurs permettant de quantifier les différentes possibilités de partitionnement, d'après une analyse des chemins du DFG.
- ✓ Une partie dynamique : elle prend en entrée toutes les données calculées par la partie statique. Dans, cette partie, l'algorithme ordonnance les tâches une à une. Les choix sont fait par l'algorithme en fonction d'un calcul de coût effectué pour chaque tâche et chacune de ses réalisations potentielles. Ces coûts sont déduits de l'urgence temporelle

de chaque tâche et des possibilités de partitionnement offertes par les contraintes matérielles et les bibliothèques.

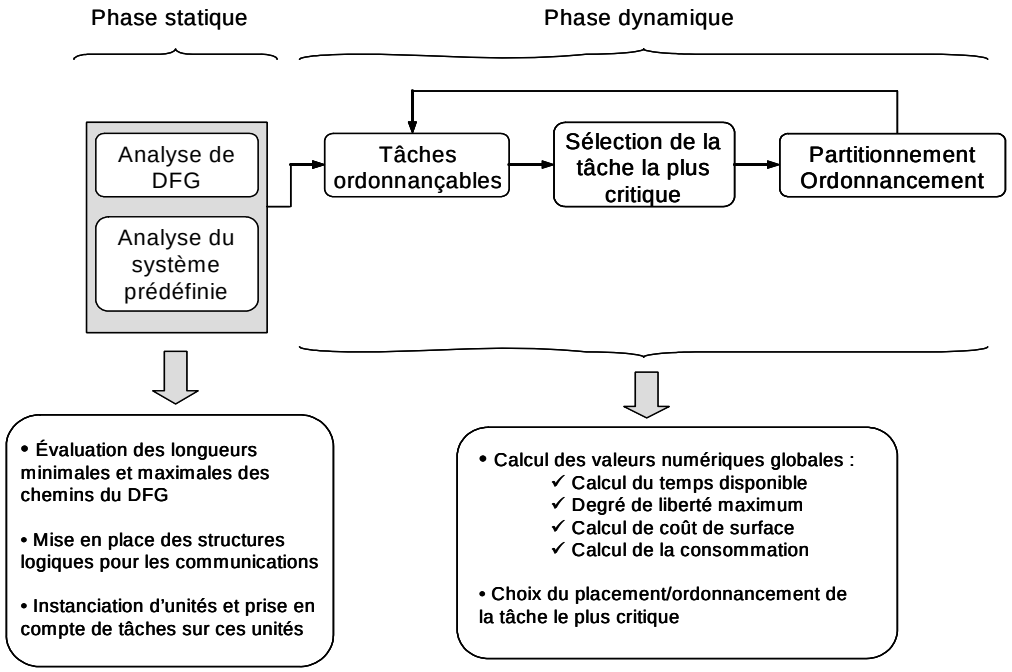


Figure 9 : Flot d'exploration CODEF

Cette méthode permet de trouver les architectures qui satisfont les contraintes temporelles et qui optimisent au mieux la surface de l'architecture. En entrée (Figure 10), le concepteur fournit les contraintes à satisfaire, le graphe de spécification de l'application (un DFG : *Data Flow Graph*) et une bibliothèque d'implémentations. Cette dernière contient toutes les implémentations possibles de chaque tâche et les performances de chaque implémentation. Le concepteur contrôle l'exploration de l'architecture. Par exemple, il peut intervenir pour le choix du nombre de processeurs, des supports de communication etc. En sortie, CODEF fournit un ensemble d'architectures satisfaisant les contraintes. Chaque architecture est fournie avec sa spécification, sa structure, son ordonnancement et un diagramme de Gantt.

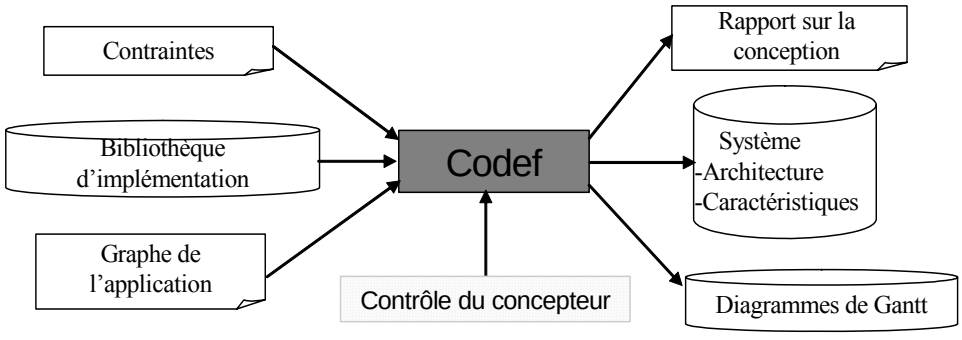


Figure 10 : Description des entrées et sorties de l'outil CODEF

Cet outil a été enrichi par [Guitton 2003] pour tenir compte de la consommation. La consommation des tâches est fournie en entrée dans la bibliothèque. CODEF effectue une exploration temps/surface. Une estimation de la consommation des architectures est également effectuée. Un raffinement est ensuite réalisé pour optimiser la consommation.

5.3. DIVISION DE L'EXPLORATION EN PLUSIEURS PARTIES

5.3.1. Séparation de l'exploration et de la communication

[Lahiri 2001] propose une approche d'exploration divisant l'exploration en deux parties (Figure 11). La première partie est un partitionnement logiciel/matériel ne tenant pas compte des temps et des conflits qui peuvent venir de l'architecture de communication. La deuxième consiste à explorer l'architecture de communication.

La première étape se base sur l'outil de codesign pour les systèmes temps réel embarqués POLIS [Balarin 1997], [Balarin 1999]. Le concepteur peut spécifier son application avec son propre langage de haut niveau comme Esterel [Berry 1992], FSM graphique, ou un des sous-ensembles Verilog ou VHDL, etc. Ce langage sera ensuite traduit en CFSM (*Concurrent Finite State Machines*) prenant en charge la concurrence à l'extérieur des modules.

POLIS effectue ensuite la vérification formelle de cette spécification à l'aide d'outils externes de vérification formelle se basant sur les FSM. En effet, POLIS permet l'interface automatique avec des outils externes.

L'étape de partitionnement inclut le partitionnement logiciel/matériel, la sélection de l'architecture cible et l'ordonnancement. L'architecture cible est constituée d'un ou plusieurs microcontrôleurs avec des RTOS et des unités matérielles.

Les sous-ensembles de CFSM destinés à une implantation matérielle sont synthétisés au niveau logique en utilisant des outils extérieurs : la synthèse logicielle inclut la génération de code C pour les CFSM logicielles ainsi que la génération d'un système d'exploitation spécifique à l'application.

La deuxième étape est la synthèse des communications. La synthèse des communications (Figure 11) se fait comme suit :

- ✓ Une cosimulation de l'architecture indépendamment des communications est établie par PTOLEMY [Buck 1994].

- ✓ A partir de la cosimulation, un premier graphe d'analyse des communications **CAG** (*Communication Analysis Graph*) est extrait.
- ✓ Ensuite, l'architecture de communication est choisie. Les supports de communication sont décrits par un modèle comportemental comportant leurs tailles, leurs fréquences, la taille du DMA et le temps de latence. Selon ces choix, les temps et les latences de communication sont injectés dans le CAG. Ceci permet d'estimer le temps total de l'application en tenant compte des communications.
- ✓ Si le temps trouvé satisfait les contraintes, l'architecture de communication est adoptée. Sinon, les paramètres des supports d'exécution seront modifiés jusqu'à ce que le temps total de l'application satisfasse les contraintes.

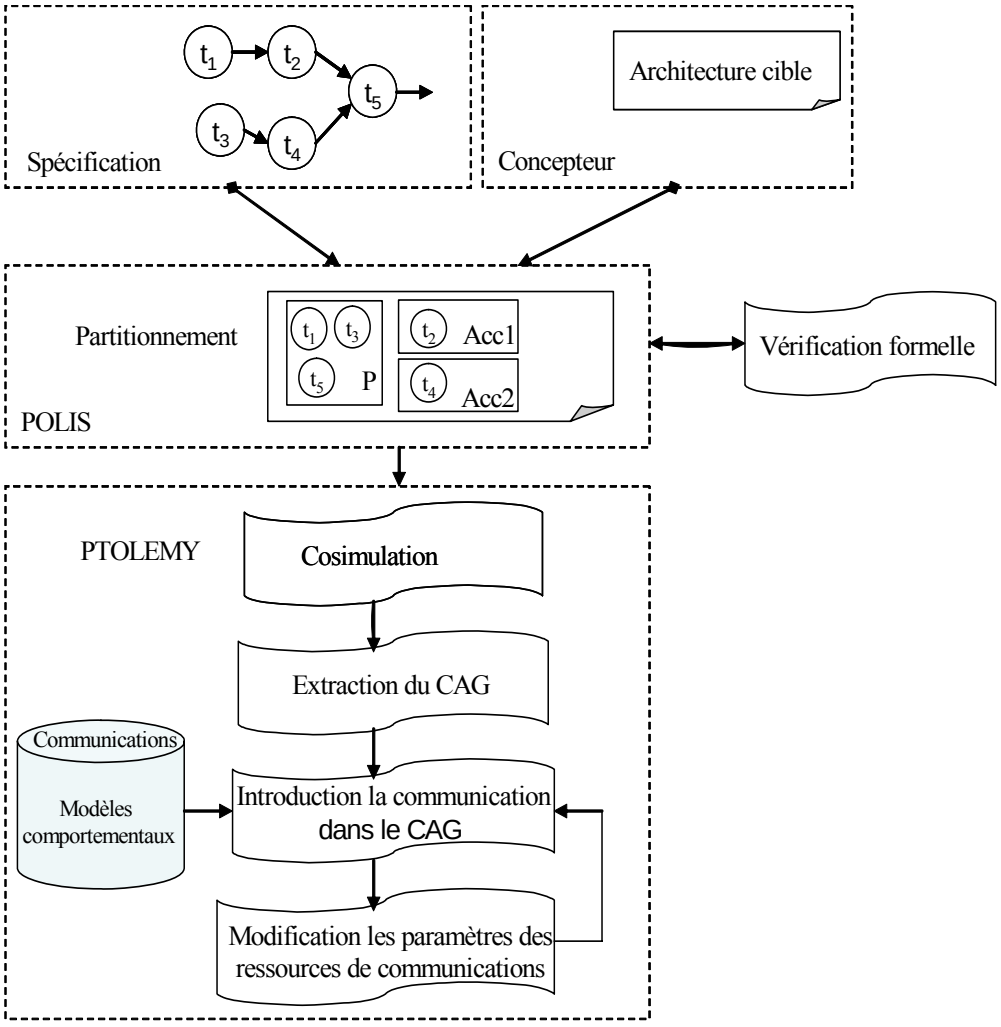


Figure 11 : Approche d'exploration séparant partitionnement et communication

Cette approche permet d'optimiser les communications d'une façon dynamique. L'optimisation est très précise et fiable. Toutefois, si l'outil n'arrive pas à satisfaire les contraintes quels que soient les paramètres de l'architecture de communication, cette dernière

est changée manuellement et ainsi de suite, jusqu'à trouver une architecture de communication permettant d'avoir un temps d'exécution de l'application qui satisfasse les contraintes. Aussi, il se peut qu'avec une architecture de communication, l'application n'arrive pas à satisfaire les contraintes. Néanmoins, si nous changeons de partitionnement logiciel/matériel, en conservant la même architecture de communication, l'architecture satisfait largement les contraintes.

5.3.2. Cosyn et COHRA

COSYN [Dave 1999] et COHRA [Dave 1998] proposent une approche d'exploration d'architectures en deux étapes principales. La première (Figure 12.b) consiste à découper l'application en plusieurs groupes de tâches. La seconde (Figure 12.c) consiste à allouer à chaque groupe une ressource d'exécution PE (*Processor Element*).

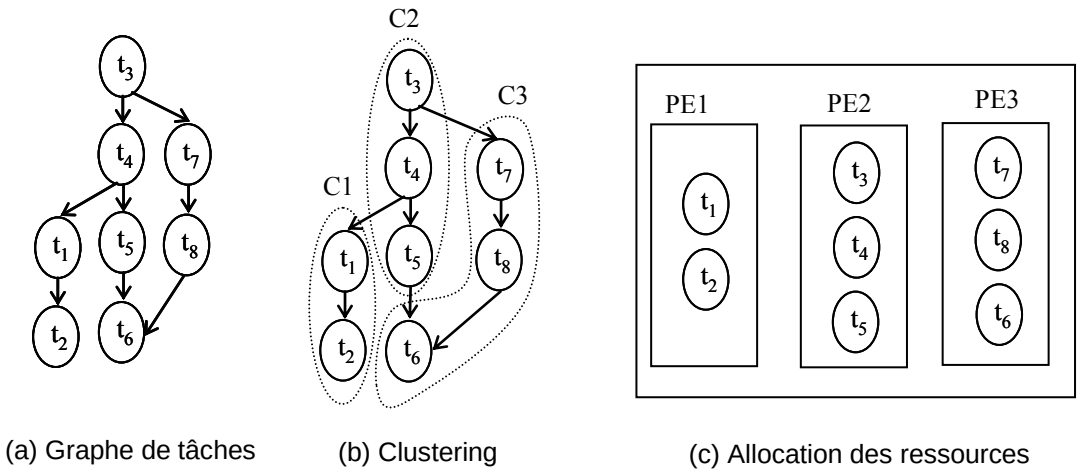


Figure 12 : Exploration de l'architecture par COSYN

Cette méthodologie considère un graphe de tâche périodique (Figure 12.a) comme spécification de l'application. Pour chaque tâche, nous connaissons la date de début au plus tôt (*earlier start time*), la période et la date de fin impérative (*deadline*). Ensuite, une collecte d'informations est effectuée. Elle consiste à fournir six groupes de vecteurs pour chaque tâche t_i :

- ✓ Vecteur d'exécution (t_i) = $\{\alpha_{i1}, \alpha_{i2}, ..., \alpha_{ij}, ...\alpha_{in}\}$: α_{ij} est le temps d'exécution de la tâche t_i par la ressource d'exécution PE j. Cette valeur est collectée soit par estimation, soit par émulation.
- ✓ Vecteur de préférence (t_i) = $\{\gamma_{i1}, \gamma_{i2}, ..., \gamma_{ij}, ...\gamma_{in}\}$: γ_{ij} est un entier qui peut être soit 0, soit 1. Il permet de signaler si la tâche t_i ne peut pas être exécutée par une des

ressources d'exécution j. Si γ_{ij} est égal à 0, alors la tâche t_i ne peut pas être exécutée par le support d'exécution j. Si γ_{ij} est égal à 1, alors il n'y a pas de contrainte qui empêche la tâche t_i de s'exécuter sur le support d'exécution j.

- ✓ Vecteur d'exclusion (t_i) = $\{\delta_{i1}, \delta_{i2}, ..., \delta_{ij}, ...\delta_{in}\}$: δ_{ij} est un entier qui peut être soit 0, soit 1. Il permet de signaler si la tâche t_i doit être exécutée par une des ressources d'exécution j. Si δ_{ij} est égal à 1, alors la tâche t_i doit être exécutée par le support d'exécution j. Si γ_{ij} est égal à 0, alors il n'y a pas de contraintes.

- ✓ Vecteur de communication (e_k) = $\{\beta_{k1}, \beta_{k2}, ..., \beta_{kj}, ... \beta_{km}\}$: e_k est un des arcs du graphe de tâches de l'application. Il représente la dépendance et le transfert des données entre deux tâches. β_{kj} est le temps qu'il faut pour transférer les données de l'arc e_k par la ressource de communication j. Cette valeur est collectée par simulation ou par modélisation de la ressource de communication.

- ✓ Vecteur de mémoire (l) = $\{\Omega_{l1}, \Omega_{l2}, ..., \Omega_{lj}, ... \Omega_{lr}\}$: Ω_{lj} est le temps d'accès par paquet, où j est le nombre de communications du lien l.

- ✓ Vecteur de consommation moyen (t_i) = $\{\xi_{l1}, \xi_{l2}, ..., \xi_{lj}, ... \xi_{ls}\}$: ξ_{lj} est la puissance moyenne de la tâche t_i si elle est exécutée par la ressource d'exécution j. Cette valeur est aussi collectée par simulation ou expérimentation.

Considérant ces informations, l'espace des découpages est parcouru à la recherche du découpage qui optimise le mieux le temps, la surface et la consommation (Figure 12.b). Une fois le découpage effectué, chaque groupe noté *cluster* est affecté à une ressource d'exécution.

Cette méthode permet de prendre en compte à la fois les ressources d'exécution, les mémoires et les communications. Aussi, elle permet d'optimiser à la fois et au même niveau le temps et la consommation. Un autre avantage de cette méthode est qu'elle peut traiter un nombre important de tâches (1000 d'après les informations données).

Toutefois, le nombre d'informations nécessaires pour cette méthode est assez important. Et, surtout, il y a un nombre élevé d'informations qui nécessite des expérimentations ou des simulations. Par conséquent, la collecte d'informations aura un coût assez élevé en temps et en argent.

5.3.3. MOCSYN

[Fei 2002] et [Dick 1999] proposent une méthode d'exploration (MOCSYN) qui se base sur trois étapes (Figure 13), la troisième étape étant imbriquée dans la deuxième :

- ✓ La sélection de la fréquence : elle consiste à trouver un compromis entre le temps d'exécution et la consommation. L'architecture utilise un seul oscillateur pour générer l'horloge pour le circuit. Ils utilisent un synthétiseur d'horloge [Bazes 1996] pour produire les fréquences des ressources d'exécution de l'architecture. La sélection de la fréquence permet de choisir la fréquence de l'oscillateur. Elle est calculée pour être un compromis entre les différentes fréquences des supports d'exécution.
- ✓ Boucle de découpage : elle consiste à choisir les ressources d'exécution (RE) qui seront utilisées pour l'architecture. Ensuite, l'architecture est choisie en utilisant ces ressources par la boucle d'architecture.
- ✓ Boucle d'architecture : elle consiste à bâtir l'architecture en utilisant les ressources d'exécution choisies par la boucle de découpage. Il s'agit d'affecter à chaque tâche une ressource d'exécution. Ensuite, il s'agit de fixer la priorité des communications, de calculer la mémorisation, de définir la structure du bus, de calculer la priorité des tâches, d'affecter les communications et d'établir l'ordonnancement des tâches. Enfin, une évaluation de l'architecture est effectuée pour voir si l'architecture satisfait les conditions d'arrêt des deux boucles.

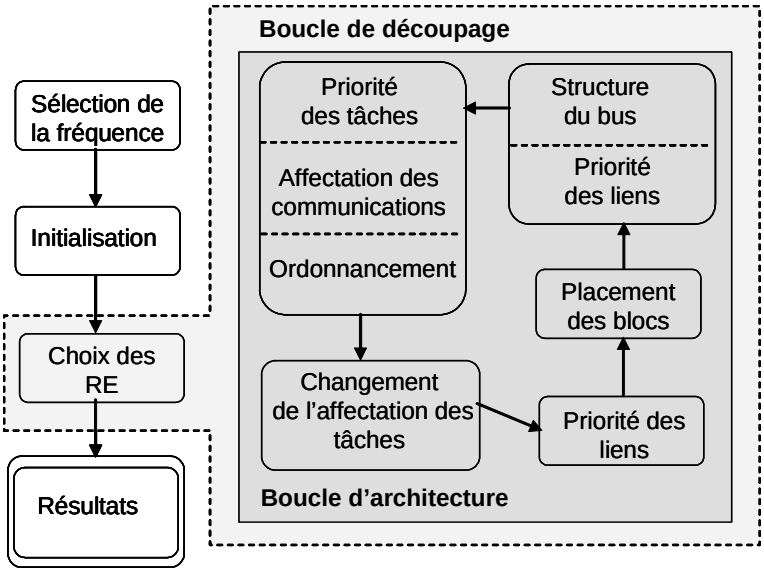


Figure 13 : Approche d'exploration d'architectures par MOCSYN

Pour leurs calculs, les deux boucles ont besoin du coût en dollars et du temps d'exécution de chaque tâche sur chacun des supports d'exécution. Plus les nombres de supports et de tâches sont importants, plus le coût de la collecte d'informations est élevé. Toutefois, cette méthode permet de traiter la possibilité d'avoir une multitude de supports d'exécution (logiciels et matériels) dans l'architecture, et de choisir parmi les ressources mises à la disposition du concepteur celles qui conviennent le mieux à l'application traitée.

5.4. PLATEFORMES

[baghdadi 2000] présente une plateforme MFSAM (*Modular Flexibal Scalable architecture Model*). Ce modèle (Figure 14) se compose de plusieurs processeurs et d'accélérateurs matériels ayant chacun sa propre mémoire et ses ressources de communication, tous connectés à travers un réseau de communication.

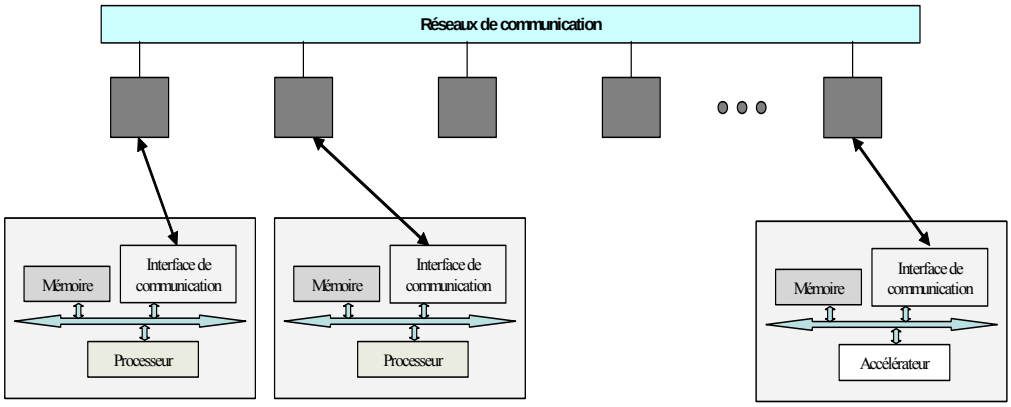


Figure 14 : Modèle d'architecture MFSAM

Ce modèle comporte comme paramètre le nombre de ressources logicielles (processeurs), les canaux de communication de chaque processeur pour les échanges de données internes (avec la mémoire) et avec les autres ressources du système, les types de processeurs, l'architecture mémoire et le type du réseau de communication. L'auteur propose d'explorer les paramètres de la plateforme par des outils de codesign.

Cette approche permet une exploration et un prototypage rapide de l'architecture. Toutefois, le modèle de cette plateforme peut conduire à une sur-utilisation des mémoires et des ressources de communication. Par exemple, un accélérateur et un processeur peuvent se partager des ressources de communication et de mémorisation sans que cela nuise aux performances du système. La plateforme ne permet pas la conception d'une telle architecture.

5.5. ANALYSE DE HAUT NIVEAU

[Brandolese 2006] propose une approche d'exploration se basant sur une analyse au niveau système (Figure 15). Cette approche est l'enrichissement de l'approche présentée dans [Sciuto 2002]. Elle se base sur une analyse au niveau système des tâches pour définir à l'aide de la métrique Analyse d'identité leurs supports d'exécution. Ensuite, chaque tâche est affectée au support d'exécution que la métrique lui attribue. [Brandolese 2006] a ajouté à la métrique affinité quatre autres métriques :

- ✓ **Affinity index I_A** : est une valeur entre 0 et 1 qui indexe la qualité de distribution des tâches par rapport aux supports d'exécution. Elle se calcule en se basant sur la métrique affinité **A**. C'est un vecteur de 3 valeurs comprises entre 0 et 1 qui sont AGPP pour l'affinité au processeur d'ordre général (General Purpose Processor), ADSP pour l'affinité au DSP et AHW pour l'affinité à une exécution matérielle. L'affinité d'une tâche par rapport à un support d'exécution est le degré d'optimisation du temps d'exécution de cette tâche si elle est exécutée par ce support d'exécution. L'analyse est faite par [Sciuto 2002].
- ✓ **Load index I_{Lsw} et L_{Lhw}** : est une valeur entre 0 et 1 visant la qualité de distribution de la charge de travail de chaque ressource d'exécution logicielle (I_{Lsw}) et matérielle (L_{Lhw}). Ces métriques se basent sur des estimations relevées à partir de la cosimulation.
- ✓ **Communication Index I_c** : est une valeur entre 0 et 1 qui indexe les communications entre les ressources d'exécution de l'architecture. Plus la quantité des données échangées entre les ressources d'exécution est faible, plus Ic est proche de 1. Cette métrique se base sur des estimations relevées à partir de la cosimulation.
- ✓ **Physical index $I_\$$** : est une valeur entre 0 et 1. Elle s'obtient par la division du coût de l'architecture par rapport au coût maximal que peut avoir l'architecture. Le coût de chaque support d'exécution est relevé de la bibliothèque (Figure 15).

L'exploration de l'architecture consiste à trouver l'architecture qui maximise la fonction de coût **CF** :

$$CF = w_A \times I_A + w_{Lsw} \times I_{Lsw} + w_{Lhw} \times I_{Lhw} + w_c \times I_c + w_\$ \times I_\$$$

$w_A, w_{Lsw}, w_{Lhw}, w_c \; et \; w_\$$ sont des poids associés aux différentes métriques pour paramétrer l'importance de chaque métrique.

Cette méthode permet à très haut niveau d'explorer l'architecture. Ceci permet une grande économie en collecte de performance des tâches.

Toutefois, l'analyse ne permet de choisir que parmi trois ressources d'exécution : GPP, DSP et matérielle. Différents types de processeurs spécifiques, comme celui proposé par [kappen 2006], sont de plus en plus utilisés et ne sont pas pris en compte par cette approche. Aussi, nous rappelons que l'exploration d'architectures consiste à trouver l'architecture qui optimise le mieux les performances temps, surface et consommation. Alors, explorer l'architecture sans calculer ces performances peut induire parfois à trouver des architectures qui ne satisfont pas les contraintes. En outre, le fait de transformer une exploration multi-objectif en une exploration mono-objectif en pondérant les différentes métriques peut détériorer la qualité de la solution trouvée.

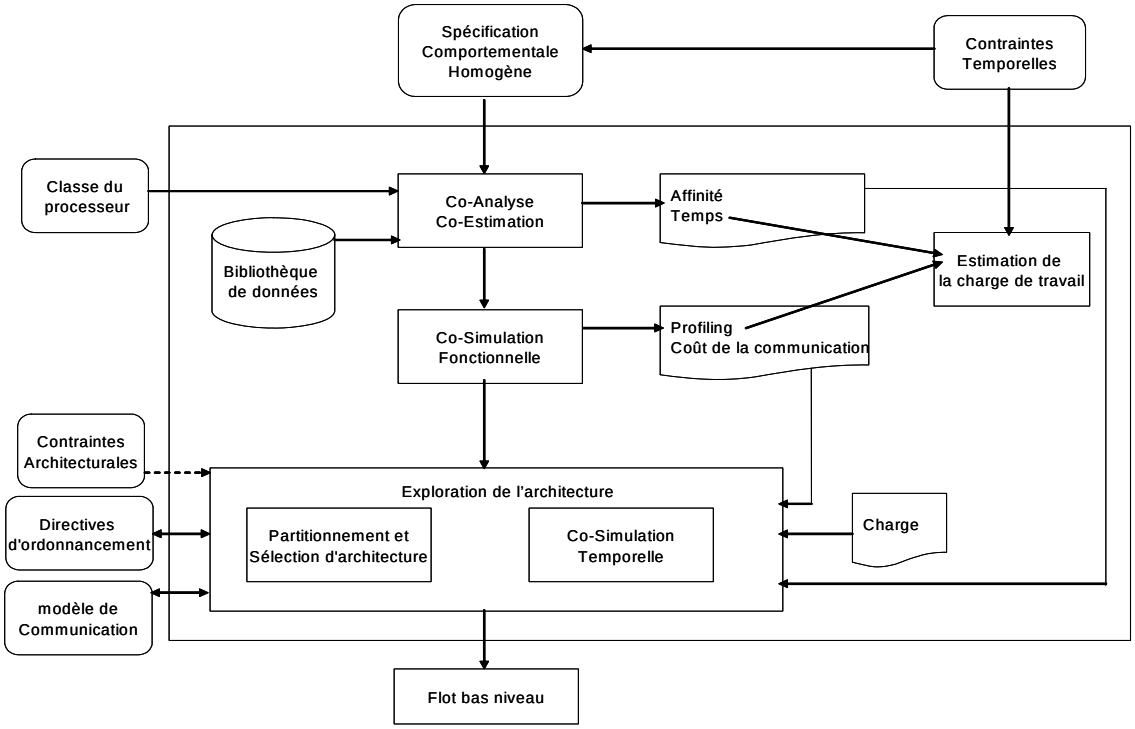


Figure 15 : Flot proposé dans [Brandolese 2006]

5.6. RESUME

Un résumé des approches d'exploration est illustré dans le Tableau 2. Les méthodes d'exploration directes sont précises mais elles ont un coût important du fait qu'elles nécessitent le rapport temps/surface/consommation de chaque tâche pour chaque support d'exécution, et auront beaucoup de difficulté quand le nombre de tâches et le nombre de processeurs deviennent élevés puisque l'espace d'architectures devient trop grand et donc complexe à explorer.

Approche	Niveau	Précision	Taille de E.A* possible	Coût de l'exploration
Exploration manuelle	Niveau système	Moyenne	Moyenne	Moyen
Exploration directe	Niveau système	Grande	- 2 à 4 processeurs - Vingtaine de tâches	Très grand
Séparation exploration communication	Niveau système	Assez grande	Moyenne	Très grand
Découpage de l'application	Niveau système	Assez grande	Nombre important de tâches et de processeurs	Très grand
Plusieurs étapes imbriquées	Niveau système	Grande	-Dizaine de processeurs - Nombre moyen de tâches	Très grand
Plateforme	Niveau système	Grande	Petite ou moyenne	Moyen
Très haut niveau	Niveau système indépendant de la technologie	Moyenne	Très grande	petit

***E.A : Espace d'architectures**

Tableau 2 : Résumé des approches d'exploration d'architectures

Une première solution est de séparer l'exploration des éléments de l'architecture en deux. Cette méthode reste coûteuse et peut être précise sous certaines conditions. En effet, comme les performances des différents éléments de l'architecture sont dépendantes les unes des autres, optimiser une partie puis l'autre peut induire des erreurs.

Le découpage de l'application pour guider l'exploration vers un espace plus réduit permet une bonne précision et l'exploration d'un grand espace d'architectures. Toutefois le coût de l'exploration reste important puisqu'il faut toujours estimer ou mesurer les

performances des tâches pour chaque support d'exécution en plus des performances de communication et de mémoire.

Découper l'exploration en deux étapes imbriquées est une approche plus précise que la précédente puisqu'il y a des retours entre les deux étapes. Toutefois le calcul pour l'exploration est plus lourd à cause des retours entre les deux étapes. Comme l'approche précédente, cette approche permet d'explorer un grand espace d'exploration (plusieurs processeurs et un nombre important de tâches dans l'application), mais elle reste très coûteuse.

Pour diminuer le coût de l'exploration, l'utilisation des plateformes est une solution pratique puisque l'espace d'exploration se limite à celui de la plateforme et donc reste restreint. Toutefois, cette limitation restreint aussi les possibilités architecturales, l'espace des solutions reste restreint et il se peut que parfois il ne contienne pas de solution qui satisfasse les contraintes.

Enfin, trouver une alternative aux performances comme fonction objectif de l'exploration permet de se dispenser de l'estimation et des mesures coûteuses des performances de chaque tâche pour chaque support d'exécution. Cela permet aussi d'explorer un espace d'architectures important. Toutefois, la fonction de coût choisie ne permet pas une aussi grande précision que celle des performances.

6. CONCLUSION

Les architectures des systèmes embarqués évoluent vers des architectures comportant de plus en plus de ressources logicielles pour faire face à l'évolution des applications et leurs besoins technologiques. Nous avons étudié plusieurs exemples qui ont déjà commencé à développer ce type d'architecture.

Les architectures sont évaluées par trois performances : le temps d'exécution, la surface et la consommation. Nous avons étudié l'influence des communications, mémoires, ressources d'exécution et parallélisme sur ces performances. Cette étude sera utilisée dans notre approche.

Face à la complexité croissante de l'exploration d'architectures, plusieurs approches ont essayé de faire évoluer les approches traditionnelles directes pour optimiser les coûts et la complexité de la résolution (la taille de l'espace d'architectures à explorer). Guider l'exploration par une étape en amont permet de faire face à une complexité plus grande sans

augmenter le coût. Proposer une analyse indépendante de la technologie améliore le coût mais au détriment de la précision.

Nous proposons, dans notre travail, une approche qui guide l'exploration par une étape amont à l'exploration pour augmenter la complexité que l'approche peut appréhender. Cette étape est une étape d'analyse de haut niveau indépendante de la technologie d'où une diminution des coûts de l'exploration. La deuxième étape est une étape d'exploration traditionnelle et donc elle permet une bonne précision. Cette approche est détaillée dans les chapitres suivants.

III. FLOT D'EXPLORATION D'ARCHITECTURES

1. INTRODUCTION 43
2. ARCHITECTURE MULTIPROCESSEUR ET NOUVEAUX DEFIS 45
2.1. LES RESSOURCES LOGICIELLES 46
2.2. LES RESSOURCES MATERIELLES 47
2.3. LES RESSOURCES DE COMMUNICATION 47
2.4. LES RESSOURCES DE MEMOIRES 48
2.5. NOUVEAUX DEFIS DES ARCHITECTURES 48
3. NOUVELLE APPROCHE D'EXPLORATION 49
4. MODELE D'ARCHITECTURE MULTI PACM 52
4.1. DEFINITION DE L'ARCHITECTURE MULTI PACM 52
4.2. DIMENSION SPATIALE 53
4.3. PARALLELISME 54
4.4. DIVERSITE LOGICIELLE 54
4.5. GENERICITE DU MODELE 55
5. ANALYSE DE HAUT NIVEAU 56
5.1. ANALYSE DES TACHES 56
5.2. ANALYSE DU CONCEPTEUR 57
5.3. GRAPHE DE SPECIFICATION 57
6. PRE-EXPLORATION 61
6.1. REDUCTION DE L'ESPACE DES ARCHITECTURES 61
6.2. MODELE D'ANALYSE 62
6.3. EXPLORATION DE L'ESPACE DES PARTITIONS ET DE L'ARCHITECTURE 64
7. CONCLUSION 65

1. INTRODUCTION

Le but de l'exploration d'architectures est de trouver pour une application l'architecture qui optimise le mieux ses performances. En particulier en terme de temps d'exécution, de surface et de consommation. L'exploration, en général, consiste à analyser les performances des architectures faisant partie de l'espace des solutions pour obtenir celle optimale suivant une fonction ou un vecteur de coût : temps, surface et consommation.

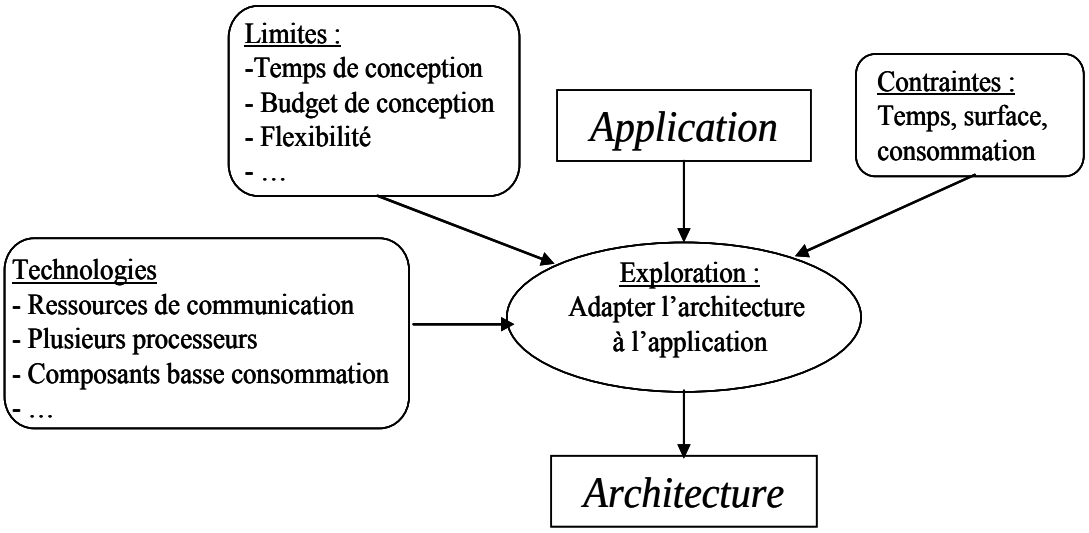


Figure 16 : Approche classique d'exploration d'architectures

Les méthodes d'exploration traditionnelles (Figure 16), consistent à chercher une architecture qui :

- ✓ Tient compte de la complexité de l'application : le nombre important de tâches, les échanges de données, le parallélisme, etc.
- ✓ Satisfait les contraintes : temps, surface et consommation.
- ✓ Supporte les pressions de l'industrialisation : limite du temps de conception lié à la concurrence qui impose un temps de mise sur le marché plus court, limite dans le budget de conception pour pouvoir mettre le produit sur le marché à des prix concurrentiels, assurance de la flexibilité pour assurer des modifications rapides du produit si le marché l'impose.
- ✓ Profite de l'évolution de la technologie : il y a de plus en plus de ressources de communication performantes. Le nombre de processeurs pouvant être intégrés dans une seule puce croît avec le temps. Cette évolution technologique permet d'avoir des

architectures plus performantes, un nombre d'architectures performantes plus important mais aussi un nombre de possibilités d'architecture croissant.

Toute la pression de l'exploration se concentre sur l'architecture. En effet, c'est l'architecture qui doit satisfaire et tenir compte des différents points présentés ci-dessus. Il suffit qu'un de ces points devienne plus complexe pour que l'élaboration de l'architecture soit elle aussi plus complexe. Dans l'état actuel des choses, l'évolution du marché impose une évolution à la fois des applications, de la technologie, des contraintes et des limites dues aux pressions de la concurrence. Par conséquent, la pression sur l'architecture est de plus en plus importante. Il faut donc que les méthodes d'exploration d'architectures actuelles évoluent vers des méthodes qui permettent d'alléger cette pression sur l'exploration de l'architecture.

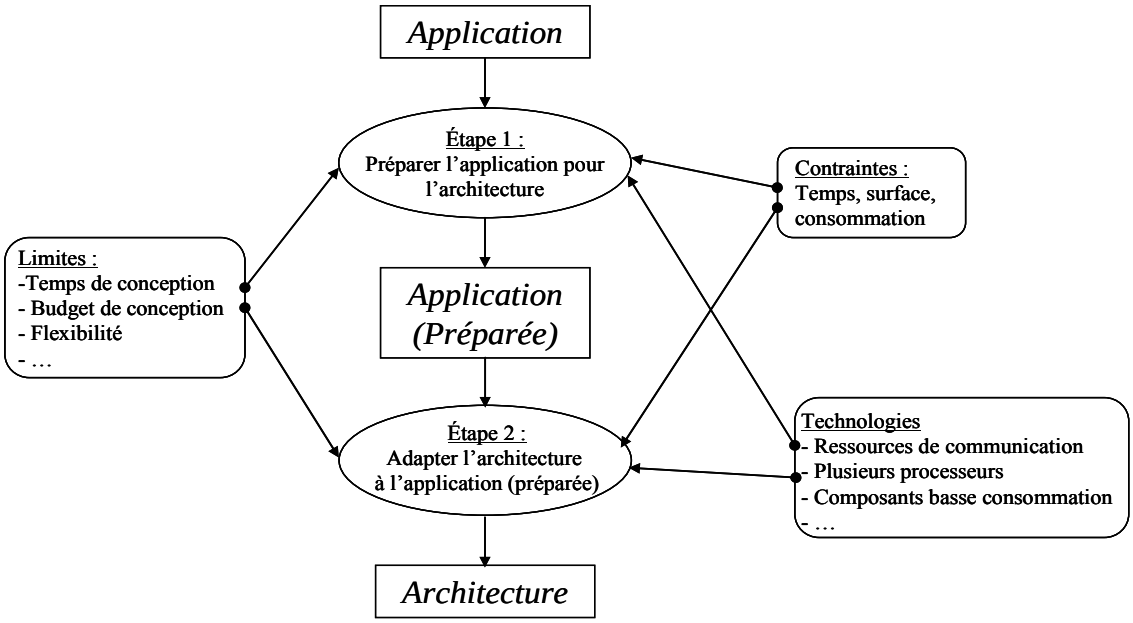


Figure 17 : Nouvelle approche d'exploration d'architectures

Afin d'alléger la pression de l'exploration sur l'architecture, nous proposons une nouvelle approche qui consiste à diviser l'exploration en deux parties (Figure 17). La première partie consiste à définir une représentation intermédiaire raffinée de l'application, en tenant compte des différentes contraintes de l'exploration, pour la préparer afin de s'adapter aux besoins de l'architecture (Figure 17 - étape 1). Il s'agit donc de trouver une représentation de l'application qui simplifie la conception de l'architecture. Par exemple, il faut mettre en œuvre une représentation de l'application qui présente une quantité de données échangées entre les tâches peu importantes. L'étude de cette étape est détaillée dans les chapitres 3 et 4. La deuxième étape (Figure 17 - étape 2) représente l'exploration traditionnelle qui consiste à adapter l'architecture à l'application. Il est à noter que dans ce flot, la première étape a déjà tenu compte d'une partie des contraintes de l'exploration d'architectures et a donc allégé

l'exploration de certaines pressions. En effet, l'étape 1 permet de trouver une représentation de l'application qui simplifie la conception et l'exploration. Par conséquent, la conception aura un coût moindre et le temps de conception sera moins élevé, d'où une atténuation de la pression du marché sur l'exploration (budget et temps de mise sur le marché).

En premier lieu, nous étudions les nouveaux défis des nouvelles architectures multiprocesseurs. Ensuite, nous détaillons l'approche d'exploration que nous proposons pour faire face à la fois à la complexité croissante des applications et aux défis engendrés par les nouvelles architectures.

2. ARCHITECTURE MULTIPROCESSEUR ET NOUVEAUX DEFIS

Dans notre travail, nous considérons des architectures pour des applications modernes et futures. Ces applications ont un nombre important de tâches, une complexité importante de transfert des données et nécessitent des architectures assez complexes. Les architectures des applications modernes et futures sont généralement des architectures multiprocesseurs. Un exemple d'une telle architecture est présenté dans la Figure 18. Cet exemple illustre une architecture qui se compose de deux processeurs, d'un coprocesseur, de plusieurs composants matériels, d'une mémoire simple port, d'une mémoire double port, d'un bus et d'un réseau de communication. Cet exemple permet déjà de nous donner une idée de la complexité de telles architectures. Nous répartissons les différents composants d'une architecture en quatre catégories : les ressources logicielles, les ressources matérielles, les ressources de communication et les ressources de mémoires. Dans cette section, nous présentons ces différentes catégories de composants. Ensuite, nous étudions l'évolution des architectures multiprocesseurs et les nouvelles difficultés qu'elles imposent à l'exploration.

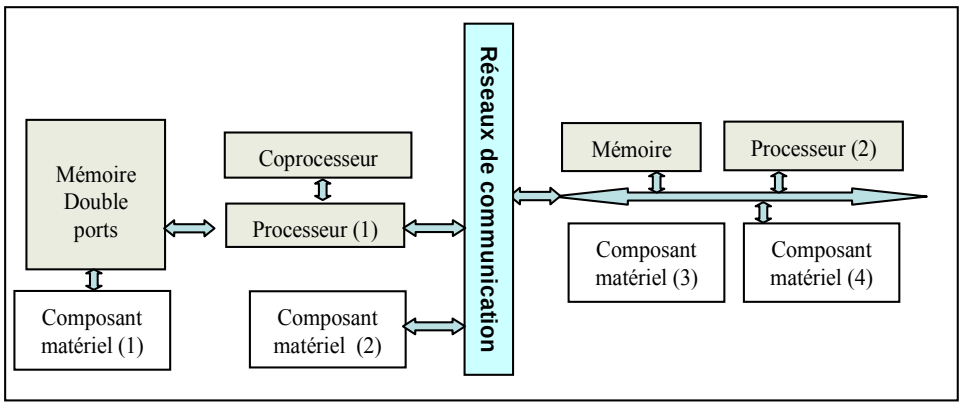


Figure 18 : Un exemple d'architecture multiprocesseur

2.1. LES RESSOURCES LOGICIELLES

Un processeur est un circuit dédié à l'exécution des instructions d'un programme. L'utilisation des composants logiciels a permis une évolution des systèmes embarqués. Il existe trois types de processeur [Airiau 2000] : processeur d'ordre général, processeur spécifique à un domaine et processeur spécifique à une application (ASIP).

Les processeurs d'ordre général sont basés sur un cœur de processeur puissant de type RISC (Reduced Instructions Set Computer) comme les processeurs LEON [Gaisler] et ARM [Arm] ou VLIW (Very long Instruction Word) comme le processeur TRIMEDIA de Philips [Kees 2002].

Les processeurs spécifiques à un domaine sont de deux types : ceux orientés vers le traitement de signal connus sous le nom DSP (*Digital Signal Processor)* et ceux orientés vers le traitement de la vidéo connus sous le nom VSP (*Video Signal Processor)*. Les VSP correspondent à une évolution des DSP pour adresser les problèmes du traitement vidéo mais ne sont généralement pas dédiés à une application particulière. Nous pouvons citer comme exemples de DSP, la famille de DSP de Texas Instruments [Texas], d'Analog Device [Analog] et de ST [ST]. Comme exemples de VSP, nous pouvons citer le IDSP de NTT [Yamauchi 1992]. Ce circuit atteint une performance de crête de 300 MOPS à une fréquence de fonctionnement de 25 MHZ. Nous pouvons également citer VSP-3 de NEC [Gwendal 1997]. Ce VSP utilise une fréquence d'horloge élevée par l'utilisation intensive du pipeline. Il peut atteindre les 1,5 GOPS, fonctionnant à 300 MHZ.

Les ASIP sont des processeurs intégrant des unités fonctionnelles et un jeu d'instructions spécifiques à une application précise. [kappen 2006] propose aussi un ASIP pour un GNSS (global navigation satellite system). [Momcilovic 2006] présente un ASIP pour une estimation de mouvement adaptative pour la vidéo (Adaptative vidéo motion estimation).

Ces exemples montrent la diversité des ressources logicielles que le concepteur peut utiliser pour mettre en œuvre l'architecture. Le nombre de ressources logicielles est en perpétuelle augmentation. Les performances d'une tâche exécutée par une ressource logicielle varient d'une ressource à une autre. Nous nommons toutes les ressources logicielles d'une architecture "processeur". Donc un processeur désigne un processeur d'ordre général, un processeur spécifique à un domaine ou un processeur spécifique à une application.

2.2. LES RESSOURCES MATERIELLES

Les ressources matérielles désignent l'implémentation en circuits câblés figés d'un composant de l'application. Le terme ASIC a été souvent utilisé pour désigner ces ressources. Le terme ASIC provient de l'anglais *Application-Specific Integrated Circuit* qui signifie circuits intégrés spécifiques à l'application. Toutefois, ce terme a évolué et désigne les circuits purement matériels qui implémentent des fonctions bien spécifiques sur une même puce [Bouchhima 2006]. Cependant, un composant matériel représente l'implémentation matérielle d'une tâche particulière.

2.3. LES RESSOURCES DE COMMUNICATION

Les ressources de communication dans une architecture sont l'ensemble des bus, interfaces, routeurs et autres ressources qui assurent les échanges entre les différents composants de l'architecture. L'évolution de ces composants a été détaillée dans le chapitre 2. Toutefois, la communication dans les architectures multiprocesseurs a aussi évolué.

Le degré de parallélisme important, dans les applications actuelles et futures, est plus en adéquation avec les architectures multiprocesseurs. En effet, l'utilisation de plusieurs composants logiciels et matériels permet d'exécuter un nombre important de tâches en parallèle. Pour profiter de ce parallélisme, il faut avoir une architecture de communication qui permet d'assurer tous ces transferts de données simultanés.

En outre, le nombre important de tâches dans les applications se traduit par un nombre important de composants logiciels et matériels dans l'architecture. Ceci donne à l'architecture une dimension spatiale qui n'était pas importante jusqu'à présent. En effet, la position d'un composant peut influer sur la qualité des communications de l'architecture. Dans la Figure 19, la position du composant matériel 2 peut influer sur la qualité de la communication. Supposons que ce composant communique avec le processeur 3. En implémentant le composant dans la position (a), le nombre de ressources de communication nécessaires pour assurer l'échange des données entre le composant 2 et le processeur 3 est assez important : le réseau de communication, deux bus et une interface. Cette communication est très pénible à gérer et provoque des latences et un temps de communication important. Donc, le fait que le composant 2 soit en position (b) améliore considérablement la qualité de la communication. En effet, la seule ressource de communication utilisée est un bus qui lie les deux composants.

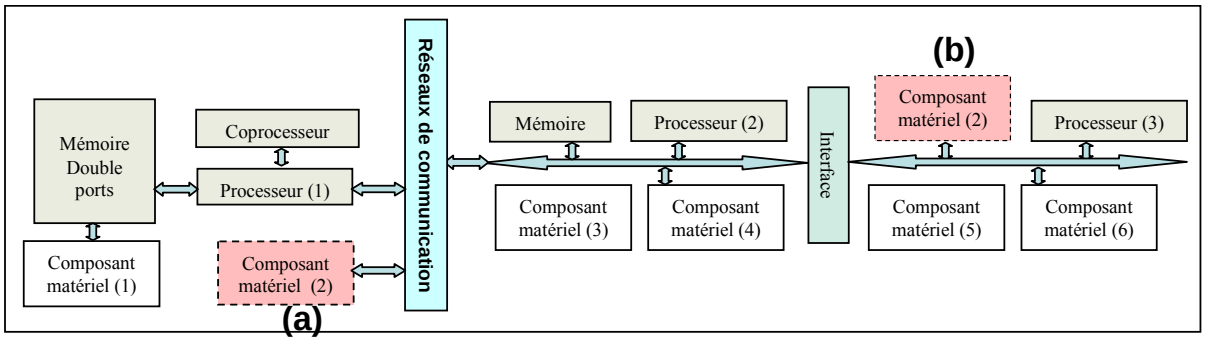


Figure 19 : Influence de la position sur les communications

2.4. LES RESSOURCES DE MEMOIRES

Une mémoire est un dispositif physique permettant la conservation et la restitution d'information ou de données. Habituellement, les seules contraintes étaient une taille suffisamment grande pour pouvoir stocker toutes les données et les codes et un temps d'accès le plus petit possible.

Toutefois, le degré de parallélisme croissant impose à l'architecture un nombre adéquat de mémoires pour assurer un nombre important d'écritures et de lectures simultanées dans les mémoires. Le nombre élevé de composants dans l'architecture fait que la position des mémoires et leur nombre influent sur les performances des communications entre le composant et les mémoires, comme expliqué ci-dessus.

2.5. NOUVEAUX DEFIS DES ARCHITECTURES

Les applications modernes ont accentué les problèmes existants. Le nombre de tâches important de l'application accroît l'expansion de l'espace d'architectures. La quantité croissante des données échangées accroît la complexité de mise en œuvre des communications.

Afin de compenser la complexité de ces applications, les architectures ont évolué vers des architectures hétérogènes (logicielle/matérielle) multiprocesseurs avec un nombre de processeurs de plus en plus important. Les composants de ces architectures ont été décrits au début de cette section. L'analyse de ces composants a permis de constater l'évolution de la complexité des architectures multiprocesseurs. Le parallélisme a été fortement accentué demandant plus de ressources de communication, de mémoire et plus de ressources d'exécution. La diversité logicielle a évolué imposant de nouveaux défis aux concepteurs. Par ailleurs, un nouveau paramètre est apparu influant sur les performances des architectures. Ce nouveau paramètre est la dimension spatiale.

Diversité des ressources logicielles

La diversité logicielle existe déjà pour les architectures monoprocesseurs. Le problème consiste à choisir quel processeur utiliser pour l'architecture parmi les n processeurs dont dispose le concepteur. Généralement, le domaine ou le type de l'application permet de choisir le processeur qu'il faut utiliser. Par exemple, pour une application de traitement de signal, nous utilisons souvent un DSP.

Les applications ont évolué et peuvent comporter des traitements appartenant à des domaines différents. Les téléphones portables et les Ipods sont des exemples d'application assez connus. Dans le même appareil, nous trouvons du traitement d'image, du traitement du son, du traitement vidéo, de la télécommunication et d'autres parties d'ordre général comme l'agenda, l'alarme, etc. Le problème des architectures multiprocesseurs pour de telles applications est de choisir une combinaison de m processeurs parmi les n types de processeurs mis à disposition du concepteur.

Dimension spatiale

Le degré de parallélisme croissant, la quantité des échanges de données élevées et le nombre de composants important de l'architecture ont été la cause de l'apparition du problème de la localisation des ressources dans l'architecture. En effet, la position du composant par rapport à ceux qui communiquent avec lui influe sur les performances de l'architecture, comme démontré pendant l'analyse des composants de l'architecture multiprocesseur. Par conséquent, une tâche de l'application n'est plus caractérisée que par son support d'exécution mais aussi selon sa "proximité" avec les autres tâches.

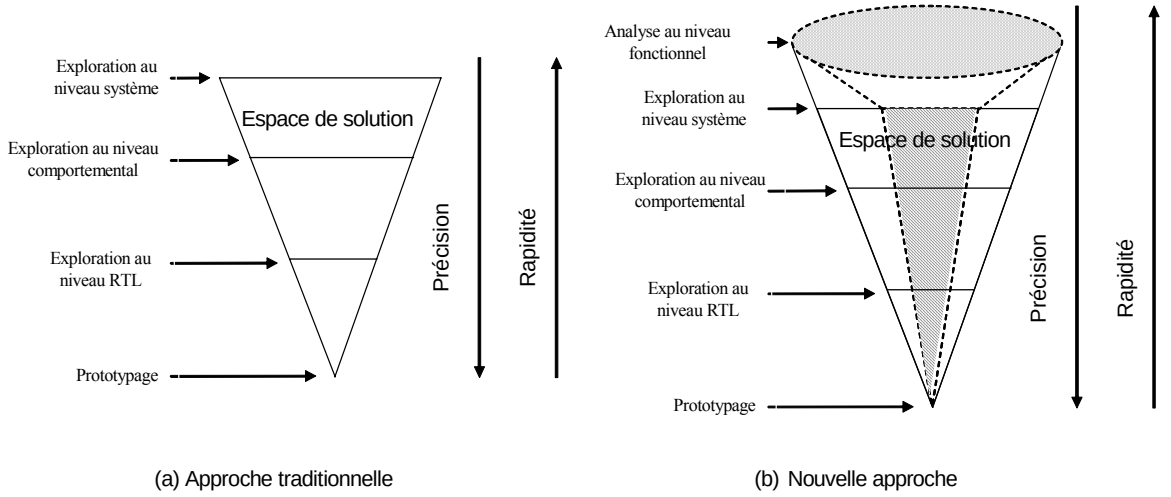
3. NOUVELLE APPROCHE D'EXPLORATION

L'analyse des nouvelles architectures multiprocesseurs a permis de découvrir des nouvelles difficultés de l'exploration : diversité logicielle, parallélisme croissant et dimension spatiale. L'analyse de l'exploration nous a permis d'analyser l'influence des différents composants de l'architecture multiprocesseur sur les performances de l'architecture. Elle a montré que, plus l'architecture de communication, l'architecture mémoire, le parallélisme et les choix du support d'exécution de chaque tâche sont optimisés, plus les performances de l'architecture sont améliorées.

Les approches d'exploration traditionnelles consistent à explorer l'espace d'architectures en tenant compte d'informations sur les paramètres technologiques de l'architecture cible. L'estimation des performances dans le cadre des approches d'exploration

a souvent un coût assez élevé du fait du nombre important des ressources technologiques (logicielles, matérielles et de communication). Aussi, l'exploration est assez complexe à cause de la croissance continue du nombre d'architectures possibles.

Les approches traditionnelles essaient d'élever le niveau d'abstraction des architectures afin de gagner en rapidité d'exploration (Figure 20.a). Certes, plus le niveau d'exploration est élevé, plus l'espace d'architectures est grand, mais en parallèle les concepteurs obtiennent plus de rapidité d'exploration à haut niveau qu'à bas niveau. Ceci s'explique par le fait que, d'une part, certaines caractéristiques à bas niveau ne sont pas indispensables pour tirer des premières conclusions, et que d'autre part, ces caractéristiques engendrent souvent un temps d'estimation relativement important. Cependant, l'espace d'architectures au niveau système peut devenir très important engendrant une complexité et un coût d'exploration très élevé.



(a) Approche traditionnelle (b) Nouvelle approche

Figure 20 : Elévation du niveau de l'exploration

Afin de faire face à ce problème, nous proposons une approche de conception qui commence l'exploration à un niveau supérieur (fonctionnel) afin de réduire l'espace d'architectures pour l'exploration traditionnelle et réduire par conséquent les coûts de l'estimation des performances et de l'exploration (Figure 20.b). La réduction de l'espace d'architectures n'est pas fait au détriment de la qualité de l'exploration. En effet, l'analyse tient compte d'un nombre suffisant de paramètres pour permettre d'éliminer les architectures les moins performantes et guider l'exploration vers un espace réduit certes, mais contenant la plupart de architectures les plus performantes. Cette approche considère à haut niveau des nouveaux paramètres des architectures multiprocesseurs modernes influant sur les performances : la dimension spatiale, le parallélisme et la diversité des ressources logicielles.

Nous proposons une approche en deux étapes (Figure 21) :

- ✓ La première étape, nommée pré-exploration ((5) de la Figure 21), consiste à analyser une spécification de l'application exprimée sous la forme d'un graphe de tâches de l'application ((1) de la Figure 21) en considérant l'analyse de l'application établie par le concepteur ((3) de la Figure 21) ainsi que l'analyse des tâches, qu'elle soit faite par des outils ou par le concepteur lui-même ((2) de la Figure 21).
- ✓ La deuxième étape, nommée exploration ((7) de la Figure 21), explore l'espace réduit par la pré-exploration pour trouver l'architecture adéquate à l'application. Cette deuxième étape se base sur l'analyse faite par la première étape pour étudier l'espace d'architectures en considérant un modèle d'architecture dit multi-PACM ((4) de la Figure 21) et génère une architecture abstraite que nous nommons partition ((6) de la Figure 21). Cette dernière guide l'exploration afin de trouver l'architecture adéquate dans un espace d'architectures considérablement réduit (Figure 20). La suite de cette section présente les points clés de cette approche.

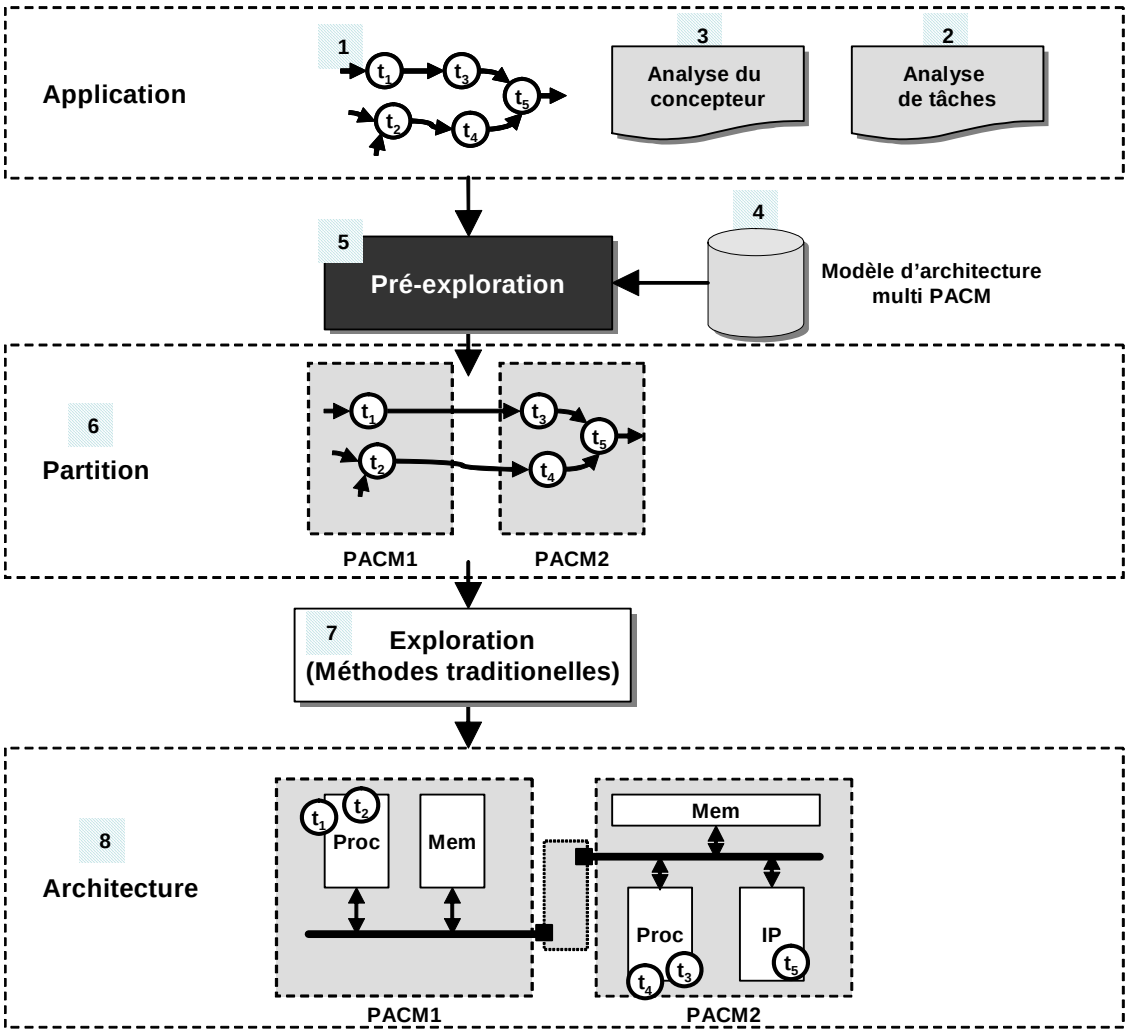


Figure 21 : Nouvelle approche d'exploration

4. MODELE D'ARCHITECTURE MULTI-PACM

Afin d'analyser l'espace d'architectures à haut niveau, il est indispensable d'avoir des informations sur l'architecture. Ces informations sont illustrées sous forme de modèle abstrait de l'architecture. Ce modèle doit être d'un côté suffisamment générique pour représenter le plus grand nombre d'architectures possibles. D'un autre côté, il doit comporter un nombre suffisant d'informations pour pouvoir bien guider l'analyse. De plus, pour les architectures multiprocesseurs, le modèle doit aussi pouvoir renseigner le concepteur à haut niveau à propos des nouveaux défis de l'exploration : la dimension spatiale, la diversité logicielle et le parallélisme.

4.1. DEFINITION DE L'ARCHITECTURE MULTI-PACM

Nous définissons un PACM (Figure 22) comme une entité comprenant un ou plusieurs de ces composants : un Processeur, un ou plusieurs Accélérateurs matériels (connu aussi comme composant matériel), un ou plusieurs Coprocesseurs et une ou plusieurs Mémoires. La communication dans un PACM est assurée par un bus ou par une interface point à point selon la complexité du PACM. L'architecture de communication est raffinée lors de l'exploration de l'architecture globale de l'application.

Une architecture multi-PACM (Figure 22) est composée de plusieurs PACM et de ressources de communication pour assurer les échanges de données. La communication entre les PACM est raffinée pendant l'exploration de l'architecture. L'architecture de communication peut être un réseau de communication, un réseau de bus selon la complexité de l'application ou même partager le même bus utilisé par les PACM. Ce modèle d'architecture fournit suffisamment d'informations sur l'architecture comme sur la dimension spatiale, le parallélisme et la diversité logicielle tout en représentant la majorité des architectures.

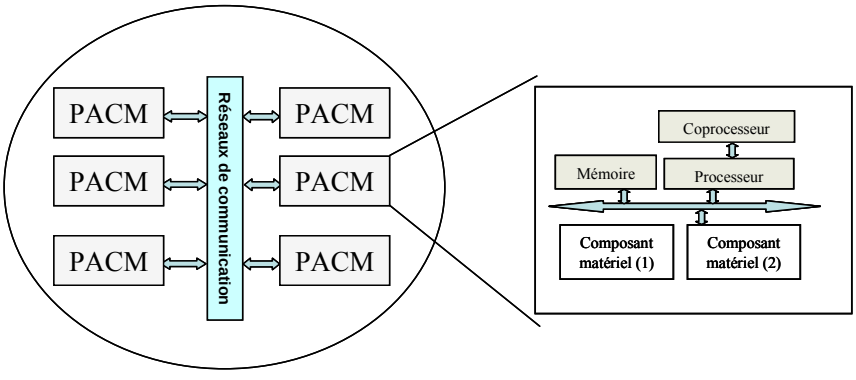


Figure 22 : Architecture multi-PACM

4.2. DIMENSION SPATIALE

La position d'un composant dans une architecture est importante du fait que deux composants qui communiquent entre eux doivent être à proximité l'un de l'autre afin d'optimiser les échanges. Il est difficile de définir la proximité dans une architecture multiprocesseur traditionnelle du fait qu'il n y a pas de notion de localisation des composants.

Pour l'architecture multi-PACM, nous définissons la proximité de deux composants comme suit :

Définition

Deux composants sont à proximité l'un de l'autre s'ils appartiennent au même PACM.

Cette définition permet de localiser les composants les uns par rapport aux autres. De plus, elle garantit l'optimisation des échanges. En effet, dans le même PACM, les composants se partagent les mêmes ressources de communication et de mémorisation. Ainsi, la proximité garantit l'optimisation des échanges de données.

La Figure 23 reprend l'exemple illustré dans la Figure 19. Le composant matériel 2 se situe à proximité du processeur 1 s'il est en position (a) car ils sont tous les deux dans le PACM 1. Mais, il est à proximité du processeur 3 s'il est en position (b) puisqu'ils sont tous les deux dans le PACM 3. La notion de PACM permet de localiser facilement la proximité des composants les uns par rapport aux autres. Ainsi, si le composant matériel 2 doit communiquer avec le processeur 3, et que dans la position (b) les deux composants sont à proximité l'un de l'autre, il est maintenant facile de juger que la position (b) est la meilleure position pour le composant matériel 2.

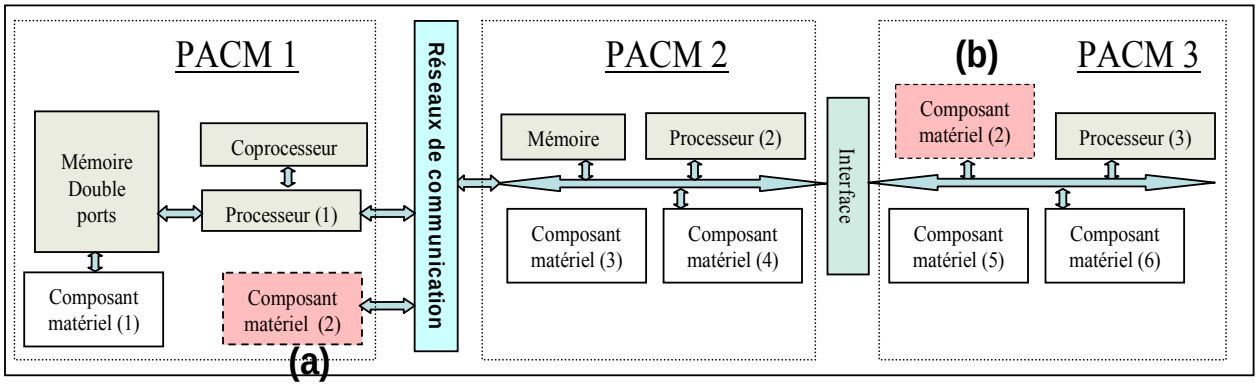


Figure 23 : Exemple de proximité de l'architecture multi-PACM

4.3. PARALLELISME

Le parallélisme entre deux tâches de l'application est possible si les deux tâches sont exécutées sur deux composants différents ou si elles sont implémentées en matériels. Ainsi, deux tâches qui sont implémentées en logiciel, doivent être exécutées sur deux processeurs différents. Pour une architecture multi-PACM, nous énonçons le lemme suivant :

Lemme

Deux tâches qui sont exécutées dans deux PACM différents auront toujours la possibilité d'être exécutées en parallèle.

En effet, si les deux tâches sont implémentées en matériel, ou l'une en matériel et l'autre en logiciel, elles peuvent toujours être exécutées en parallèle. De plus, même si les deux tâches sont exécutées en logiciel, étant donné que les deux tâches sont exécutées dans deux PACM différents, elles sont exécutées sur deux processeurs différents et par conséquent elles peuvent être exécutées en parallèle.

Par ailleurs, deux tâches qui sont implémentées dans le même PACM peuvent aussi être exécutées en parallèle, si elles sont implémentées en matériel, ou l'une en logiciel et l'autre en matériel. Enfin, l'architecture multi-PACM permet d'avoir une idée assez riche sur les possibilités de parallélisme de deux ou plusieurs tâches sans connaître exactement leurs supports d'exécution.

4.4. DIVERSITE LOGICIELLE

L'architecture multi-PACM peut simplifier le problème de la diversité logicielle. Afin d'étudier cette dernière nous considérons le problème suivant :

Considérons une architecture qui dispose de n processeurs et m tâches. Supposons que la technologie nous permet d'utiliser k types de processeurs différents. Le problème est de trouver de quel type sera chacun des n processeurs de l'architecture.

Pour une architecture multiprocesseur traditionnelle le problème est le suivant : sachant que chaque tâche présente une performance différente sur chaque type de processeur et sachant que deux tâches qui s'exécutent en parallèle doivent être exécutées sur deux supports logiciels différents, quel type de processeur devons-nous choisir pour chacun des n processeurs de l'architecture ? La résolution de ce problème est complexe du fait qu'il faut traiter tous les compromis et demande une quantité d'analyse très importante. La complexité

de ce problème est exponentielle du fait de l'augmentation continue du nombre de processeurs utilisés.

Pour une architecture multi-PACM, si nous connaissons les tâches qui s'exécuteront dans chaque PACM (chose rendue possible par le flot proposé et qui sera détaillé plus tard), le problème est le suivant : pour chacun des n PACM de l'architecture, sachant que chaque tâche du PACM présente une performance différente sur chaque type de processeur, quel processeur devons-nous choisir pour exécuter ces tâches ? Résoudre ce problème consiste à trouver le processeur qui convient à la plupart des tâches qui seront implémentées dans le PACM.

Avec l'architecture multi-PACM, il est possible de faire face au problème de la diversité logicielle croissante. Non seulement le problème est simplifié par rapport à une architecture normale, mais aussi il conserve la même complexité quel que soit le nombre de processeurs dans l'architecture.

4.5. GENERICITE DU MODELE

Le modèle d'architecture multi-PACM est utilisé par l'étape de pré-exploration pour analyser l'espace d'architectures à haut niveau. Il est important que ce modèle représente le plus grand nombre d'architectures possibles. Le modèle d'architecture multi-PACM peut couvrir les architectures multiprocesseurs les plus fréquemment utilisées. En effet, que ce soit une architecture classique à base de bus (Figure 24.b), que ce soit une architecture utilisant une multitude de ressources de communication (Figure 24.a) ou que ce soit une architecture à base de réseaux (Figure 24.a), le modèle multi-PACM peut couvrir toutes ces configurations (Figure 24). Bien sur, la Figure 24 ne couvre pas tout les modèles d'architecture possibles mais présente juste un exemple couvrant diverses architectures.

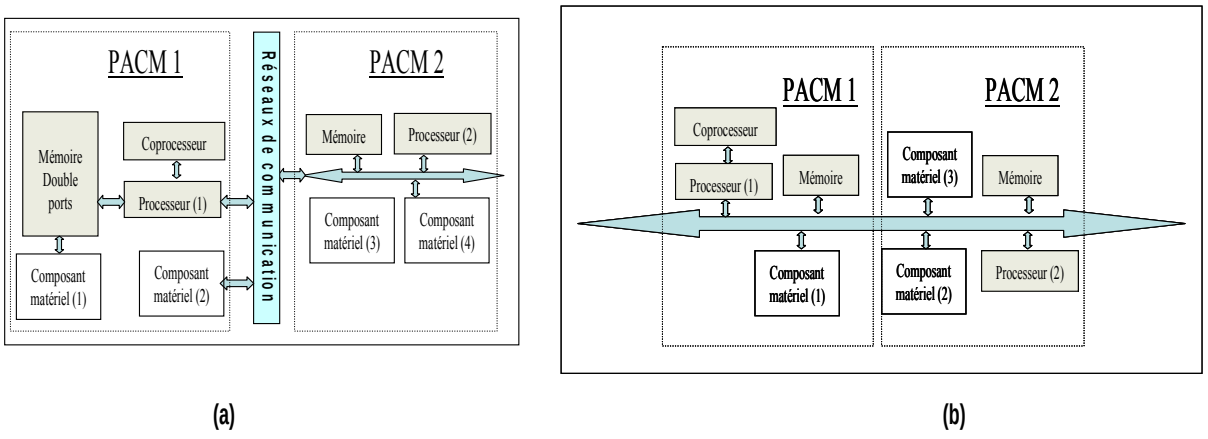


Figure 24 : Généricité de l'architecture multi-PACM

5. ANALYSE DE HAUT NIVEAU

Nous explorons l'espace d'architectures à haut niveau. Pour cela, nous récoltons le plus d'informations possibles sur l'architecture à haut niveau. Ces informations sont résumées dans le graphe de spécification que nous définissons. Le graphe est établi par une compréhension fonctionnelle de l'application, par l'analyse des tâches, qu'elle soit automatisée ou non, et par une analyse de l'application établie par le concepteur.

5.1. ANALYSE DES TACHES

Un des éléments clés de l'exploration d'architectures est de tenir compte des performances et coûts de l'implémentation des tâches sur les différents supports d'exécution. En effet, une partie importante des coûts de l'application en temps, surface et consommation provient des tâches qui la composent. Toutefois, la communication, les mémoires et l'ordonnancement des tâches sont aussi des éléments importants sur les performances du système.

Le moyen le plus simple et le plus efficace pour quantifier l'impact de l'implémentation des tâches consiste à implémenter chacune de ces tâches sur les différents supports possibles et de relever les différents coûts : temps, surface, mémoire utilisée, consommation… Toutefois, une telle méthode a un coût relativement élevé puisqu'il faut d'abord développer ou acheter l'implémentation de la tâche sur les différents supports et ensuite relever tous les coûts et performances. Il parait clairement que ce type d'approche n'est pas compatible avec la complexité à laquelle seront confrontés les concepteurs dans l'avenir.

Aussi, des travaux se sont concentrés sur l'analyse des tâches à haut niveau indépendamment de la technologie afin de donner des informations sur les performances des tâches. En effet, des travaux comme ceux de [sciuto 2002] et [le moullec 2003] permettent d'avoir une idée à haut niveau sur les performances et les coûts. [sciuto 2002] analyse les tâches de l'application en calculant leurs affinités pour des supports d'exécution différents. L'affinité d'une tâche à un support d'exécution est définie comme une valeur entre 0 et 1 représentant le degré de performance de l'exécution de cette tâche s'exécutant sur ce support d'exécution. [le moullec 2003] analyse les tâches et fournit entre autre une estimation des ressources de mémoires nécessaires pour l'exécution d'une tâche.

Ces analyses, bien qu'ayant une influence importante sur les performances de l'architecture, ne sont pas suffisantes pour explorer l'architecture. En effet, la communication,

le partage mémoire et les différentes autres interactions entre les tâches ont également une influence importante sur l'exploration de l'architecture.

Dans le flot que nous proposons, nous tenons compte de ces analyses. Plus précisément, nous tenons compte des analyses des tâches à haut niveau ((2) de la Figure 21) qui ont été réalisées à l'aide des outils automatisés d'analyse de tâches comme ceux décrits ci-dessus ou par le concepteur. Ces analyses sont utilisées pour enrichir le graphe de spécification de l'application et ensuite pour guider l'analyse à haut niveau des architectures et la réduction de l'espace d'architectures. Les informations que nous recueillons de ces analyses sont détaillées dans la suite avec la spécification du graphe de tâches.

5.2. ANALYSE DU CONCEPTEUR

Les concepteurs doivent généralement analyser l'application pour contrôler les coûts du système. En effet, ils doivent vérifier si l'architecture définie n'utilise pas beaucoup de composants non présents dans la bibliothèque. Ce type de composant augmente les coûts système puisque ces composants doivent être développés à part. Par conséquent, plus l'architecture contient de composants inexistants dans la bibliothèque, plus les coûts auront tendance à augmenter. Toutefois, il est parfois nécessaire de faire appel à des composants externes à la bibliothèque, même si cela augmente le coût de la conception, pour respecter les contraintes du système (de temps, de surface et de consommation). Par conséquent, il est difficile pour le concepteur d'établir un compromis entre les coûts et les performances du système. Pour amortir le coût de l'exploration, le concepteur est souvent amené à prendre des décisions sur l'architecture à haut niveau. Bien que ces décisions soient basées sur une expérience riche et s'avèrent donc assez judicieuses, elles peuvent fausser l'exploration d'architectures et conduire à une architecture avec des performances qui ne répondent pas aux contraintes.

Dans ce travail, nous tenons compte des différentes analyses du concepteur qui sont riches en information à haut niveau. Toutefois, au lieu de les appliquer aveuglément, nous les combinons avec les autres analyses de l'application pour guider l'exploration vers une architecture présentant des performances optimales au regard de la fonction coût définie.

5.3. GRAPHE DE SPECIFICATION

Nous proposons d'illustrer l'application par un graphe de spécification (Figure 25) qui décrit les dépendances et les échanges de données entre les tâches, ainsi que d'autres

paramètres issus de l'analyse du concepteur et de l'analyse des tâches. Ce graphe est facile à spécifier et ne demande pas une expertise approfondie de l'application.

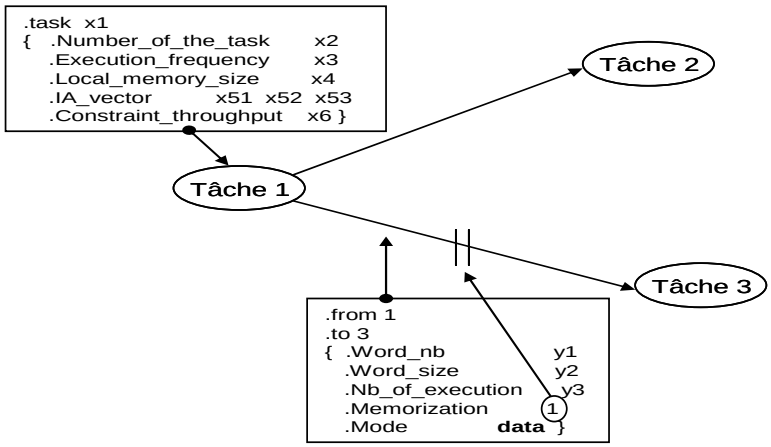


Figure 25 : Exemple du graphe de spécification

Le graphe est composé de nœuds et d'arcs. Les nœuds représentent les tâches et les arcs représentent les dépendances et les transferts de données.

5.3.1. Les nœuds

Nous définissons l'ensemble des tâches d'une application **T**. Soit **T** l'ensemble des tâches de l'application :

$$T = \{t_i, i \in [1, n]\}$$

t_i : une tâche de l'application.

n : nombre des tâches de l'application.

Un nœud représente une des tâches de l'application. Il est accompagné par des valeurs qui caractérisent cette tâche (Figure 25). La spécification d'un nœud est sous la forme d'une déclaration "***.task***" suivie du nom de la tâche que représente le nœud (x1 sur la Figure 25), puis des différents paramètres du nœud entre 2 accolades. Pour une tâche $\mathbf{t_i}$, ces paramètres sont les suivants :

- ✓ Numéro de la tâche (Number_of_the_task(ti)) : un numéro est attribué à chaque tâche afin de faciliter l'automatisation des traitements par des outils informatiques. Le numéro attribué à la tâche ti est i.
- ✓ Fréquence d'exécution (Execution_frequency(ti)) : cette valeur représente le nombre d'exécutions de la tâche ti pendant une seule exécution de toute l'application. Cette valeur est notée dorénavant Exec_freq(ti).

- ✓ Taille de mémoire locale (Local_memory_size(ti)) : elle représente l'estimation des ressources de mémoires nécessaires pour l'exécution de la tâche ti. Elle est fournie par la phase d'analyse des tâches. Cette valeur est déterminée en utilisant l'outil DesignTrotter [Le Moullec 2003]. Elle est notée dorénavant LM_size(ti).

- ✓ Vecteur d'analyse d'identité (IA_vector(ti): Identity Analysis vector) : le vecteur d'analyse d'identité de la tâche ti est un vecteur qui contient l'ensemble des analyses d'identité de cette tâche par rapport aux différents types de processeurs dont dispose le concepteur (DSP, VSP, Processeur de vision, etc.). En supposant que le concepteur dispose de k types de processeur, nous définissons PE l'ensemble des types de processeurs que le concepteur peut utiliser.

  Soit PE l'ensemble des types de processeur dont dispose le concepteur :

  $$PE = \{PE_j,\ j \in [1,k]\}$$

  PE_i : un type de processeur disponible.

  $$IA\_vector : T \rightarrow \Re^k$$

  $$t_i \rightarrow IA\_vector(t_i) = \begin{pmatrix} LIA(t_i, PE_1) \\ LIA(t_i, PE_2) \\ . \\ . \\ LIA(t_i, PE_k) \end{pmatrix}$$

  Avec :

  $$LIA : T \times PE \rightarrow \Re$$
  $$(t_i, PE_j) \rightarrow LIA(t_i, PE_j)$$

LIA(t_i,PE_j) pour 0 ≤ j ≤ k représente l'analyse d'identité de la tâche ***ti*** par rapport aux différents types de processeurs **PE_j**. Cette valeur est un chiffre entre 0 et 1 qui correspond à une note sur les performances de la tâche **t_i** si elle est exécutée par le processeur ***PE_j***. Cette note est attribuée par le concepteur (selon son expérience et son expertise de la tâche), par des estimations (par exemple, dans les travaux de [Sciuto 2002]), et par la connaissance des composants existants dans la bibliothèque disponible et du coût de probables achats ou développements. Cette valeur est proche de "1" si le concepteur pense que le processeur ***PE_j*** est le meilleur support d'exécution de la tâche ***t_i***. Cette valeur est proche de "0" si le concepteur juge qu'il faut éviter d'exécuter la tâche ***t_i*** sur le processeur ***PE_j***.

Par exemple, le concepteur peut utiliser trois types de processeur : un processeur de vision **PE_1**, un DSP **PE_2** et un processeur général **PE_3**. Une première tâche de l'application est une tâche de vision **t_1** (appliquée à la robotique). Le concepteur, sachant que cette tâche ne

sera pas du tout optimisée par le DSP et par le processeur général, décide qu'il est indispensable que cette tâche soit exécutée par le processeur de vision qui possède des unités de calcul spécifique à cette tâche. Donc, le vecteur d'identité d'analyse de la tâche $\mathbf{t_1}$ est (1 0 0). Une deuxième tâche $\mathbf{t_2}$ est plutôt destinée pour être exécutée par un DSP. Toutefois, elle peut être exécutée aussi par le processeur de vision car ce dernier a été conçu à partir du DSP. Mais, une exécution sur un processeur général dégrade les performances de cette tâche. Donc, le vecteur d'identité d'analyse de la tâche $\mathbf{t_2}$ est (0,8 0,8 0).

- ✓ Débit contrainte (Constraint_throughput(t_i)) : généralement les tâches d'entrée et de sortie et certaines autres tâches de l'application disposent de contraintes de temps à satisfaire. Ces contraintes se traduisent par un débit de traitement que ces tâches doivent respecter. Ce débit est noté **thc(t_i)**. C'est un entier qui est défini en bits par seconde.

5.3.2. Les arcs

Les arcs représentent les interactions entre les tâches comme les dépendances et les transferts de données. Les arcs sont accompagnés par des valeurs représentant ces interactions (Figure 25). Pour un arc entre les tâches $\mathbf{t_1}$ et $\mathbf{t_j}$, la spécification d'un arc est sous la forme de deux déclarations "***.from***", suivies du numéro de la tâche productrice **i** de l'arc et "***.to***", du numéro de la tâche réceptrice **j** de l'arc, puis des différents paramètres de l'arc entre 2 accolades. Ces paramètres sont les suivants :

- ✓ Nombre de mots (Word_nb(i,j)) : cette valeur indique le nombre de mots transférés d'une tâche à une autre.
- ✓ Taille des mots (Word_size(i,j)) : cette valeur représente la granularité du transfert de données.
- ✓ Nombre d'exécutions (Nb_of_execution(i,j)) : cette valeur représente le nombre d'exécutions de l'arc. Si, par exemple, une tâche envoie à une autre tâche 3 fois un tableau de 10 entiers, le nombre de mots est 10, la taille du mot est la taille d'un entier en bits et la fréquence d'exécution est de 3.
- ✓ Mémorisation (Memorization(i,j)) : cette valeur indique si le transfert de données d'une tâche est sans mémorisation (Memorization 0) ou avec mémorisation (Memorization 1). Cette valeur est fournie par le concepteur. Il décide, a priori, si ce transfert nécessitera une mémorisation ou non. Cette décision n'est pas finale. Ce choix est raffiné dans la deuxième partie de l'exploration d'architectures. Toutefois, même si cette décision n'est

pas précise, elle donne un ordre de grandeur important sur la quantité des données mémorisées dans l'architecture.

- ✓ Mode (Mode(i,j)) : le mode définit si l'arc est un arc de contrôle ou de données. Dans nos travaux, nous nous sommes focalisés uniquement sur les arcs de type données. Tous les arcs ont un mode labellisé data. Toutefois, il est possible de labelliser des arcs Ctrl pour les définir comme arcs de type contrôle. Ce type d'arc sera étudié dans de futurs travaux.

6. PRE-EXPLORATION

La description fonctionnelle de l'application, l'analyse des tâches et l'analyse du concepteur sont formulées dans le graphe de spécification de l'application. La pré-exploration se base sur le graphe de spécification et le modèle d'architecture multi-PACM pour réduire l'espace d'architectures. La réduction est décrite ci-après.

6.1. REDUCTION DE L'ESPACE D'ARCHITECTURES

La pré-exploration consiste à analyser l'espace d'architectures au niveau fonctionnel afin de réduire cet espace pour l'exploration. L'espace d'architectures, de façon grossière, est le croisement de l'ensemble des tâches avec celui des possibilités d'implémentation de chaque tâche. La pré-exploration réduit les possibilités d'implémentation de chaque tâche, ce qui réduit l'espace d'architectures. Ceci consiste à déterminer une partition adéquate de l'architecture à partir de l'analyse à haut niveau. Une partition est l'affectation de chacune des tâches à des PACM de l'architecture (Figure 26). Une tâche dans un PACM ne peut être exécutée qu'en matériel ou par le processeur du PACM. Ce choix est fait pendant l'exploration. Ainsi, les possibilités d'implémentation d'une tâche se réduisent à deux au lieu des n possibilités logicielles (n étant le nombre des types de processeurs dont dispose le concepteur) et de la possibilité d'implémentation matérielle.

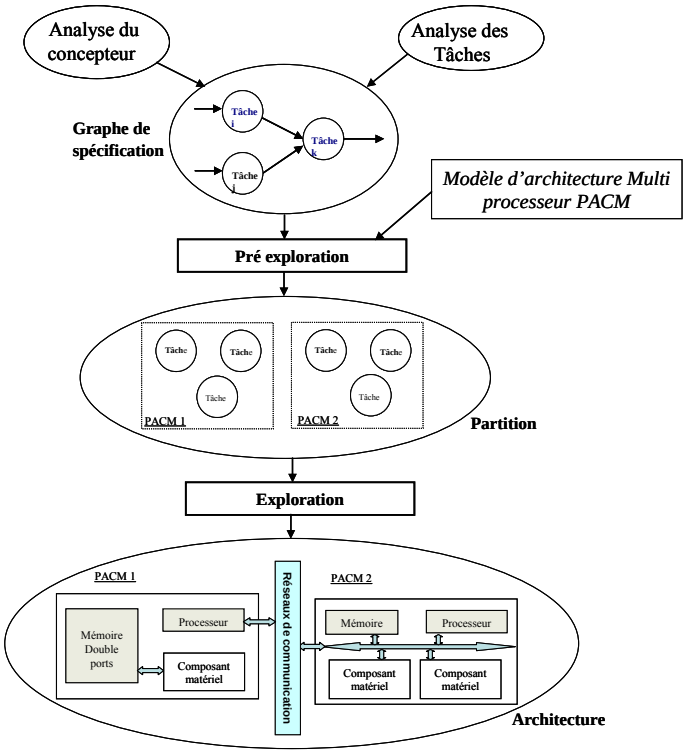


Figure 26 : Pré-exploration et exploration

Choisir une partition réduit l'espace d'architectures et allège donc le coût et la complexité de l'exploration. Toutefois, baser l'exploration sur seulement une partie de l'espace d'architectures que couvre la partition peut amener à se demander comment garantir l'obtention de bonnes architectures dans cet espace réduit. La partition est déterminée en se basant, d'un côté sur un modèle d'analyse à haut niveau permettant de comparer les partitions et de déterminer quelle est la mieux placée pour couvrir l'espace de solutions qui contient les architectures adéquates, et d'un autre côté sur un algorithme génétique qui explore l'espace des partitions pour trouver la meilleure en se basant sur le modèle d'analyse. Ci-après nous présentons le modèle d'analyse.

6.2. MODELE D'ANALYSE

Le modèle consiste à analyser les performances de l'architecture. A haut niveau, nous n'avons ni informations sur les performances des différentes tâches, ni informations détaillées sur la structure de l'architecture. Pour pouvoir évaluer les performances à haut niveau, nous étudions les principaux paramètres influant sur les performances de l'architecture. Les performances que nous considérons sont le temps, la surface et la consommation. L'étude de ces performances (chapitre 2) a montré que, pour optimiser le temps il faut optimiser

l'exécution des tâches, les communications, la mémoire et le parallélisme comme résumé dans le Tableau 3.

Performances	Tâches	Communication	Mémoire	Parallélisme
Temps	Optimiser le temps d'exécution des tâches	Minimiser les latences	- Minimiser le temps d'accès - Minimiser le temps d'attente	Favoriser le parallélisme
Surface	Optimiser la surface des supports d'exécution	Optimiser la surface des composants de communication		
Consommation	- Optimiser la consommation des ressources d'exécution - Diminuer la fréquence des ressources d'exécution	- Optimiser la consommation des ressources de communication - Diminuer la fréquence des ressources de communication	- Optimiser la consommation des mémoires	

Tableau 3 : Les paramètres influant sur les performances

Optimiser ces différents paramètres de l'architecture permet d'améliorer les performances de l'architecture. Afin d'analyser ces paramètres, nous proposons un modèle d'analyse qui permet de juger de la qualité d'optimisation du temps d'exécution des tâches, des mémoires, des communications et du parallélisme à haut niveau. Ci-après nous présentons comment nous considérons ces différents paramètres à haut niveau. Les métriques nécessaires et l'analyse seront détaillées dans le chapitre suivant.

Performances d'exécution des tâches

Une fois une tâche affectée à un PACM, elle aura deux possibilités d'exécution, soit en matériel, soit sur le processeur du PACM. Le processeur du PACM est choisi de façon à améliorer les performances de toutes les tâches du PACM. Une métrique d'analyse d'identité ***IA*** (*Identity Analysis)* a été établie dans le modèle d'analyse pour juger de la qualité d'optimisation de ce paramètre.

Les mémoires

Nous considérons la mémorisation à haut niveau comme la somme des mémorisations locales, de chaque tâche et des données transférées par les différents arcs à mémorisation (***Memorization 1***). Une métrique mémoire ***MEM*** a été établie dans le modèle d'analyse pour juger de la qualité d'optimisation de ce paramètre.

Les communications

Nous considérons les communications à haut niveau comme les différents transferts de données dans les PACM et entre les PACM. La quantité des données échangées est la somme

des quantités des données transportées par les arcs. Aussi, les différents *débits contraintes* de chaque PACM sont considérés comme élément influant sur les performances des communications. Plusieurs métriques sont établies (***EDE, DEIC, CIC et Tc***) dans le modèle d'analyse pour juger de la qualité d'optimisation de ce paramètre.

Le parallélisme

Le parallélisme est difficile à considérer sans savoir sur quel support d'exécution la tâche va être exécutée. Au niveau de la pré-exploration, nous ne considérons aucun ordonnancement des tâches et donc il est très difficile de savoir si une architecture a un parallélisme optimal ou non. Toutefois, le parallélisme est favorisé par les autres métriques du modèle.

6.3. EXPLORATION DE L'ESPACE DES PARTITIONS ET D'ARCHITECTURES

La valeur de chacune de ces métriques est un réel compris entre 0 et 1. Plus la partition optimise la propriété que la métrique analyse, plus cette métrique tend vers 1. Les 6 métriques constituent un espace des métriques (Figure 27). Plus les métriques couvrent cet espace, meilleure est la partition.

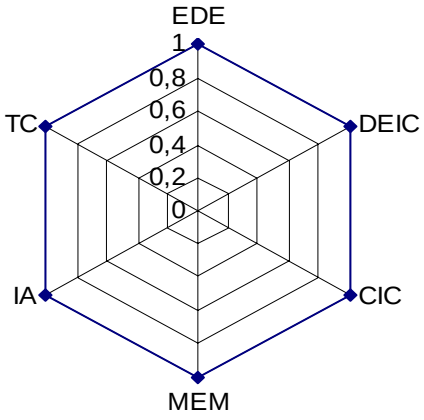


Figure 27 : Espace des métriques

La pré-exploration parcourt l'espace des partitions, analyse les différentes partitions en utilisant les métriques et trouve un ensemble de partitions qui optimise au mieux les performances des architectures. Nous utilisons un algorithme génétique multi-objectif pour l'exploration des partitions. Cet algorithme est détaillé dans le chapitre suivant.

Le flot d'exploration est le flot classique (Figure 28). Il consiste à estimer les performances des différentes tâches de l'application en matériel et sur les différents processeurs sur lesquels elles sont susceptibles d'êtres exécutées. Comme les tâches sont réparties dans des PACM, chaque tâche n'a que deux possibilités d'exécution, soit en matériel, soit par le processeur affecté au PACM. Donc, il y a au maximum deux performances à estimer pour chaque tâche. Par conséquent, la bibliothèque des performances du flot que nous proposons (en trait continu sur la Figure 28) est plus petite que celle pour les flots traditionnels (en trait discontinu sur la Figure 28). Ceci se traduit par une réduction importante du coût de l'exploration. Pour ces mêmes raisons, l'espace d'architectures à explorer est plus petit que celui des flots traditionnels. Par conséquent, la complexité de l'exploration est moins importante. Pour cette étape d'exploration, nous utilisons les méthodes de *codesign* (partitionnement, exploration d'architectures) traditionnelles existantes.

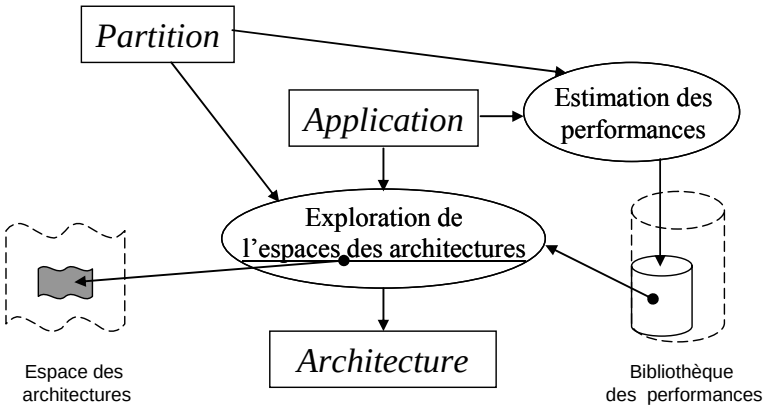


Figure 28 : Exploration de l'espace d'architectures

7. CONCLUSION

L'évolution des applications, de la technologie et les progrès réalisés dans le domaine de la conception des systèmes numériques ont augmenté le niveau de difficulté de conception de ces derniers. Aussi, les architectures ont évolué vers des architectures multiprocesseurs dont le nombre de processeurs ne cessera pas d'évoluer dans les années à venir.

Ces nouvelles architectures ont aussi engendré de nouveaux défis relatifs à la conception des systèmes numériques. En effet, le parallélisme est plus intense, le problème de la diversité logicielle a évolué en un problème assez complexe et la dimension spatiale a vu le jour compliquant le processus de conception.

Nous proposons un flot capable à la fois de :

- ✓ Tenir compte des défis engendrés par l'évolution des applications, de la technologie et des nouvelles architectures.
- ✓ Trouver une architecture qui satisfait les contraintes malgré la complexité d'exploration croissante.
- ✓ Respecter les coûts d'exploration.

Le flot sépare l'exploration en deux parties. Une partie d'exploration classique précédée par une étape de pré-exploration qui est à haut niveau et à moindre coût. Pour la pré-exploration haut niveau, nous collectons des informations dans un graphe de spécification à partir de la compréhension fonctionnelle de l'application, l'analyse des tâches et l'analyse du concepteur. La pré-exploration consiste à optimiser plusieurs critères significatifs (communication, mémoire, performance d'exécution des tâches et parallélisme). Différentes métriques sont établies pour mettre en œuvre l'analyse de ces critères. Un algorithme génétique est mis en œuvre pour assurer l'exploration. Cet algorithme et ces métriques sont détaillés dans le chapitre suivant.

IV. METRIQUES ET EXPLORATION

1. INTRODUCTION 69
2. METRIQUES 69
2.1. LES PERFORMANCES D'EXECUTION DES TACHES 70
2.2. LES MEMOIRES 74
2.3. LES COMMUNICATIONS 77
2.4. PARALLELISME 83
2.5. L'ESPACE DES METRIQUES 83
3. ALGORITHME GENETIQUE 84
3.1. DEFINITIONS 84
3.2. FORMULATION DU PROBLEME 84
3.3. L'OPTIMISATION MULTI-OBJECTIF 86
3.4. ALGORITHME GENETIQUE 87
4. OUTIL GAMA² ET DECISIONS DU CONCEPTEUR 92
4.1. L'OUTIL GAMA² 92
4.2. LA DECISION DU CONCEPTEUR 93
5. CONCLUSION 94

1. INTRODUCTION

Dans ce chapitre, nous nous focalisons sur l'étape de pré-exploration qui consiste à analyser l'espace des partitions pour aider le concepteur à choisir une partition adéquate pour l'application. Cette analyse a pour but d'optimiser plusieurs critères. Ces derniers sont les communications, la mémoire, les performances d'exécution des tâches et le parallélisme. L'optimisation de l'un de ces critères peut se faire au détriment d'autres critères. En résumé, le problème de la pré-exploration consiste à aider le concepteur à choisir la partition qui réalise le meilleur compromis entre les différents critères à optimiser.

"L'objet de l'optimisation multi-critère est d'aider le concepteur dans sa prise de décision à trouver les meilleures solutions de compromis" [Renaud 2006]. Nous définissons donc le problème de pré-exploration comme un problème d'optimisation multi-critère. Par définition, l'analyse multi-critère consiste à fournir au **décideur** des **outils** ou des **méthodes** lui permettant de progresser dans la résolution d'un problème de décision où **plusieurs critères**, souvent contradictoires, doivent être pris en compte. Ainsi, le problème d'analyse multi-critère se compose de 3 composants fondamentaux : les critères, les outils ou/et les méthodologies et le décideur.

Ci-après nous détaillons chacun de ces éléments. Nous étudions les différents critères et nous définissons des métriques pour les quantifier. Ensuite, nous présentons l'algorithme génétique multi-objectif pour aider le concepteur dans son analyse. Enfin, nous expliquons l'enjeu des décisions du concepteur afin d'optimiser les performances de l'architecture.

2. METRIQUES

Comme indiqué dans le chapitre précédent, à haut niveau, nous ne possédons ni informations sur les performances des différentes tâches, ni informations détaillées sur la structure de l'architecture. Afin de pouvoir évaluer les performances à haut niveau, nous avons étudié et proposé les principaux paramètres influant sur les performances de l'architecture. L'étude de ces performances (chapitre 2) a montré qu'il faut améliorer l'exécution des tâches, les communications, la mémoire et le parallélisme.

L'optimisation de ces différents critères permet d'améliorer les performances de l'architecture. Ils sont considérés différemment au niveau architecture et au niveau application (Tableau 4). Par exemple, les performances d'exécution d'une tâche au niveau architecture

sont considérées comme son temps, sa surface et sa consommation lors de son exécution par chaque support d'exécution. Au niveau application, ces performances sont évaluées par l'analyse du concepteur dans le vecteur d'analyse d'identité. Toutes les équivalences entre le niveau architecture et le niveau application sont détaillées dans le Tableau 4.

Nous nous basons sur cette analogie pour analyser les performances de l'architecture à un niveau assez haut "indépendamment" des caractéristiques technologiques de l'architecture. En effet, nous avons étudié les différents critères de l'architecture au niveau application et nous avons mis en œuvre six métriques pour évaluer et analyser les performances de l'architecture à un haut niveau "indépendamment" de la technologie.

Critères	Architecture	Application
Performance d'exécution des tâches	Temps, surface et consommation de la tâche pour un support d'exécution défini	Analyse d'identité d'une tâche
Mémoires	Taille et temps d'accès des mémoires	Estimation de la quantité des données mémorisées
Communication	- Architecture des différents supports de communication - Fréquence des différents supports de communication	- Estimation de la quantité des données échangées - Débit contrainte
Parallélisme	Ordonnancement des différentes tâches en fonction de leurs dépendances et en fonction de leurs supports d'exécution imposant le parallélisme entre deux tâches	Deux tâches dont les exécutions ne se succèdent pas peuvent s'exécuter en parallèle

Tableau 4 : Analogie architecture application

Nous détaillons dans la suite l'analyse des différents critères et nous présentons les différentes métriques proposées. Pour cela, nous considérons le problème de l'exploration de l'espace des architectures à la recherche d'une architecture à **n** processeurs qui satisfait les contraintes de l'application. Le concepteur a la possibilité d'implémenter **k** types de processeurs différents.

2.1. LES PERFORMANCES D'EXECUTION DES TACHES

Les performances d'exécution (Tableau 4) au niveau architecture sont le temps d'exécution, la surface et la consommation. Au niveau application, nous ne connaissons pas la ressource d'exécution de cette tâche. Pour juger de la qualité d'optimisation des différentes performances d'une tâche $\mathbf{t_i}$ par rapport aux différents supports d'exécution, nous nous basons sur le vecteur d'analyse $\boldsymbol{IA\_vector(t_i)}$. Ce vecteur est donné par le concepteur sur le graphe de spécification. Ci après, nous définissons le problème. Ensuite, nous analysons les partitions à haut niveau. Enfin, nous définissons le calcul de la métrique d'analyse d'identité **IA**.

2.1.1. Problématique

Les performances d'exécution d'une tâche dépendent de son support d'exécution. Les performances d'exécution de l'application dépendent de la performance d'exécution des différentes tâches et du parallélisme entre elles. Ce problème est accentué par le problème de diversité logicielle. Au niveau architecture, le problème que nous traitons est :

- ✓ Choisir le support d'exécution de chaque tâche parmi les **n** processeurs de l'architecture et l'implémentation matérielle afin d'améliorer les performances de chaque tâche tout en favorisant le parallélisme.
- ✓ Choisir le type de chacun des **n** processeurs parmi les **k** types possibles afin de permettre au plus grand nombre de tâches une exécution performante.
- ✓ Tenir compte de la position spatiale de chaque tâche afin de garantir la proximité entre les tâches qui s'échangent des données.

Ainsi, ce problème constitue un niveau de difficulté assez élevé. La résolution de ce problème est complexe. Au niveau application, nous cherchons la partition qui améliore le plus les performances de l'architecture. Nous rappelons qu'une tâche dans un PACM ne peut être exécutée que par le processeur affecté au PACM ou en matériel. Nous rappelons que deux tâches qui appartiennent au même PACM sont à proximité l'une de l'autre. Ainsi, au niveau application, le problème devient :

- ✓ Choisir le PACM de chaque tâche.
- ✓ Choisir le type du processeur affecté à chaque PACM.
- ✓ Choisir le support d'exécution de la tâche entre le processeur du PACM et une implémentation matérielle. Ceci se fait pendant l'exploration (partitionnement logiciel/matériel) et n'intervient pas pendant l'étape de pré-exploration.

Ainsi, le problème qui était résolu en une seule étape en faisant face à des complexités importantes est divisé en deux. Les deux premiers points sont résolus pendant la pré-exploration. Le troisième point est résolu pendant l'exploration. Ainsi, la complexité de ce problème pour l'étape de l'exploration est fortement diminuée ainsi que les coûts nécessaires à l'exploration. En effet, le choix du support d'exécution est généralement décidé en se basant sur des estimations ou des mesures des performances de l'exécution de chaque tâche. Avec une approche traditionnelle, il faut avoir les estimations/mesures des performances de chaque tâche pour les **k** différents types de processeurs plus les performances des **q** exécutions

matérielles. Grâce à l'approche que nous proposons, il suffit de disposer des performances de la tâche en cas d'exécution sur le processeur affecté au PACM et en cas d'une des **q** exécutions matérielles, ce qui est équivalent à **k-1** estimations/mesures de moins que pour les approches traditionnelles. Ci-après, nous présentons l'analyse des partitions afin d'optimiser les performances d'exécution des tâches.

2.1.2. Optimisation des performances d'exécution des tâches

La pré-exploration consiste à affecter chaque tâche à un des n PACM de l'architecture. Chaque tâche aura la possibilité d'exécution en matériel ou par le processeur affecté au PACM. Alors, les tâches d'un même PACM seront toutes exécutées par le processeur affecté au PACM ou en matériel. Il est donc logique que, pour optimiser les performances d'exécution des tâches, il faut que ce soit l'exécution par le processeur affecté au PACM, ou l'exécution matérielle, qui optimise le mieux les performances d'exécution de chacune des tâches du PACM.

Le vecteur d'analyse d'identité de chaque tâche correspond à un jugement sur la qualité des performances d'exécution de chaque tâche pour chacun des k types de processeurs. Donc pour améliorer les performances d'exécution des tâches, il faut regrouper les tâches qui ont les meilleures performances d'exécution avec le même processeur dans un même PACM. Ci-après, nous présentons la métrique d'analyse d'identité qui permet d'analyser l'effet des partitions des tâches sur leurs performances d'exécution.

2.1.3. Métrique d'analyse d'identité

Nous définissons l'ensemble des PACM d'une architecture **P** :

$$P = \{P_k, k \in [1, m]\}$$

P_k : un PACM de l'architecture.

m : nombre des PACM de l'architecture.

Afin d'analyser l'influence d'une partition sur les performances d'exécution d'une tâche, nous faisons la somme de tous les vecteurs d'analyse d'identité de toutes les tâches dans chaque PACM dans un LSIA (Local Sum of Identity Analysis) :

$$LSIA : P \rightarrow \Re^k$$

$$P_j \rightarrow LSIA(P_j) = \sum_{t_i \in P_i} IA\_vector(t_i) = \begin{pmatrix} LSIA\_PE(PE_1, P_j) \\ LSIA\_PE(PE_2, P_j) \\ . \\ . \\ . \\ LSIA\_PE(PE_k, P_j) \end{pmatrix}$$

$$LSIA\_PE : PE \times P \to \Re$$
$$(PE_k, P_j) \to LSIA\_PE(PE_k, P_j) = \sum_{t_i \in P_j} LIA(t_i, PE_k)$$

LSIA (P_j) est la somme locale dans un PACM P_j de toutes les identités d'analyse des tâches lui appartenant. Ce vecteur est calculé pour chaque PACM d'une partition. Il permet de désigner pour chaque PACM P_j le processeur qui lui convient le mieux $PE_{max}(P_j)$:

$$PE_{max} : P \to PE$$
$$Pj \to PE_{max}(P_j)$$
$$\forall i \in [1.k] \qquad LSIA\_PE(PE_{max}(P_j), P_j) \geq LSIA\_PE(PE_i, P_j)$$

Nous définissons **$nbt(P_j)$** le nombre de tâches dans le PACM P_j :

$$nbt : P \to \aleph$$
$$Pj \to nbt(P_j)$$

Considérons l'exemple de 3 tâches t_1, t_2 et t_3 dont les vecteurs d'analyse d'identité respectifs sont (0,9 0,4), (0,9 0,3) et (0,5 0,8) pour deux ressources d'exécution PE_1 et PE_2. Considérons que l'architecture contient deux PACM, le premier contient les tâches t_1 et t_3, le second contient la tâche t_2. LSIA du PACM P1 est :

$$LSIA(P_j) = \begin{pmatrix} 0,9+0,5 \\ 0,4+0,8 \end{pmatrix} = \begin{pmatrix} 1,4 \\ 1,2 \end{pmatrix}$$

PE_1 est donc le $PE_{max}(P_1)$ puisqu'il a la plus grande valeur de LSIA_PE.

Une fois le type de processeur $PE_{max}(P_j)$ le plus approprié au PACM P_j identifié et son identité d'analyse $LSIA\_PE(PE_{max},P_j)$ calculée, nous comparons P_j avec un PACM idéal $P_{idéal}$. Sachant qu'un processeur de type $PE_{max}(P_j)$ est affecté à un PACM P_j, l'idéal pour améliorer les performances d'exécution des tâches est d'affecter à ce PACM les $nbt(P_j)$ tâches qui présentent les meilleures performances en s'exécutant par le processeur de type PE_{max}. Ces nbt tâches t_i sont celles qui ont les $LIA(t_i,PE_{max}(P_j))$ les plus élevées. Pour $P_{idéal}$ nous calculons son $LSIA\_PE(PE_{max}(P_j),P_{idéal})$ que nous nommons $MLSIA(PE_{max}(P_j), nbt(P_j))$.

$$MLSIA : PE \times \aleph \to \Re$$
$$(PE_{max}(P_j), nbt) \to MLSIA(PE_{max}, nbt)$$

Pour l'exemple considéré, le PACM P_1 contient 2 tâches. Donc $nbt(P_1)$ est égale à deux. Aussi PE_1 est le $PE_{max}(P_1)$ de P_1. Alors, $P_{idéal}$ est celui qui contient les deux tâches qui ont le plus grand $LIA(t,PE_1)$. Dans notre cas, il s'agit de t_1 et t_2 d'où $MLSIA(PE_1,2)$ est égal à 1,8. Ensuite, nous calculons une métrique d'analyse d'identité ***$MLIA(P_j)$*** locale au PACM P_j :

$$LIAP : P \to \Re$$
$$P_j \to LIAP(P_j) = \frac{LSIA\_PE(PE_{max}(P_j), P_j)}{MLSIA(PE_{max}(P_j), nbt(P_j))}$$

LIAP permet de juger de la capacité du PACM P_j à améliorer les performances d'exécution des tâches. Plus LIAP(P_j) s'approche de 1, plus le PACM Pj est proche du PACM idéal et plus il améliore les performances d'exécution. Enfin, pour juger de la qualité d'optimisation des performances d'exécution des tâches de toute la partition, nous définissons une métrique d'analyse d'identité globale **IA** (Identity Analysis) :

$$IA = \min_{P_j \in P} (LIAP(P_j))$$

La métrique d'analyse d'identité IA est le minimum de toutes les métriques locales LIA(p) de la partition. Ceci permet de garantir une bonne répartition des tâches en augmentant les performances d'exécution des tâches dans tous les PACM de la partition.

En résumé, la métrique IA est un réel entre 0 et 1 qui permet de juger de la qualité des performances d'exécution des tâches de toute l'application. Plus IA est proche de 1, plus les performances sont optimisées.

2.2. LES MEMOIRES

Les mémoires au niveau architecture sont caractérisées par leurs temps d'accès et leurs tailles. Au niveau application, nous estimons la quantité des données mémorisées en fonction de l'analyse du concepteur (la variable mémorisation sur les arcs) et l'analyse des tâches (ressources de mémorisation locale sur les nœuds). Nous analysons la distribution des mémoires à haut niveau (dans les partitions) afin de juger de la qualité de la distribution mémoire et son potentiel à optimiser les performances de l'architecture. Ci-après, nous définissons la métrique mémoire **MEM**.

2.2.1. Problématique

Les mémoires sont des éléments importants de l'architecture. Ils influent sur les performances de l'architecture selon plusieurs points :

- ✓ Le temps d'accès de la mémoire influe sur le temps d'exécution total de l'application et sur la consommation.
- ✓ La taille de la mémoire influe sur la surface et la consommation. En effet, plus la mémoire est grande, plus la densité du trafic est importante et le temps d'accès est lent et donc la consommation est plus importante.

- ✓ La disponibilité : le parallélisme des applications modernes est traduit par l'exécution parallèle d'un nombre croissant de tâches. Ces tâches nécessitent des accès simultanés à la/aux mémoire(s). Ainsi, si les tâches doivent se succéder pour l'accès aux mémoires, le parallélisme perd son intérêt. Ainsi, il doit y avoir une architecture mémoire qui favorise un accès parallèle pour les tâches qui s'exécutent en parallèle.

Le problème du choix de l'architecture mémoire doit concilier taille, temps d'accès et disponibilité pour le parallélisme. Toutefois, il ne faut pas que l'architecture comporte un nombre infini de mémoires, sinon la surface et la consommation augmenteraient considérablement.

2.2.2. Optimisation de la distribution de la mémoire

La pré-exploration génère une partition des tâches sur les différents PACM. Ainsi toutes les données mémorisées sont distribuées sur les PACM de la partition. Dans chaque PACM une hiérarchie mémoire sera établie pendant l'étape d'exploration. Comme cela est expliqué pendant l'analyse du parallélisme, nous favorisons le parallélisme en distribuant les tâches qui risquent de s'exécuter en parallèle dans des PACM différents. Ainsi, les tâches qui s'exécutent en parallèle ont peu de chance de se partager les ressources de mémoires du fait que chaque PACM dispose de ses propres ressources de mémoires. L'architecture mémoire de chaque PACM est raffinée pendant l'exploration. Toutefois, étant donné que nous avons distribué le parallélisme, l'exploration de l'architecture mémoire a moins de complexité pour résoudre la disponibilité pour le parallélisme. Il reste encore à gérer la taille et le temps d'accès des mémoires utilisées.

Plus les tailles de mémoires sont grandes, plus les temps d'accès sont longs. Afin de minimiser la taille des mémoires utilisées tout en veillant à ne pas augmenter considérablement leur nombre, nous distribuons les données mémorisées d'une façon homogène entre les différents PACM. En effet, si nous distribuons d'une façon non symétrique les mémorisations (exemple Figure 29.b), il y aura certes une petite quantité de données mémorisées dans l'un des PACM (PACM 4) qui aura alors une petite taille et un temps d'accès assez optimisé. Toutefois, il y aura aussi un PACM où il y aura une grande concentration de données à mémoriser (PACM 1) où la taille des ressources de mémoires sera importante et par conséquent le temps d'accès élevé. Donc, si nous distribuons de façon équilibrée les données à mémoriser (exemple Figure 29.a), la plus grande concentration de ces données sera la plus petite possible, soit toutes les données à mémoriser divisées par le nombre de PACM.

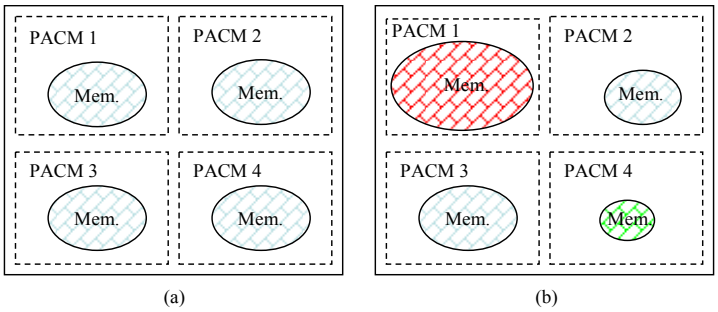


Figure 29 : Distribution des données mémorisées

2.2.3. Métrique mémoire

Nous considérons que la quantité des données à mémoriser dans un PACM P_k (**Mem_ressources(P_k)**) est la somme des données échangées à mémoriser **Shared_mem(P_k)** (*Shared memory*) plus la somme des mémoires locales **LMP_size(P_k)** :

$$\text{Mem\_ressources} : P \rightarrow \aleph$$
$$P_k \rightarrow \text{Mem\_ressource}(P_k) = \text{Shared\_mem}(P_k) + \text{LMP\_size}(P_k)$$

$$\text{LMP\_size} : P \rightarrow \aleph$$
$$P_k \rightarrow \text{LMP\_size}(P_k) = \sum_{t_i \in P_k} \text{LM\_size}(t_i)$$

$$\text{Shared\_mem} : P \rightarrow \aleph$$
$$P_k \rightarrow \text{Shared\_mem}(P_k) = \sum_{\substack{t_i \in T \\ t_j \in P_k}} \text{memorisation}(t_i, t_j) \times \text{data}(t_i, t_j)$$

$$\text{Data} : T \times T \rightarrow \aleph$$
$$(t_i, t_j) : \text{Data}(t_i, t_j) = \text{Word\_nb}(t_i, t_j) \times \text{word\_size}(t_i, t_j) \times \text{Nb\_of\_execution}(t_i, t_j) \times \text{execution\_frequency}(t_i)$$

Si une tâche t_j appartenant à un PACM P_k recevant un arc **(t_i, t_j)** mémorise une quantité de données **Data(t_i, t_j)**, cette quantité de données est considérée comme stockée par les ressources de mémorisation de P_k.

Après avoir calculé la quantité de données à mémoriser dans les PACM, nous calculons une métrique locale de mémoire ***Mem_degree(p)*** qui est le pourcentage des données mémorisées dans le PACM par rapport à toutes les données mémorisées dans toute l'application **G_Mem_ressource** (global memory ressources) :

$$\text{Mem\_degree} : P \rightarrow \Re$$
$$P_k : \text{Mem\_degree}(P_k) = \frac{\text{Mem\_ressource}(P_k)}{\text{G\_Mem\_ressource}}$$

La métrique locale Mem_degree(p) permet de quantifier la proportion des données à mémoriser dans chaque PACM par rapport à toutes les données à mémoriser. L'idéal est que ce soit la même proportion dans tous les PACM. Alors, nous calculons une métrique globale ***MEM*** pour vérifier si la distribution des mémoires s'approche du cas idéal ou non :

$$Mem = \frac{\underset{P_i \in P}{Min}\ (Mem\_degree(P_i))}{\underset{P_j \in P}{Max}\ (Mem\_degree(P_j))}$$

Ainsi, si la proportion de données mémorisées est la même partout, **Mem** est égal à 1. Mais, si toutes les données à mémoriser sont concentrées dans le même PACM la métrique sera égale à 0. Cette métrique analyse la distribution de la quantité des données mémorisées et veille à ce que cette distribution soit équilibrée.

2.3. LES COMMUNICATIONS

La communication au niveau architecture est un point clé. En effet, en plus d'influer sur l'architecture en temps, en surface et consommation, elle correspond à une étape assez complexe de la conception des systèmes embarqués. Au niveau application, elle se manifeste comme la quantité de données échangées par les tâches de l'application et aussi comme les débits de transfert des tâches de l'application. Ci-après, nous définissons le problème des communications. Ensuite, nous analysons la distribution des données dans les partitions afin d'optimiser les communications en termes de performance et de complexité de conception. Enfin, nous définissons le calcul des métriques des communications : EDE, CIC, DEIC et Tc**.**

2.3.1. Problématique

La communication a toujours été un goulet d'étranglement lors de la conception des systèmes embarqués. Pour les architectures modernes, le degré de parallélisme important et la quantité croissante des données échangées entre les différents composants de l'architecture ont accentué la complexité de la conception des communications.

En effet, prenons l'exemple d'une architecture moderne (Figure 30). Cette architecture contient un nombre important de composants : 6 processeurs, 8 mémoires, 7 composants matériels et 3 coprocesseurs, soit 24 composants. Le concepteur doit trouver l'architecture qui permet d'assurer la communication entre ces composants et assurer l'échange de 12,5 Mbits de données. Aussi, il doit procéder à la définition d'une position pour chaque composant afin

d'optimiser les communications. Bien que les composants de l'architecture soient définis, la conception de la communication s'avère une opération complexe.

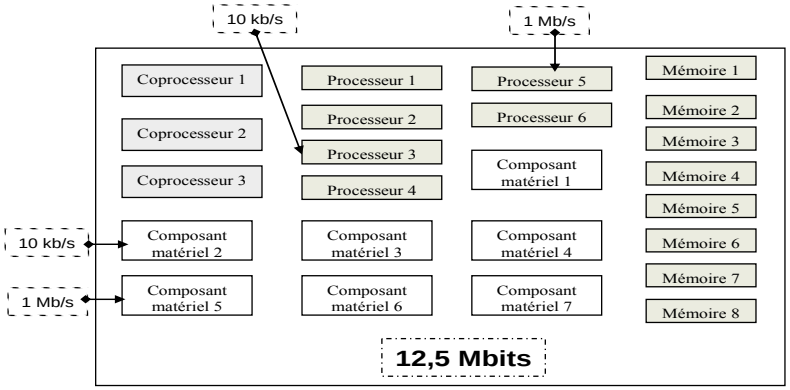


Figure 30 : Exemple d'architecture multiprocesseur

En outre, certains composants peuvent être contraints d'effectuer le transfert des données à des débits différents, ce qui veut dire que le concepteur doit choisir des ressources de communication qui soient assez rapides pour convenir à tous les débits de transfert et pas trop rapides pour consommer moins. Aussi, les ressources de communication doivent d'un côté se munir de protocoles et de contrôleurs suffisamment efficaces pour gérer les différents échanges de données et d'un autre côté avoir le minimum de contrôleurs possible pour consommer moins de surface et moins d'énergie.

Bref, au niveau architecture, la conception de la communication est une étape très compliquée devant gérer de nombreux compromis.

2.3.2. Optimisation des communications

Si nous regroupons les tâches ayant les plus grandes quantités de transfert de données dans un seul PACM de l'application partageant les mêmes ressources de communication, cela engendre une complexité pour gérer les communications et donc pour concevoir les supports et les ressources nécessaires sur ce PACM. Aussi, la concentration des communications en un seul PACM crée une dissymétrie entre les différents groupes de tâches, ce qui engendre un coût supplémentaire au niveau du temps de conception pour gérer cette complexité, au niveau de la surface, du temps et de la consommation. En effet, un flot de communication important nécessite un temps de conception plus important et des ressources de transfert, d'interfaçage, de synchronisation et de mémorisation également plus importantes.

Minimiser la quantité de données échangées et le nombre d'échanges entre les groupes de tâches se partageant une même ressource de communication améliore les performances du système. En effet, la communication entre deux tâches de deux PACM différents monopolise les ressources de communication de chacun des PACM et les moyens de communication entre

eux. Ceci engendre le blocage de tout transfert de données utilisant les ressources de communication des deux groupes et aussi tout transfert de données entre PACM utilisant les mêmes ressources. Cela peut représenter une nette dégradation des performances en temps.

La communication se manifeste aussi comme les débits de transfert des tâches de l'application. Généralement, les applications ont quelques contraintes de temps à respecter. En général, elles concernent les tâches d'entrée et de sortie, ainsi que quelques autres tâches de l'application. Ces contraintes se traduisent par un débit de transfert de données à respecter. Ces débits représentent le plus petit débit que les supports de communication doivent avoir pour acheminer les données selon les contraintes. Donc, si une tâche du même PACM (utilisant les mêmes ressources de communication) a un débit différent (inférieur ou supérieur), les ressources de communication doivent respecter au moins le plus grand débit des deux. En conséquence, ces ressources de communication sont surdimensionnées au moins pour l'une des deux tâches ce qui engendre un coût plus important en temps, surface, consommation (fréquence) et complexité.

En résumé, pour optimiser les performances et la complexité des communications dans l'architecture, nous veillons à :

- ✓ Distribuer de façon équilibrée les données échangées dans les PACM afin de minimiser les quantités de données échangées dans chaque PACM.
- ✓ Minimiser le nombre et la quantité des échanges de données entre les PACM de l'architecture.
- ✓ Ne pas mettre dans le même PACM des tâches qui ont des *débits contraintes* trop différents.

Par conséquent, pour l'exemple de la Figure 30, l'idéal est de trouver une partition qui distribue les données de façon équilibrée (Figure 31). Ceci implique que, pour le concepteur au lieu de concevoir une architecture de communication capable de gérer l'échange de 12,5 Mbits de données, il n'a plus qu'à concevoir sept architectures de communication pour des échanges de données dont le plus important transfert est de 2 Mbits. Le fait de minimiser le nombre et la quantité des données échangées entre PACM implique que les tâches qui communiquent entre elles soient dans le même PACM. Ceci implique de veiller à ce que deux tâches qui communiquent entre elles soient à proximité l'une de l'autre selon la définition énoncée dans le chapitre précédent. Regrouper les tâches qui ont des contraintes proches dans les mêmes PACM (Figure 31) permet d'éviter d'avoir à gérer des débits de 10 kbits/s et 1 Mbits/s en même temps.

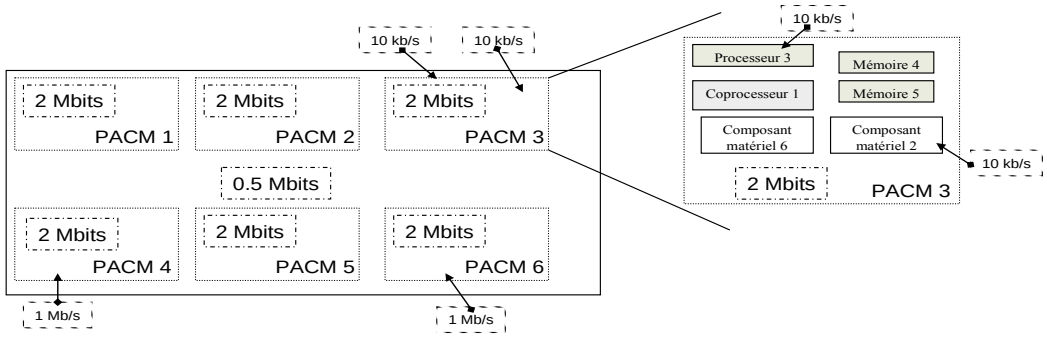


Figure 31 : Exemple de distribution des données

2.3.3. Métriques de communication

Dans le cadre de l'analyse de la communication, nous définissons quatre métriques. Ces métriques ont pour but de prévoir l'équilibre des échanges de données entre les différents PACM et l'équilibre de la distribution des débits de certaines tâches dans les PACM. La métrique d'équilibre d'échange des données ***EDE*** (*Equilibrate Data Exchange*) analyse la distribution des données dans chaque PACM. Les deux métriques d'équilibre des échanges des données entre PACM ***DEIC*** (*Data Exchange Inter Cluster*) et ***CIC*** (*Connection Inter Cluster*) analysent les échanges de données entre les PACM. Enfin, la métrique des *débits contraintes* **Tc** (Throughput constraint) analyse la distribution des différents débits dans le PACM.

2.3.3.1. Métrique d'équilibre des échanges des données EDE

Afin d'analyser les échanges de données dans les PACM, nous calculons la quantité des données échangées dans chaque PACM **Data_exchange(P_k)**

$$\text{Data\_exchange} : P \to \aleph$$
$$P_k \to \text{Data\_exchange}(P_k) = \sum_{(t_i,t_j)\in P_k \times P_k} \text{Data}(t_i, t_j)$$

Afin de positionner la quantité de données échangées dans un PACM P_k par rapport à la quantité globale de données échangées dans toute l'application **G_Data_exchange,** nous calculons une métrique locale **Com_degree** :

$$\text{Com\_degree} : P \to \Re$$
$$P_k \to \text{Com\_degree}(P_k) = \frac{\text{Data\_exchange}(P_k)}{\text{G\_Data\_exchange}}$$

$$\text{G\_Data\_exchange} = \sum_{(t_i,t_j)\in T\times T} \text{Data}(t_i, t_j)$$

La métrique locale Com_degree(P_k) permet de quantifier la proportion des données à échanger dans chaque PACM par rapport à toutes les données échangées. L'idéal est que la même proportion soit dans tous les PACM. Nous calculons alors une métrique globale ***EDE*** pour vérifier si la distribution des échanges des données s'approche du cas idéal ou non :

$$EDE = \frac{\underset{P_i \in P}{Min}\ (Com\_degree(P_i))}{\underset{P_j \in P}{Max}\ (Com\_degree(P_j))}$$

Si toutes les quantités des données échangées sont les mêmes partout EDE est égale à 1. Si toutes les données à échanger sont concentrées dans le même PACM, la métrique sera égale à 0. Cette métrique analyse la distribution de la quantité des données échangées dans les PACM et veille à ce que cette distribution soit équilibrée.

2.3.3.2. Métriques d'échange des données entre les PACM

Afin d'analyser les échanges des données entre les PACM, nous calculons la quantité des données échangées **Data_exchange_IC** (*Data exchange Inter Cluster*)

$$Data\_exchange\_IC = \sum_{\substack{(P_n, P_m) \in P \times P \\ P_n \neq P_m}} \ \sum_{\substack{t_i \in P_n \\ t_j \in P_m}} Data(t_i, t_j)$$

Nous définissons ensuite une métrique ***DEIC*** pour analyser les échanges des données entre les PACM et juger si la minimisation de cette quantité de données est suffisante.

$$DEIC = 1 - \frac{Data\_exchange\_IC}{G\_Data\_exchange}$$

Cette métrique correspond au pourcentage de données échangées entre les groupes de l'architecture (Data_exchange_IC) par rapport à la quantité totale de données échangées dans toute l'application. Plus la quantité de données échangées entre les groupes est importante plus DEIC est proche de 0. Inversement, plus cette quantité est faible, plus DEIC est proche de 1.

Nous calculons aussi le nombre de connexions entre les PACM **Connexion_IC** (Connexions Inter Cluster) et le nombre de connexions globales **G_connexions** (Global connexions). Une connexion est l'arc qui relie les tâches. **Connexion_IC** est le nombre d'arcs dont la tâche émettrice et la tâche réceptrice n'appartiennent pas au même PACM.

G_connexions est le nombre d'arcs de l'application. Nous définissons la métrique de connexion inter cluster ***CIC*** (*connexion inter cluster*) :

$$CIC = 1 - \frac{Connexion\_IC}{G\_connexions}$$

Il s'agit du pourcentage des connexions entre les groupes de l'architecture par rapport à toutes les connexions de l'application. Cette métrique a pour but d'évaluer la minimisation des connexions entre les groupes de l'architecture. Par conséquent, plus le nombre de connexions entre les groupes est petit, plus **CIC** est proche de 1 et inversement.

2.3.3.3. Métrique de débits contraintes

Afin d'analyser la distribution des débits contraintes entre les PACM, nous calculons d'abord la différence entre le plus grand débit contrainte et le plus petit dans chaque PACM P_k Diff_throughput(P_k) et dans toute l'architecture Diff_throughput_GS.

$$Diff\_throughput : P \to \aleph$$
$$P_k \to Diff\_throughput(P_k) = \max_{t_i \in P_k}(Thc(t_i)) - \min_{t_j \in P_k}(Thc(t_j))$$

$$Diff\_throughput\_GS = \max_{t_i \in T}(Thc(t_i)) - \min_{t_j \in T}(Thc(t_j))$$

Ensuite pour chaque PACM P_k, nous calculons le throughput_degree(P_k) qui représente une division de la différence du débit dans un PACM par la différence dans toute l'application. Supposons, par exemple, que dans une application le plus grand débit est de 1 Mbits/s et le plus petit est de 10 kbits/s, la différence est alors de 990 kbits/s. Si dans un PACM P_i, nous avons deux débits de 100 kbits/s et de 120 kbit/s, throughput_degree(P_i) est petit (20/990). Et donc, nous avons réussi dans ce PACM à réduire les différences de débit qui existent dans toute l'application. Si dans un PACM P_j, il y a un débit de 800 kbits/s et 50 kbit/s, alors throughput_degree(P_j) est proche de 1 (750/990).

$$Throughput\_degree : P \to \Re$$
$$P_k \to Throughput\_degree(P_k) = \frac{Diff\_throughput(P_k)}{Diff\_throughput\_GS}$$

Enfin, nous calculons la métrique débits contraintes **Tc** pour s'assurer que dans un groupe il y aura que des débits proches :

$$Tc = \min_{P_k \in P}(throughput\_degree(P_k))$$

2.4. PARALLELISME

Au niveau architecture, le parallélisme est considéré lors de l'ordonnancement des différentes tâches de l'application. L'ordonnancement consiste à établir un ordre d'exécution des tâches pour améliorer le temps. L'ordonnancement peut favoriser le parallélisme entre les tâches. Malheureusement, à ce niveau les possibilités de parallélisme sont limitées. En effet, comme le support d'exécution des tâches est défini, il suffit que deux tâches soient exécutées par le même processeur pour qu'il soit impossible de les exécuter en parallèle.

Ce problème se pose parce que le parallélisme est traité tardivement dans le flot d'exploration. Nous proposons de favoriser le parallélisme à haut niveau afin que, plus tard dans le flot, il y ait le plus de parallélismes possibles. Ainsi l'ordonnancement aura plus de possibilités d'optimiser les performances.

Pour favoriser le parallélisme, nous nous basons sur le lemme énoncé lors de la définition des architectures PACM. Deux tâches qui sont dans deux PACM différents ont la possibilité d'être exécutées en parallèle. Donc, pour favoriser le parallélisme, il faut favoriser le fait que deux tâches qui ne communiquent pas ensemble sont dans deux PACM différents.

Nous n'avons pas défini une métrique parallélisme. Les métriques de communications veillent à ce que les tâches qui communiquent ensemble soient dans le même PACM et qu'il y ait peu de tâches qui communiquent entre elles dans des PACM différents. Ainsi les métriques de communication veillent à favoriser le parallélisme.

2.5. L'ESPACE DES METRIQUES

Les métriques définissent un espace (Figure 32). Plus la partition couvre cet espace, plus elle optimise les différents critères suivants : les communications, les mémoires, les performances d'exécution et le parallélisme.

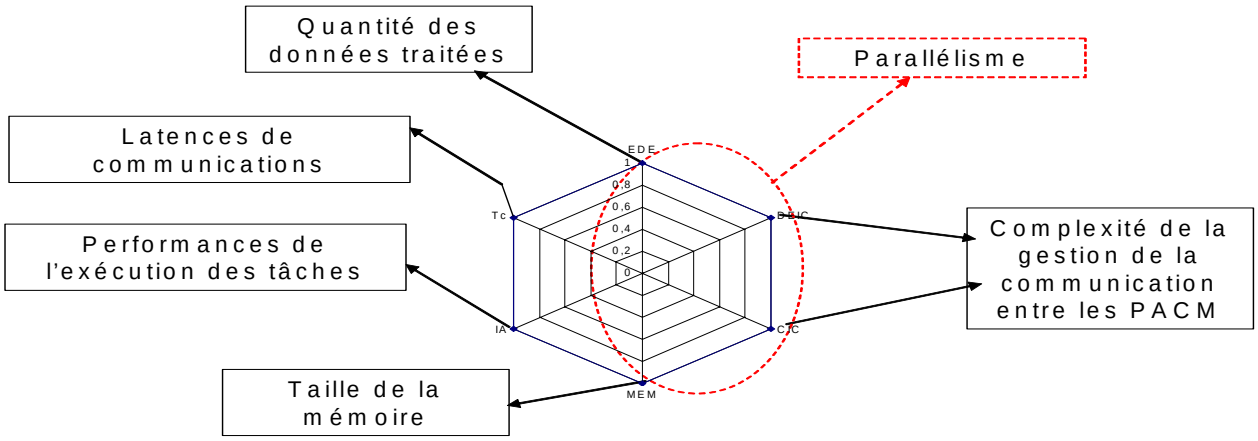


Figure 32 : Influence des métriques

3. ALGORITHME GENETIQUE

Nous détaillons dans cette section l'outil d'exploration. Ci-après, nous énonçons plusieurs définitions pour introduire quelques notions de l'optimisation multi-objectif. Nous étudions ensuite le problème d'optimisation multi-objectif de la pré-exploration. Après, nous étudions les algorithmes d'optimisation et nous expliquons le choix de l'algorithme génétique pour résoudre ce problème. Ensuite, nous détaillons l'algorithme génétique que nous avons utilisé.

3.1. DEFINITIONS

Nous énonçons ci-dessous les définitions des notions que nous utilisons par la suite. Les solutions que nous traitons intègrent plusieurs critères (les métriques). Afin de pouvoir les comparer, nous définissons la notion de dominance. Si une solution A domine une solution B, alors A est meilleure que B.

Définition 1 : Domination

Une solution A est dominée par une solution B si :
- Chaque métrique de la solution B est supérieure ou égale à la métrique de la solution A.
- Il existe une métrique de la solution B qui est strictement supérieure à la métrique de la solution A.

En termes mathématiques :

Une solution A est dominée par une solution B si :
$\forall$ i $\in$ [1,n] Métrique.i (B) $\geq$ Métrique.i (A)
si $\exists$ i $\in$ [1,n] Métrique.i (B) > Métrique.i (A)

Les solutions qui dominent les autres mais ne se dominent pas entre elles sont appelées solution optimale au sens de Pareto [Pareto1896].

Définition 2 : Optimalité au sens du Pareto

Une solution A est optimale au sens de Pareto (ou optimale globalement au sens de pareto) s'il n'existe pas de solution B telle que B domine A.

3.2. FORMULATION DU PROBLEME

Nous considérons le problème de la répartition des tâches de l'application sur m PACM. Soit **T** l'ensemble des tâches de l'application :

$$T = \{t_i, i \in [1, n]\}$$

t_i : une tâche de l'application.

n : nombre des tâches de l'application.

Soit **P** l'ensemble des PACM

$$P = \{P_i, i \in [1, m]\}$$

P_i : un PACM de l'application.

m : nombre des PACM de l'application.

$m<n$

Un vecteur $\vec{x} \in (T \times P)^n$ est une partition des n tâches de l'application dans les m PACM

$$\vec{x} = \begin{pmatrix} (t_0, P_{k_0}) \\ (t_1, P_{k_1}) \\ . \\ (t_i, P_{k_i}) \\ . \\ (t_n, P_{k_n}) \end{pmatrix};$$

Avec

$\forall i \in [1, n]$,

$t_i \in T$;
$k_i \in [1, m]$;
$P_{k_i} \in P$;

Soit $\vec{f}$ la fonction d'évaluation d'une partition $\vec{x}$. Elle correspond aux valeurs des 6 métriques calculées pour cette partition.

$\forall \vec{x} \in (T \times P)^n$

$$\vec{f}(\vec{x}) = \begin{pmatrix} EDE(\vec{x}) \\ DEIC(\vec{x}) \\ CIC(\vec{x}) \\ IA(\vec{x}) \\ MEM(\vec{x}) \\ TC(\vec{x}) \end{pmatrix}$$

Problème d'optimisation multi-objectif :

Maximiser $\vec{f}(\vec{x})$

Résolution

Trouver les $\vec{x} \in (T \times P)^n$ ***tel que*** $\vec{x}$ **est optimal globalement au sens de Pareto**

Le problème de pré-exploration est un problème d'optimisation combinatoire NP complet. Il existe une large classe d'algorithmes pour la résolution de ces problèmes. Ci-après, nous faisons un résumé de l'optimisation multi-objectif.

3.3. L'OPTIMISATION MULTI-OBJECTIF

Les problèmes d'optimisation issus de problématiques réelles sont la plupart du temps de nature multi-objectif car plusieurs critères sont à considérer simultanément. Optimiser un tel problème relève donc de l'optimisation combinatoire multi-objectif. Les premières études concernant l'optimisation combinatoire multi-objectif transformaient les problèmes multi-objectifs en une succession de problèmes mono-objectifs. Pour cela, un ordre d'importance sur les objectifs pouvait être donné, et l'optimisation consistait à optimiser un objectif sans dégrader les valeurs déjà obtenues pour les objectifs plus prioritaires. Une autre approche consistait en l'optimisation d'une agrégation linéaire des objectifs, chacun pouvant avoir un poids représentant son importance. Lorsque nous nous trouvons dans un réel contexte multi-objectif, il n'est pas toujours possible de trouver un ordre d'importance sur les critères. Il est alors nécessaire de rechercher les solutions présentant le meilleur compromis entre les objectifs.

Dans notre cas, aucun des objectifs n'est prioritaire par rapport aux autres. Nous cherchons les solutions qui optimisent le mieux les différents objectifs à la fois.

La Figure 33 [Dréo 2003] résume une classification des différents algorithmes d'optimisation. Notre problème est un problème d'optimisation combinatoire. Le problème est un problème NP-complet, ce qui implique qu'il est considéré comme un problème d'optimisation difficile. Donc, il nous faut une méthode approchée. Ceci implique que pour résoudre notre problème, il nous faut une métaheuristique.

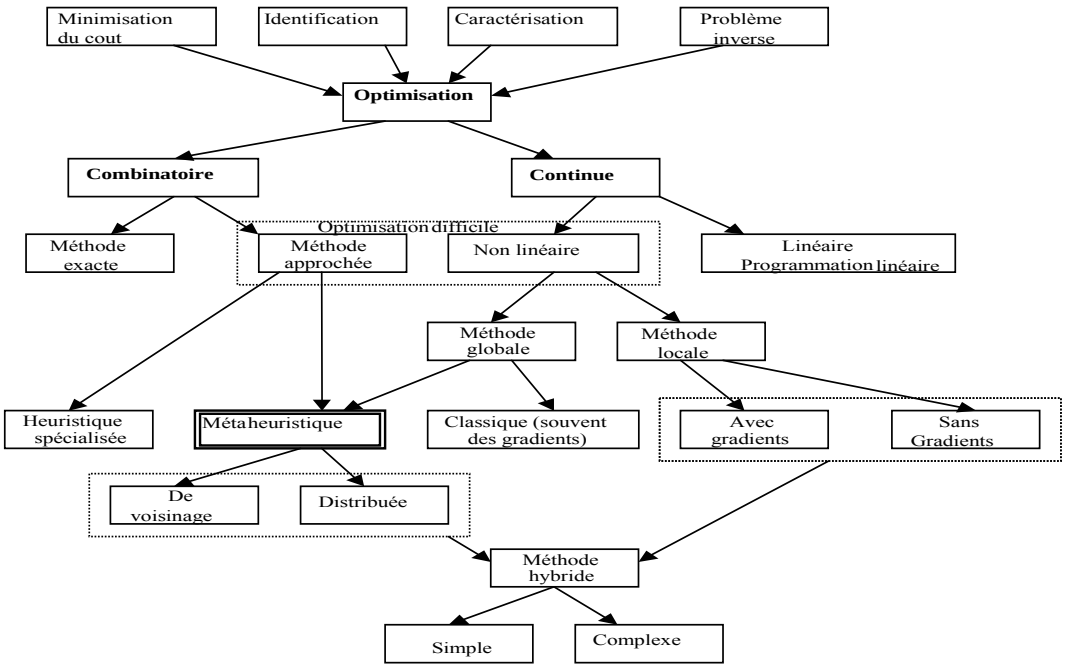


Figure 33 : Classification générale des méthodes d'optimisation

Les métaheuristiques sont des méthodes générales de recherche dédiées aux problèmes d'optimisation difficile [Dréo 2003]. Ces méthodes, en général, reprennent des idées parfois présentes dans la vie courante comme la colonie de fourmis ou le recuit simulé. Les principales métaheuristiques sont le recuit simulé, la recherche tabou et les algorithmes génétiques.

Pour notre problème, nous avons besoin d'une métaheuristique capable d'assurer :

- ✓ La résolution d'un problème avec un nombre d'objectifs important.
- ✓ La flexibilité du choix et des nombres d'objectifs.

Dans le cadre de notre travail, nous choisissons d'utiliser un algorithme génétique multi-objectif. Cet algorithme est détaillé ci-après.

3.4. ALGORITHME GENETIQUE

L'algorithme génétique se base sur la transmission de gênes forts d'une génération à une autre et la disparition au fur et à mesure du temps des gênes faibles. Il s'agit de croiser les membres forts d'une population afin de ne trouver, après plusieurs générations, que les membres forts.

En termes mathématiques, l'algorithme génétique est une méthode qui sert à trouver le minimum absolu d'une fonction ou d'un ensemble. L'algorithme consiste à explorer un ensemble de solutions initiales et à comparer les différentes solutions. Ensuite il s'agit de combiner les meilleures solutions (les minimums de l'ensemble initial) pour créer un nouvel ensemble de solutions qui converge le plus vers le minimum global. Il s'agit ensuite de réitérer ces opérations plusieurs fois jusqu'à ce que l'ensemble des solutions ne contienne que le minimum global.

La boucle générationnelle de l'algorithme génétique multi-objectif (Figure 34) que nous utilisons est comme suit :

i. Initialisation de la population : choisir n individus pour constituer la population de départ.
ii. Evaluation des n individus de la population.
iii. Sélection de n individus pour la reproduction.
iv. Croisement des n-λ individus sélectionnés.
v. Mutation des λ individus sélectionnés.

vi. Si la population générée a convergé suffisamment, nous arrêtons, sinon nous reprenons en (ii).

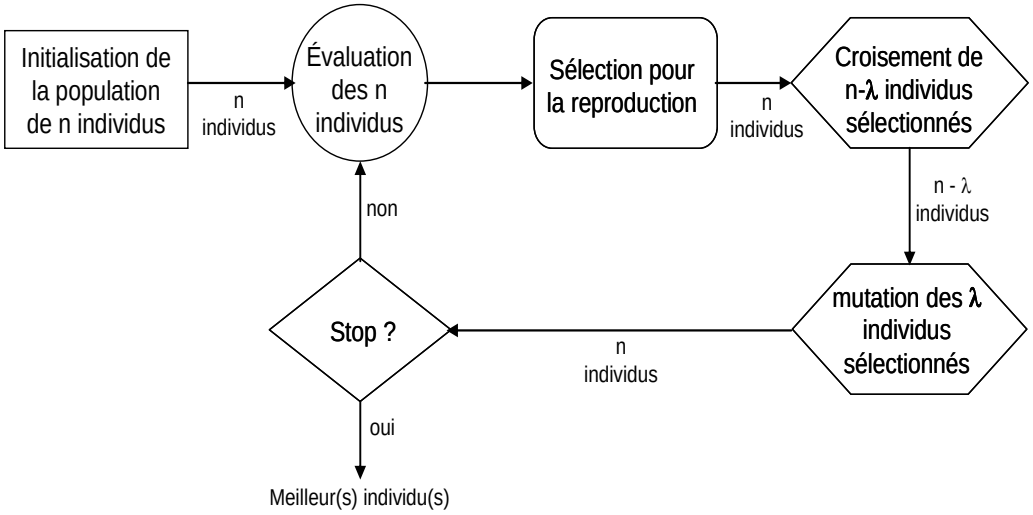


Figure 34 : Boucle générationnelle de l'algorithme génétique

3.4.1. Population

Une population est un ensemble d'individus. La population évolue durant une succession d'itérations appelées générations jusqu'à ce qu'un critère d'arrêt, qui prend en compte a priori la qualité des solutions obtenues, soit vérifié. Durant chaque génération, une succession d'opérateurs est appliquée aux individus d'une population pour engendrer la nouvelle population à la génération suivante.

Un individu est une solution potentielle, plus ou moins performante, à un problème posé. Ces solutions appartiennent à l'espace de recherche du problème d'optimisation. Pour le problème que nous traitons, un individu est une partition des tâches sur les différents PACM. L'individu est représenté par un vecteur (comme celui de la Figure 35.a). La Figure 35 illustre la représentation d'un exemple de classification d'une application en deux PACM. Le vecteur, représentant l'individu, illustre l'affectation de chaque tâche à un des PACM. Le chiffre représente le numéro du PACM. La position du chiffre dans le vecteur représente le numéro de la tâche. Le tableau de la Figure 35.b présente la liste des tâches et le numéro du PACM auquel appartient chaque tâche. La Figure 35.c illustre le nom de la tâche, son numéro et le PACM auquel elle appartient.

La modélisation des individus par des vecteurs est aussi simple à établir qu'à manipuler (tout au long des calculs). Toutefois, deux vecteurs différents peuvent représenter le même individu. En effet, l'ordre des PACM n'influe pas sur l'individu. Par conséquent, les individus "111222" et "222111" représentent la même partition qui est constituée par les 3 premières

tâches dans un PACM et les 3 autres dans l'autre PACM. Afin de résoudre la redondance des individus, nous imposons que les numéros de PACM dans un individu apparaissent selon un ordre croissant. Une fonction Résoudre_redondance applique cette règle aux individus de la population.

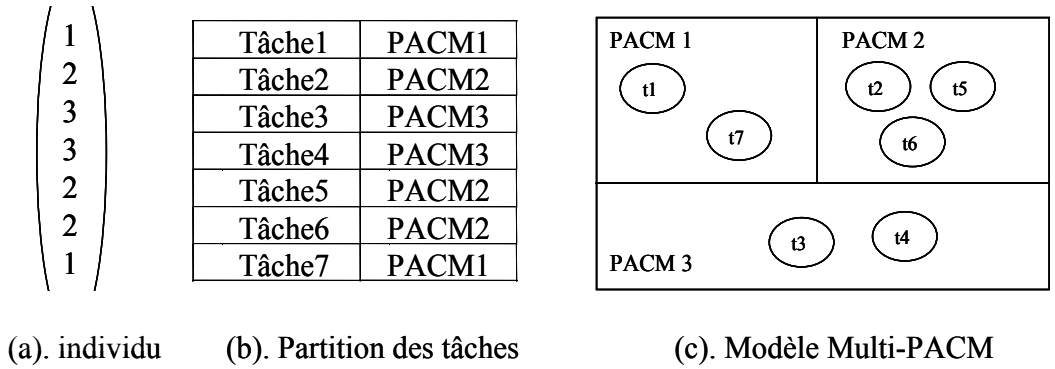


Figure 35 : Exemple d'un individu

La population initiale est choisie aléatoirement. Il s'agit de désigner aléatoirement un ensemble d'individus. La population initiale est un ensemble de vecteurs. Chaque vecteur représente un individu.

3.4.2. Evaluation des individus

Il s'agit d'évaluer d'abord les différentes solutions. Pour chaque solution, un vecteur métrique est calculé. Ce vecteur représente les différentes métriques que nous avons définies. Ces métriques servent d'outil d'analyse des différents individus qui représentent la population des différentes générations. Nous utilisons les métriques comme modèle d'analyse afin de comparer les différents individus par la notion de domination.

3.4.3. Sélection

Pour la sélection des parents qui vont générer la future génération nous utilisons la méthode de sélection par tournois. La méthode de sélection par tournois consiste à choisir aléatoirement un nombre de k individus dans la population, et à sélectionner pour la reproduction celui qui a la meilleure performance.

La comparaison des performances des individus sélectionnés se fait selon la notion de dominance. Tout d'abord, tous les individus sont considérés comme de potentiel vainqueur du tournoi (Figure 36, partie I). Ensuite, les individus dominés sont éliminés du tournoi (Figure 36, partie II). Un individu A est considéré comme dominé s'il existe parmi les (k-1) autres

individus choisis pour le tournoi un individu B tel que B domine A. Ce choix permet de garder tous les optimums globaux au sens du pareto.

Enfin, un individu non dominé est sélectionné pour le croisement (Figure 36, partie III). Dans le cas où il ne reste qu'un seul individu parmi les k sélectionnés, il est désigné comme l'individu vainqueur du tournoi et il est sélectionné pour le croisement. Dans le cas où il reste plusieurs individus, ils sont tous considérés comme vainqueurs du tournoi et l'un d'eux est tiré aléatoirement pour participer au croisement.

```
Fonction Selection par tournois
{
        Pour i allant de 1 à k
            Table_domination[i]=0;  // Initialiser tout les individus comme non dominées

         Pour i allant de 1 à k
        {
        j=0;  domin=faux;
        tant que ((j<k) et (domin==faux))
            {domin=dominate(partition[i],partition[j],met); //vérifier si l'individus i est dominé par j
             j++; }
        Si (domin=faux) Table_domination[i]=1; //Marquer l'individus dominé par 1
        }

        individus_selectectionné=-1;
        Tant que (individus_selectectionné==-1)
        {
        si (Table_domination[i]) individus_selectectionné=(partition[i]);
        i++;
        }
}
```

Figure 36 : Code de la sélection par tournois

Au cours d'une étape de sélection, il y a autant de tournois que d'individus sélectionnés. Les individus qui participent à un tournoi sont remis dans la population. La probabilité que le meilleur individu de la population ne soit pas choisi en k tirages est ((N-1)/N)k [Déro 2003]. En supposant que N est très grand devant k, cette probabilité est approximativement 1-(k/N) par un développement limité à l'ordre 1. Ainsi la probabilité que le meilleur individu soit tiré au moins une fois dans un tournoi est proche de k/N. Dans une génération, il y a N-λ tournois. Donc, le meilleur individu aura k(N-λ)/N sélections espérées. λ est le nombre de mutations dans une génération. Dans [Déro 2003], pour le cas où il y a N sélections par génération, il est conseillé d'avoir un cœfficient de pression k au minimum égal à 2 pour assurer la convergence. Nous choisissons de fixer 3 comme valeur minimum de k pour assurer la convergence de l'algorithme génétique.

3.4.4. Croisement

Nous utilisons un croisement à un point (Figure 37). Il s'agit tout d'abord de sélectionner deux parents (Figure 37-a). Ensuite, nous choisissons au hasard un point de croisement (Figure 37-b). Ce point est une position à laquelle nous découpons les parents en deux parties. Le croisement donne naissance à deux enfants en mélangeant les parties de leurs parents (Figure 37-c). Chaque enfant aura une partie du parent 1 et l'autre partie du parent 2.

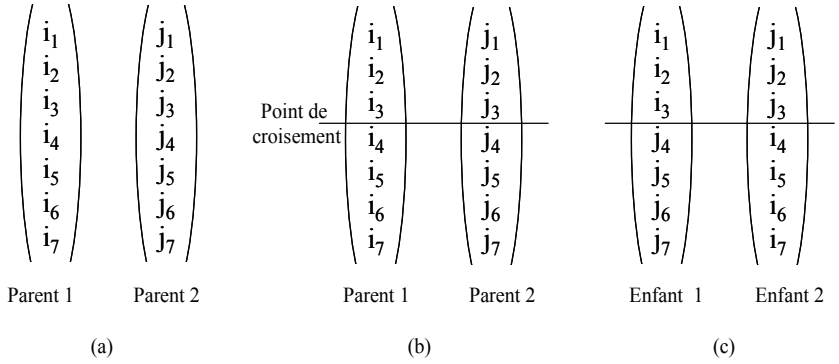


Figure 37 : Croisement

3.4.5. Mutation

La mutation consiste à faire apparaître des nouveaux individus dans la population. Ceci à pour intérêt d'assurer une diversité continue parmi les individus de la population. La mutation consiste à sélectionner un individu de la population (Figure 38-a), à choisir aléatoirement un gène (une affectation d'une des tâches de l'application à un des PACM comme le montre la Figure 38-b), et à choisir un numéro de PACM aléatoire pour remplacer l'ancien PACM.

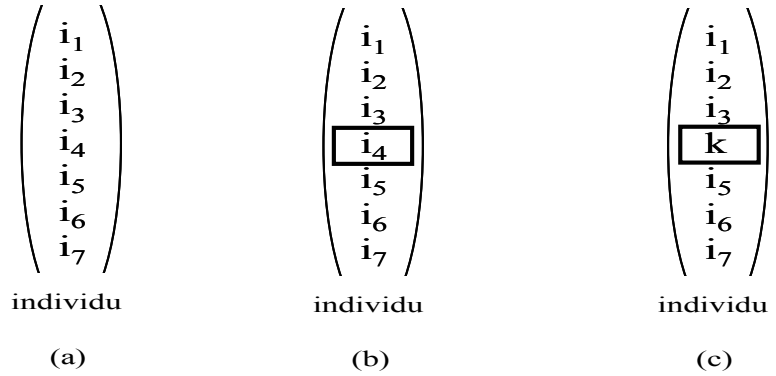


Figure 38 : Mutation

4. OUTIL GAMA² ET DECISIONS DU CONCEPTEUR

Nous avons réalisé un outil GAMA² pour effectuer le calcul des métriques et l'exploration par l'algorithme génétique. Le concepteur intervient pour guider l'outil à trouver l'architecture adéquate à ses exigences.

4.1. L'OUTIL GAMA²

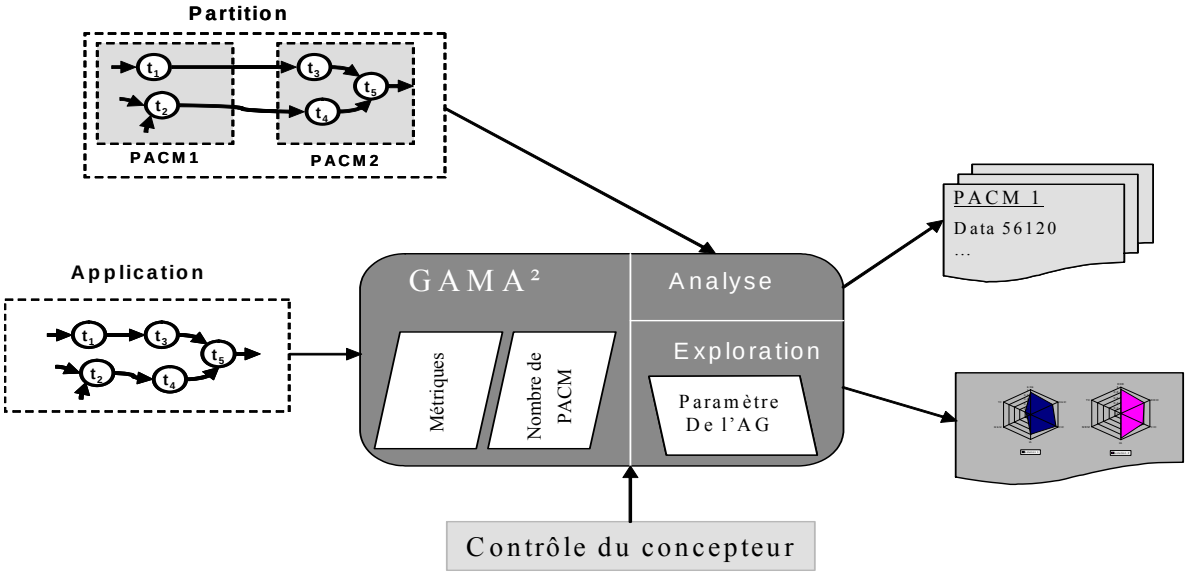


Figure 39 : Description des entrées et sorties de l'outil GAMA²

L'outil GAMA² comporte deux modules (Figure 39). Le premier est un module d'analyse qui permet de fournir pour chaque partition particulière des informations détaillées comme les données échangées dans et entre les PACM. Ce module a pour entrée le graphe de spécification de l'application et la partition à analyser. Le concepteur définit le nombre de PACM de l'architecture cible. L'outil analyse cette partition et fournit :

- ✓ La valeur des métriques choisie pour l'analyse.
- ✓ Le nombre des données échangées et des données mémorisées dans chaque PACM.
- ✓ Le nombre des données échangées et des connexions entre les PACM.
- ✓ La métrique d'analyse d'identité locale de chaque PACM P_j, $LIAP(P_j)$.
- ✓ Le plus grand et le plus petit débit contrainte de chaque PACM.
- ✓ Des données globales pour toute l'architecture :
    - o Le total des données échangées

- Le total des données mémorisées
- Le plus grand débit contrainte
- Le plus petit débit contrainte

Le deuxième module est celui de l'exploration. Il a pour entrée le graphe de spécification de l'application. Le concepteur définit les paramètres de l'architecture, les métriques utilisées pour l'analyse et les paramètres de l'algorithme génétique (Figure 40). Les paramètres de l'architecture sont le nombre de PACM qu'elle contient et le nombre de types de processeur mis à la disposition du concepteur. Les paramètres de l'algorithme génétique sont :

- ✓ Le nombre d'individus dans la population
- ✓ Le nombre de générations : le nombre d'itérations à effectuer par l'algorithme génétique.
- ✓ Le nombre de mutations et de croisements nécessaires pour la création de la nouvelle génération. Nous choisissons généralement un nombre de mutations autour de 8% de la population.
- ✓ Le nombre d'individus choisis pour le K tournoi.

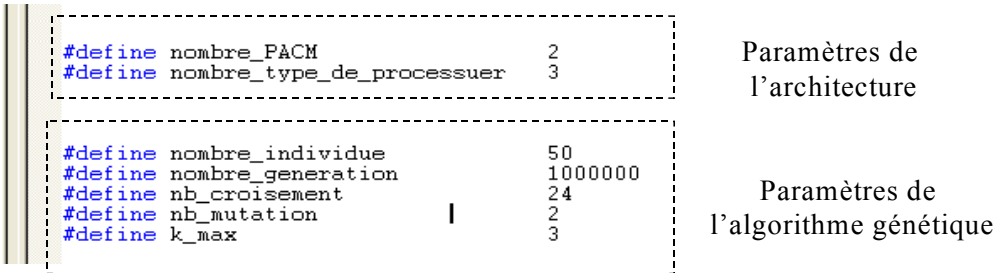


Figure 40 : Paramètres de l'exploration de GAMA²

4.2. *LA DECISION DU CONCEPTEUR*

Les métriques ont permis de comparer les différentes partitions. L'algorithme génétique multi-objectif a permis de trouver un groupe de solutions. Quelques exemples sont illustrés dans la Figure 41. L'exemple b de la figure représente une solution qui optimise toutes les métriques en même temps. Cette solution représente une bonne solution pour le problème. Toutefois, quand toutes les métriques sont assez élevées et pour des raisons qu'il juge intéressantes, il arrive que le concepteur ait une préférence pour optimiser une des métriques

par rapport aux autres. L'algorithme génétique permet aussi de trouver ce genre de solution. En effet, comme défini plus haut, il s'agit de trouver les optimums au sens de Pareto.

L'algorithme génétique multi-objectif permet de trouver des solutions qui peuvent intéresser le concepteur comme les c et d de la Figure 41. Ces deux solutions optimisent la métrique **EDE** mieux que la solution b. Toutefois, une ou plusieurs autres métriques sont légèrement inférieures à celles de l'exemple b. La différence des surfaces est marquée en hachuré. Cependant, l'algorithme peut trouver des solutions où est optimisée une des métriques mais qui perd beaucoup au niveau des autres métriques (l'exemple e de la figure).

Métriques	EDE	DEIC	CIC	MEM	IA	Tc
Exemple 1	0,8	0,8	0,8	0,8	0,8	0,8
Exemple 2	0,9	0,8	0,8	0,7	0,7	0,8
Exemple 3	0,9	0,7	0,7	0,7	0,7	0,7
Exemple 4	1	0,2	0,2	0,2	0,2	0,2

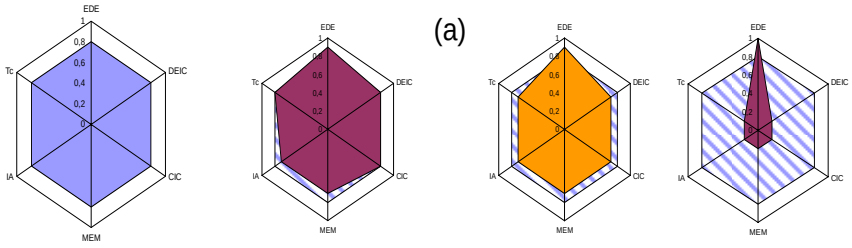


Figure 41 : Exemple de quelques espace des métriques

Donc, il incombe au concepteur d'éliminer les solutions telles que la solution e de la Figure 41. Toutefois, il aura le choix des solutions en fonction de ses besoins. Nous pouvons nous demander pourquoi il faut laisser une solution comme la e si c'est au concepteur de la supprimer plus tard. En fait, toutes les métriques ne s'approchent pas toujours de 1. Il arrive donc que la solution soit peut-être une bonne solution. Par conséquent, il est difficile de savoir si une solution est bonne ou non avant de voir les autres solutions. En outre, la décision dans ce genre de cas est rapide vu qu'une appréciation visuelle rapide permet de repérer les mauvaises solutions.

5. CONCLUSION

Le problème de pré-exploration est un problème multi-objectif. Il se base sur 3 composants :

- ✓ Un modèle d'analyse : nous avons défini un ensemble de 6 métriques capables d'analyser les communications, les mémoires, les performances d'exécution des tâches

et le parallélisme. Chaque métrique est un réel compris entre 0 et 1. Plus la métrique est proche de 1 plus elle optimise le critère auquel elle est affectée.

- ✓ Un outil d'analyse : nous avons utilisé un algorithme génétique multi-objectif. L'algorithme génétique fournit un groupe de solutions permettant un grand choix pour le concepteur.
- ✓ Le décideur : le concepteur élimine les solutions qui ne l'intéressent pas et choisit la solution qui correspond à ses préférences selon les besoins de l'application, le marché ou la flexibilité.

Dans le chapitre suivant nous utilisons différentes applications pour valider les métriques, le graphe et l'approche que nous proposons.

V.APPLICATIONS ET RESULTATS

1. INTRODUCTION 97
2. EMETTEUR UMTS 97
2.1. PRESENTATION DE L'APPLICATION 98
2.2. GRAPHE DE SPECIFICATION 99
2.3. ANALYSE ET METRIQUES 102
2.4. EXPLORATION 111
2.5. IMPLEMENTATION DE L'EMETTEUR UMTS 115
3. SYSTEME DE DECODAGE DE SIGNAUX AC3 117
3.1. PRESENTATION DE L'APPLICATION 118
3.2. GRAPHE DE SPECIFICATION 118
3.3. EXPLORATION POUR UNE ARCHITECTURE EN DEUX PACM 120
3.4. EXPLORATION PAR CODEF 121
3.5. PRE-EXPLORATION PAR GAMA² ET CODEF 122
3.6. COMPARAISON DES RESULTATS DES DEUX APPROCHES 125
4. APPLICATION ICAM 126
4.1. PRESENTATION DE L'APPLICATION 127
4.2. GRAPHE DE SPECIFICATION 129
4.3. ICAM QUATRE FLOTS 129
4.4. ICAM HUIT FLOTS 134
4.5. CONCLUSION 135
5. CONCLUSION 136

1. INTRODUCTION

Dans ce chapitre, nous utiliserons plusieurs applications pour valider les différents aspects de l'approche présentée précédemment. Nous expérimentons l'approche sur 3 applications.

La première est l'émetteur d'un terminal UMTS. Nous détaillons, la spécification du graphe de tâches, l'analyse du concepteur et des tâches. Nous utilisons cette application pour valider les métriques et montrer leurs pertinences à quantifier les caractéristiques qu'elles analysent. Le résultat de l'exploration automatique par l'outil GAMA^2 sera comparé avec les résultats d'une optimisation manuelle.

La seconde étude porte sur une application de codage audio AC3. Cette application permet de valider le flot proposé en deux étapes. L'outil d'exploration logiciel/matériel CODEF est utilisé pour la deuxième étape du flot d'exploration. La combinaison des deux outils permet de réduire l'espace de conception et d'identifier certaines architectures optimisées.

Enfin la troisième étude teste l'efficacité de l'approche proposée et de l'outil GAMA² pour des applications contenant un nombre de tâches élevé. Plusieurs scénarios sont considérés depuis une spécification d'une dizaine de tâches jusqu'à une cinquantaine de tâches et pour une architecture contenant une dizaine de processeurs. Cette troisième application (suivi d'objets dans une image, ICAM) teste la capacité de l'approche proposée à appréhender un espace de conception étendu aussi bien du point de vu de l'application que de l'architecture.

2. EMETTEUR UMTS

UMTS (abréviation de *Universal Mobile Telecommunications System*) désigne une nouvelle norme de téléphonie mobile. Nous parlons plus généralement de téléphonie de troisième génération ou 3G. Conçu en 1999 et depuis peu intégré dans nos téléphones portables, l'UMTS [Gpp] est un standard de transmission permettant d'acheminer texte, voix numérisée, vidéo et multimédia à un débit maximal de 2 *Mbit/s.* Ce standard a révolutionné le monde de la télécommunication mobile. Il a permis de transmettre des messages téléphoniques vocaux d'une qualité équivalente à celle de la téléphonie par réseaux fixes. Ce

standard permet d'offrir plusieurs services multimédia de très haut débit comme la transmission des vidéos et des images d'une haute qualité.

2.1. PRESENTATION DE L'APPLICATION

L'émetteur UMTS a été choisi pour valider différents points de la méthodologie développée. Le schéma fonctionnel de l'émetteur UMTS est celui développé dans le cadre du projet A3S [Rouxel 2006].

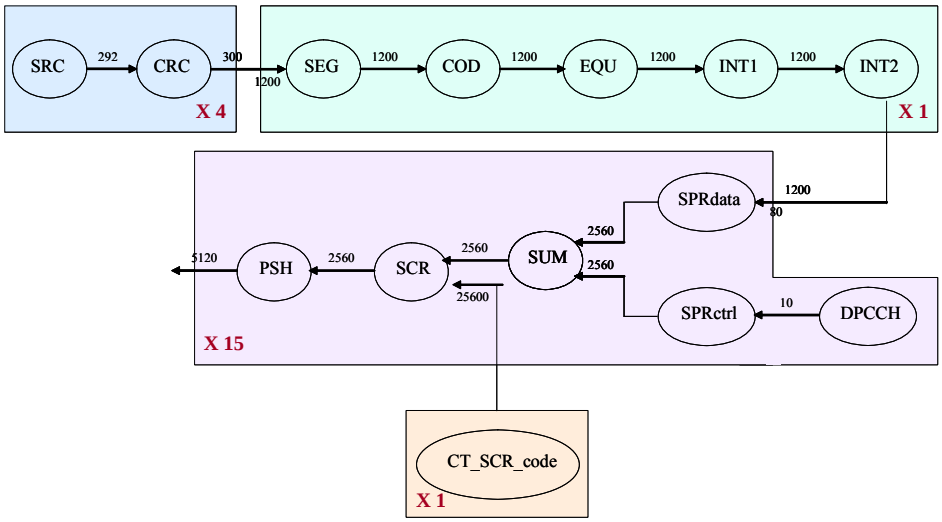


Figure 42 : Schéma de spécification de l'émetteur UMTS

Le schéma fonctionnel de l'émetteur est illustré sur la. Il présente les différentes tâches mises en œuvre dans l'émetteur ainsi que leurs interactions. Le schéma représente l'interactivité des différents blocs fonctionnels pour la génération d'une trame de 10 ms. Une exécution de ce schéma permet d'obtenir une trame.

La construction d'une trame est débutée par le *« transport block »*. Il est constitué par des données indicées (SRC), complétées au sein de chaque bloc de transport par un code de redondance cyclique (CRC) qui peut varier de 0 à 24 bits. Pour la configuration d'un débit de 117kbits/s, ce bloc est exécuté 4 fois (il est indexé par X4 sur la).

Les blocs à émettre devant avoir une taille fixe, ils subissent une concaténation ou une segmentation (SEG) avant la phase du codage canal (COD) (soit par un code convolutionnel, soit par turbo code). Ils subissent ensuite un ajustement ou une égalisation (EQU) pour être adaptés à la taille des blocs fixée et donc respecter la capacité du canal physique. Le codage canal fait en effet varier le nombre de bits de chaque bloc de transport. Les entrelacements (INT1 et INT2) qui suivent l'adaptation de débit ont pour effet de répartir aléatoirement les erreurs et donc d'augmenter les performances du CRC. Pour la configuration d'un débit de

117 kbits/s, ces différentes tâches sont exécutées une seule fois (ce bloc est indexé par X1 sur la).

L'étalement de spectre (SPR) qui sépare les différents canaux utilisés par un même utilisateur, peut enfin avoir lieu d'après les informations issues du canal physique (DPCCH). Une fois étalé, un code d'embrouillage (SCR) aléatoire est appliqué pour identifier les différents utilisateurs d'une même cellule. Un dernier filtrage (PSH) est ensuite exécuté avant la modélisation et l'émission de l'onde radio. Ces tâches constituent le bloc fonctionnel d'une trame. Pour une configuration d'un débit de 117 kbits/s, ce bloc est exécuté 15 fois (il est indexé par X15 sur la).

Un générateur d'embrouillage (CT_SCR_code) est utilisé pour induire des erreurs sur la trame émise pour simuler les brouillages de transmission.

2.2. GRAPHE DE SPECIFICATION

Dans cette section, nous nous intéressons à la mise en œuvre du graphe de spécification de l'émetteur UMTS. Cette spécification est décrite dans le chapitre précédent.

La spécification de l'émetteur UMTS est basée sur une compréhension fonctionnelle de l'application enrichie par une analyse du concepteur. Le graphe de spécification de l'émetteur UMTS contient 14 nœuds et 13 arcs (ce qui correspond aux 14 tâches et aux 13 interactions entre les tâches). Cette information est codée dans le fichier de spécification par les déclarations **« .nodes 14 »** et **« .arcs 13 »**. La fréquence d'exécution de chaque tâche (décrite dans la section précédente) est codée dans le fichier de spécification par la déclaration **« Execution_frequency »** suivie du nombre d'exécutions de la tâche. Dans la Figure 43, qui représente une partie du fichier de spécification de l'émetteur UMTS, la tâche SCR ainsi que la tâche CRC sont exécutées 4 fois.

```
.system Emetteur_d_un_terminal_UMTS
        .nodes 14
        {
                .task SRC
                        {
                        .Number_of_the_task 1
                        .Execution_frequency 4
                        .Local_memory_size 0
                        .IA_vector 0.8  0.3
                        .Constraint_throughput 116800
                        }
                .task CRC
                        {
                        .Number_of_the_task 2
                        .Execution_frequency 4
                        .Local_memory_size 0
                        .IA_vector 0.8  0.3
                        .Constraint_throughput 0
                        }
```

Figure 43 : Exemple de spécification des tâches SRC et CRC de l'émetteur UMTS

L'émetteur UMTS, pour assurer la configuration 117 kbits/s ou plus précisément 116 800 bits/s, comporte 3 tâches qui ont des contraintes temporelles. La tâche d'entrée SCR a une contrainte de 116 800 bits/s. La tâche de sortie PSH a une contrainte de 7 680 000 bits/s. Le générateur d'embrouillage CT_SCR_code a une contrainte 3 840 000 bits/s. Les contraintes sont codées dans le fichier de spécification par la déclaration **"Constraint_throughput"** suivie par la valeur du débit contrainte. La déclaration **"Constraint_throughput"** dans la spécification de la tâche SRC (Figure 43) est suivie de 116 800 pour coder la contrainte de 116 800 bits/s. La tâche CRC n'ayant pas de contrainte temporelle spécifique, la déclaration **"Constraint_throughput"** de sa spécification est suivie de 0. Il n'y a pas eu d'analyse des tâches pour l'émetteur UMTS permettant d'estimer la *"local memory size"*. Cette information est donc codée en mettant 0 après la déclaration Local_memory _size.

L'architecture visée est basée sur plusieurs DSP identiques. Toutes les tâches sont disponibles en code exécutable sur les DSP. Le temps d'exécution des tâches sur DSP semble assez rapide sauf pour la tâche PSH qui prend trop de temps. Il semble donc plus judicieux de l'implémenter en matériel. L'analyse (du concepteur), ainsi faite, est traduite par un vecteur d'analyse d'identité (1) pour toutes les tâches. Ceci signifie que toutes les tâches ont au maximum la possibilité d'être exécutées sur DSP ou en matériel. Pour enrichir un peu la spécification et permettre plus de discussions lors des tests, nous supposons que la tâche PSH a la possibilité d'être exécutée sur un processeur spécifique qui lui permet une exécution plus optimale. Cette analyse se traduit alors par un vecteur d'analyse d'identité (0,8 0,3) pour toutes les tâches sauf pour PSH pour lequel il sera (0,3 0,8). Ceci signifie que toutes les tâches sauf PSH doivent être exécutées sur les PACM qui contiennent un DSP permettant l'optimisation de l'exécution de ces tâches. Toutefois, il reste possible d'exécuter ces tâches sur le PACM contenant le processeur spécifique même si cette solution n'optimise pas les performances de la tâche.

```
.Arcs 13
        {
                .from 1
                .to 2
                        {
                        .Word_nb 292
                        .Word_size 1
                        .Nb_of_execution 1
                        .Memorization 0
                        .Mode data
                        }
                .from 2
                .to 3
                        {
                        .Word_nb 300
                        .Word_size 1
                        .Nb_of_execution 1
                        .Memorization 1
                        .Mode data
                        }
```

Figure 44 : Exemple de spécification de deux arcs de l'émetteur UMTS

La quantité des données transférées d'une tâche à une autre est exprimée par les trois paramètres ***Word_nb***, ***Word_size***, et ***Nb_of_execution***. Par exemple la tâche **SRC** envoie à la tâche CRC 292 bits une seule fois. Ceci est exprimé, comme le montre la Figure 44, par ***Word_nb : 292***, ***Word_size : 1***, et ***Nb_of_execution : 1.*** La tâche **CRC** est exécutée 4 fois et envoie à chaque fois 300 bits à la tâche **SEG** à qui il faut les 1200 bits pour commencer son exécution. Ainsi, l'analyse montre qu'il est judicieux de mettre en mémorisation ce transfert de données. Ceci est codé en mettant 1 devant la déclaration ***Memorization***.

Dans le Tableau 5, nous résumons les informations importantes de la spécification de l'émetteur UMTS. Il y a 200 518 bits de données échangés entre les différentes tâches de l'application UMTS et 13 échanges. A priori, 40 800 bits seront mémorisés. Les arcs qui transfèrent les données mémorisées sont énumérés dans le tableau. L'application a des contraintes sur 4 tâches, nommées C1, C2, C3 et C4.

Quantité de données échangées	**200 518 bits**
Nombre de connexions total	**13 bits**
Quantité de données mémorisées	**40 800 bits**
Arcs mémorisés	**3 arcs mémorisés** o **De CRC à SEG : 1200** o **De INT2 à SPRdata : 1200** o **De CT_SCR_code à SCR : 38400**
Contraintes	**4 contraintes** o **C1 (SRC : 116 800 bits/s) :** o **C2 (CT_SCR_code : 38 400 000 bist/s)** o **C3 (DPCCH : 15 000 bits/s)** o **C4 (PSH : 7 680 000 bits/s)**

Tableau 5 : Résumé de spécification de l'émetteur UMTS

La spécification de l'émetteur UMTS est basée sur la spécification fonctionnelle de l'application. Elle n'a demandé ni une compréhension détaillée de l'application, ni une connaissance détaillée des spécifications de la tâche. Elle a nécessité un coût minimal en temps de spécification. Elle n'a pas exigé de coûts d'estimation des performances des différentes tâches sur chaque DSP ou en matériel. Toutefois, une connaissance minimale des tâches et l'expérience du concepteur ont permis d'enrichir la spécification. En effet, le vecteur d'analyse d'identité demande une analyse des tâches par le concepteur ou par des outils d'analyse de tâche à haut niveau. Cette analyse reste toujours moins coûteuse que l'estimation

des performances de chaque tâche pour chaque support. Le concepteur peut intervenir aussi pour guider l'exploration pour minimiser le coût du système ou favoriser la flexibilité.

2.3. ANALYSE ET METRIQUES

Dans cette section, nous considérons l'analyse de plusieurs exemples afin de montrer la fiabilité des métriques à analyser et des caractéristiques qui leur sont attribuées.

Dorénavant, nous utiliserons des exemples de classification pour discuter et valider les métriques. Les classifications seront représentées par un vecteur (comme celui de la Figure 45.a). L'exemple de la Figure 45 permet d'expliquer la signification de ce vecteur. Elle illustre la représentation d'un exemple de classification de l'émetteur UMTS en deux PACM. La Figure 45.a montre le vecteur représentant la classification utilisée par l'outil d'analyse et d'exploration. Ce vecteur représente l'affectation de chaque tâche au PACM 1 ou au PACM 2. Le chiffre représente le numéro du PACM. La position du chiffre dans le vecteur représente le numéro de la tâche. Le tableau de la Figure 45.d présente la liste des tâches et le numéro du PACM auquel appartient chaque tâche. La Figure 45.c illustre le nom de la tâche, son numéro et le PACM auquel elle appartient. Finalement sur la Figure 45.d, nous pouvons voir le schéma de spécification de l'émetteur UMTS et la classification précédemment expliquée.

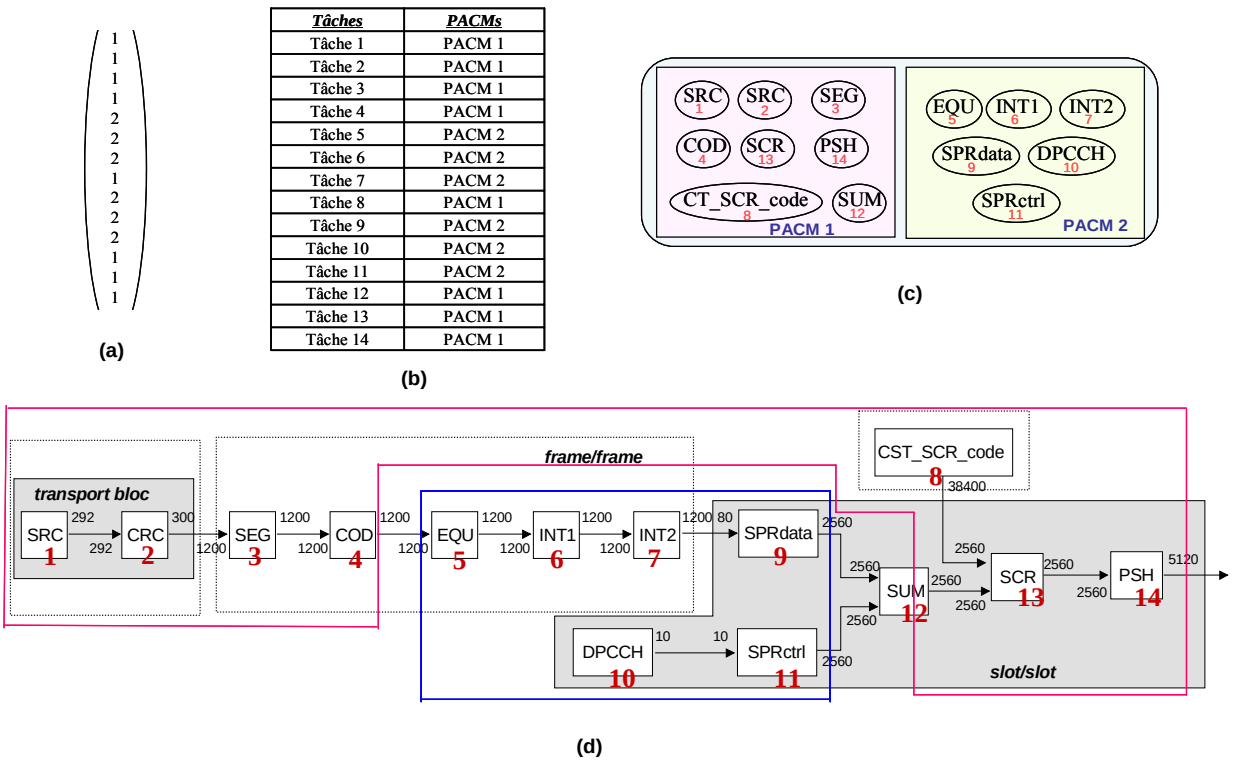


Figure 45 : Exemple d'un individu de l'émetteur UMTS

2.3.1. Analyse de la communication

L'analyse de la communication comme expliquée dans le chapitre précédent consiste en deux points clés :

- ✓ Equilibrer la quantité des données échangées dans chaque PACM. Ceci a pour but de diminuer la quantité des données gérées dans chaque PACM. Cette analyse est assurée par la métrique **EDE**
- ✓ Minimiser les échanges des données entre les PACM. Ceci consiste en, respectivement, minimiser la quantité des données échangées entre les PACM et minimiser le nombre d'échanges de données entre les PACM. Cette analyse est assurée par deux métriques **DEIC** et **CIC**.

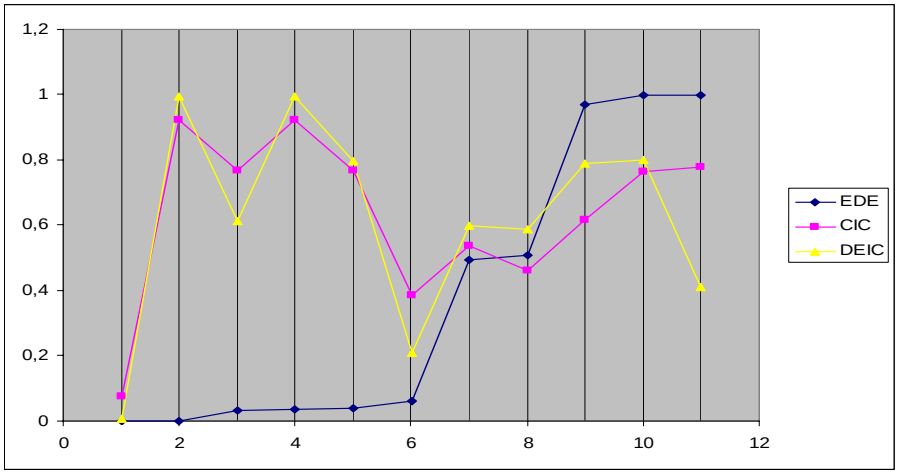


Figure 46 : Métriques de communication pour quelques exemples d'implémentation de l'émetteur UMTS en deux PACM

Nous considérons l'implémentation de l'application UMTS en deux PACM. La Figure 46 illustre les métriques de communication de plusieurs exemples d'implémentation de l'émetteur UMTS en deux PACM. Cette figure représente les valeurs des 3 métriques de communication (axe vertical) pour chaque exemple (le numéro de l'exemple est représenté sur l'axe horizontal). Les exemples sont triés par ordre croissant de EDE.

Sur la figure, nous distinguons des exemples de partitions (exemples 1 et 6) qui présentent une mauvaise distribution des données entre et dans les PACM. L'exemple 1, détaillé dans la Figure 47.a, a seulement 1% des données échangées dans le PACM 1 (soit 1 200 bits) et 0 données échangées dans le PACM 2. La gestion de la communication dans les

PACM est alors très simple. Elle se limite à la communication entre les tâches INT2 (tâche 7) et la tâche SPRdata (tâche 9). Il suffira que ces deux tâches soient exécutées sur le DSP pour que la communication ne nécessite aucune ressource. Par contre, la majorité des données échangées sont échangées entre les PACM. Les ressources de communication nécessaires sont conséquentes à la quantité des données gérées. Donc, le coût des communications augmente. En effet, plus la quantité de données est importante, plus les ressources de communication doivent comporter des protocoles complexes, assurer des fréquences élevées et une largeur plus importante. Plus la fréquence est élevée, plus la consommation est élevée. Plus le bus est large, plus la consommation est élevée. Plus la quantité de données échangées est grande, plus il y a des risques de latence. Ceci implique que le coût des communications entre les PACM est maximal. En résumé, pour l'exemple 1, bien qu'il présente une communication très simple à réaliser, voire inexistante, la communication entre les PACM est coûteuse. Ceci est dû à une mauvaise distribution des échanges des données. L'analyse automatique de la distribution confirme l'analyse manuelle que nous venons de faire. En effet (EDE ≈ DEIC ≈ CIC ≈ 0) indique que la plupart des données échangées (99%) et des échanges (12 arcs/13) s'effectuent entre les PACM.

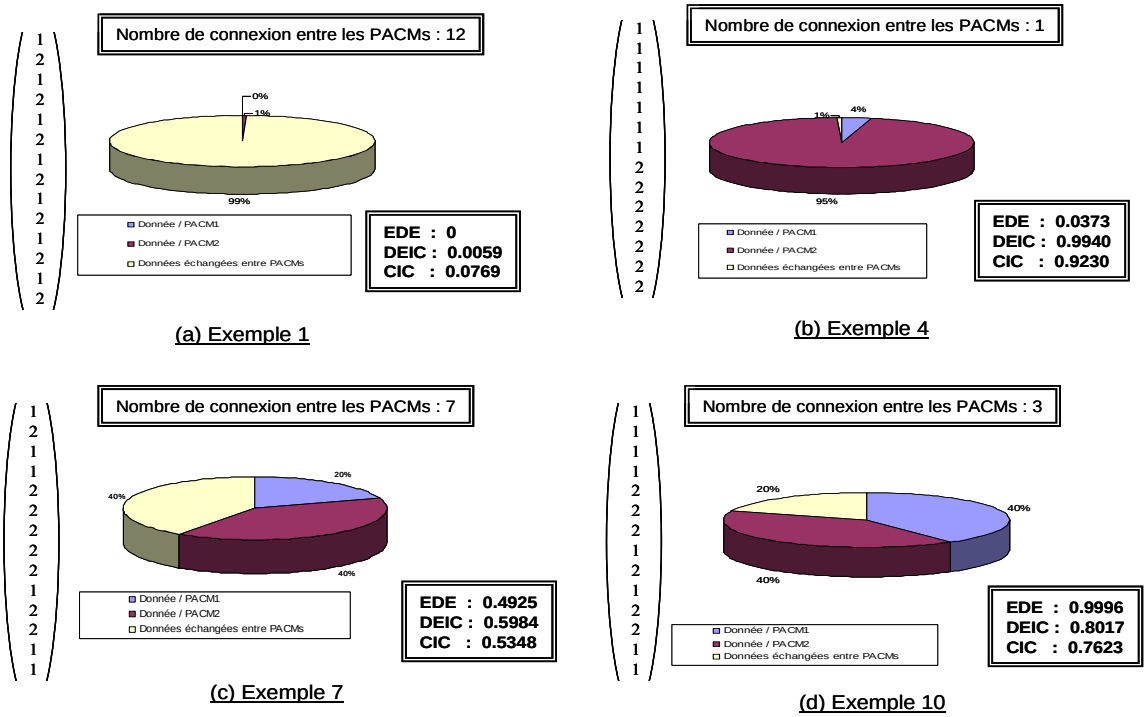


Figure 47 : Le vecteur de classification, la distribution des données et les métriques de 4 exemples

Les exemples 2 à 5 de la Figure 46 illustrent des partitions où la communication entre les PACM est bien optimisée. La partition 4 (détaillée dans la Figure 47.b), par exemple, a une communication entre les PACM bien optimisée. En effet, la quantité de données échangées entre les PACM est très faible (à peine 1%). Il s'agit d'un seul échange entre les tâches INT2 et SPRdada (1 200 bits). Ainsi, les ressources de communication nécessaires peuvent se restreindre à une interface point à point. Donc, les coûts de la conception de la communication entre les PACM en temps de conception, en ressources sont assez optimisés. L'analyse automatique de cette partition confirme l'analyse manuelle. En effet, les métriques DEIC et CIC sont très proches de 1. Ces deux métriques, quand elles sont proches de 1, indiquent que les communications entre les PACM sont bien optimisées. La quantité des données échangées dans le PACM 1 est, aussi, assez faible. Elle est de 7 168 bits, soit 4% des données totales échangées. Donc les coûts de la conception de la communication dans le PACM 1 sont aussi assez faibles. Par contre, la quantité des données échangées dans le PACM 2 est assez importante. Elle est de 192 150 bits, soit 95%. Les coûts de la conception de ces communications, pour les mêmes raisons que pour la conception des communications entre PACM de l'exemple 1, sont par conséquent assez importants. L'analyse automatique confirme encore l'analyse manuelle. La métrique EDE, qui analyse la distribution des données dans les PACM, est de 0,03. Ceci signifie que la distribution des données entre les PACM est très mauvaise du fait que la majorité des données échangées est concentrée dans un seul PACM.

Les exemples 7 à 11 présentent des partitions qui n'ont pas une très mauvaise distribution des données dans les PACM et entre les PACM. Dans la partition 7, 20% des données échangées sont dans le PACM 1 et 40 % dans le PACM 2. La quantité de données échangées dans les PACM est raisonnable bien qu'elle reste encore déséquilibrée. L'analyse automatique des tâches confirme cette analyse. La métrique EDE est égale à 0,4924, ce qui signifie que l'analyse automatique juge que la qualité de la distribution des données dans les PACM n'est pas assez optimisée. 40 % des données échangées sont entre les PACM. Il y a 7 échanges parmi les 13 qui sont effectués entre les PACM. Bien que cette solution soit plus optimisée que l'exemple 1, la communication entre les PACM reste encore assez mal optimisée. L'analyse automatique confirme ce résultat : "DEIC = 0,5984" et "CIC = 0,5348". Donc, dans l'ensemble la communication n'est pas encore assez optimisée. Pour la partition 10 par exemple, 40% des données échangées totales sont échangées dans chaque PACM, 20% entre les PACM qui correspondent à 3 échanges de données. La distribution des données est

assez bonne et équilibrée dans et entre les PACM. L'analyse automatique le confirme facilement puisque les 3 métriques sont assez proches de 1.

En résumé, ces différents exemples ont montré que l'analyse automatique de la communication est capable de valider les résultats d'une analyse manuelle. Ceci montre la fiabilité des métriques à refléter la qualité de la distribution des échanges de données et de prévoir son impact sur les coûts de la conception de la communication.

2.3.2. Analyse des contraintes

L'analyse des contraintes consiste à analyser la qualité de la distribution des différents débits entre les groupes. Plus il y a de débits contraintes similaires dans le même PACM, plus la métrique contrainte tend vers 1.

	Exemple	Groupe 1	Groupe 2	Tc
1	11222221212221	C1C2C3C4		0
2	11222221212222	C1C2C3	C4	0
3	11222221222211	C1C2C4	C3	0
4	11111111222222	C1C2	C3C4	0,002652
5	11111112222222	C1	C2C3C4	0
6	11111221222211	C1C3C4	C2	0,002652
7	11111222112222	C1C3	C2C4	0,1996
8	11111222222211	C1C4	C3C2	0

Tableau 6 : Métriques contraintes de l'émetteur UMTS

Comme résumé dans le Tableau 5, l'émetteur a 4 contraintes sur 4 tâches nommées C1, C2, C3 et C4. C1 et C3 sont respectivement d'un débit d'une centaine et une dizaine de kbits/s alors que C2 et C4 dépassent les 5 Mbits/s. Par conséquent, une partition qui distribue C1 et C3 dans un PACM et C2 et C4 dans le deuxième permet d'optimiser au mieux la distribution des contraintes. Toutefois, il y a une différence de 100 kbits/s entre C1 et C3 et surtout une différence de 30 Mbits/s entre C2 et C4. Donc, même avec la meilleure distribution, il reste dans le même PACM des débits assez différents. Pour étudier l'analyse automatique de la distributions des *débits contraintes*, nous avons calculé les métriques *débits contraintes* de 8 exemples représentant les différentes possibilités de distribution *des débits contraintes*. Le Tableau 6 récapitule les différents exemples choisis, leurs distributions des contraintes et leurs métriques de débits contraintes **Tc**. L'analyse automatique confirme bien l'analyse manuelle que nous avons effectuée. En effet, la métrique **Tc** de toutes les distributions de *débits contraintes* est presque égale à zéro sauf pour la partition 7 qui distribue C1 et C3 dans un PACM et C2 et C4 dans un deuxième qui est égal à 0,2. Ceci montre bien que l'analyse caractérise bien la qualité de la distribution des *débits contraintes*.

Elle permet aussi de trouver la partition qui améliore au mieux la distribution des contraintes même si cette dernière n'est pas optimisée.

Contraintes	Débit
C1	11 000
C2	13 000
C3	8 000 000
C4	7 680 000

(a)

	Solution	Groupe 1	Groupe 2	TC
1	11222221212221	C1C2C3C4		0
2	11222221212222	C1C2C3	C4	0
3	11222221222211	C1C2C4	C3	0
4	11111111222222	C1C2	C3C4	0
5	11111112222222	C1	C2C3C4	0.000250
6	11111221222211	C1C3C4	C2	0
7	11111222112222	C1C3	C2C4	0.959945
8	11111222222211	C1C4	C2C3	0.000250

(b)

Figure 48 : (a) Les exemples de débits utilisés (b) Métriques contraintes pour l'émetteur UMTS avec les contraintes modifiées

Afin d'enrichir la discussion de la métrique des *débits contraintes,* nous modifions la valeur des contraintes de l'émetteur UMTS par les valeurs de la Figure 48.a. Les contraintes C1 et C2 sont assez faibles et de valeurs proches. C3 et C4 sont assez élevées et de valeurs proches. Ainsi, seule une partition assurant une distribution qui permet d'avoir C1 et C3 dans un PACM et C2 et C4 dans l'autre PACM optimise la distribution des contraintes. En effet, l'analyse automatique le confirme bien. Dans le tableau de la Figure 48.b, il y a 8 partitions présentant toutes les possibilités de distribution des contraintes ainsi que leurs métriques. A part la métrique de la partition 7, toutes les autres métriques sont égales ou proches de zéro. La métrique de la partition 7 est très proche de 1, ce qui montre que la partition 7 assure une très bonne optimisation des *débits contraintes* et présente la meilleure distribution des contraintes. En résumé, ces quelques exemples ont montré que l'analyse automatique des *débits contraintes* est capable de valider les résultats d'une analyse manuelle

2.3.3. Analyse d'identité

L'analyse de l'identité comme expliquée dans le chapitre précédent consiste à vérifier l'équilibrage de la qualité de la distribution des tâches en groupes ayant les mêmes identités. Plus la métrique IA est proche de 1 plus l'exécution de chaque tâche sur la ressource choisie est performante.

Toutes les tâches de l'émetteur UMTS ont un vecteur d'analyse d'identité qui juge que ces dernières sont performantes si elles sont exécutées sur le DSP, sauf la tâche PSH qui a une analyse d'identité qui la tend à être exécutée sur le processeur spécifique. Quelle que soit la

partition en deux PACM de l'émetteur UMTS, la distribution des tâches est majoritairement optimisée car toutes les tâches ont la possibilité d'être exécutées sur le DSP qui les optimisent, sauf PSH qui n'a pas la possibilité d'être exécutée sur le processeur spécifique qui l'optimise. Nous prenons l'exemple de la partition "11111 11222 2222". Dans le PACM 1, toutes les tâches ont une identité maximale pour le DSP. Donc, la métrique d'identité locale est 1 pour le DSP. Dans le PACM 2, à part la tâche PSH, toutes les tâches ont une identité maximale pour le DSP. Comme PSH a seulement une identité 0,3 pour le DSP, la métrique d'analyse d'identité est de 0,91 pour le DSP. La métrique d'analyse d'identité globale **IA**, qui est le minimum des métriques d'analyse d'identité locale, est de 0,91. Ce qui montre que la métrique d'analyse d'identité **IA** reflète bien l'analyse manuelle de la distribution des tâches.

2.3.4. Analyse de la mémoire

L'analyse de la mémoire consiste à vérifier la qualité de la distribution de la mémoire entre les différents groupes. Améliorer cette métrique consiste à minimiser la taille de la mémoire dans chaque groupe pour optimiser les temps d'accès, la consommation et la surface et simplifier la gestion des mémoires.

94% des données (38 400 bits) qui sont a priori mémorisées dans l'émetteur UMTS sont les données échangées entre la tâche CT_SCR_code et la tâche SCR (voir Tableau 5). Les deux autres échanges mémorisés ne sont que de 1 200 bits chacun. Par conséquent, il est presque impossible d'équilibrer la distribution de la mémorisation. L'analyse automatique confirme cette analyse. Pour une partition qui a toutes les mémorisations en un seul PACM, la métrique mémoire **MEM** est égale à zéro. Dans les autres cas, **MEM** varie entre 0,058 et 0,117. Donc quelle que soit la distribution, **MEM** reste proche de zéro.

2.3.5. Espace des métriques

L'analyse automatique effectuée à l'aide des métriques de communication, des *débits contraintes*, d'analyse de l'identité et de la mémoire a permis de quantifier et de refléter l'analyse manuelle pour les exemples étudiés. Ceci confirme l'efficacité de l'analyse automatique avec les métriques à caractériser la partition. Dans cette section, nous utilisons les métriques pour discuter et enrichir les explorations manuelles.

Nous tentons de proposer manuellement une partition en deux PACM adéquate pour l'émetteur UMTS. Nous proposons une partition simple qui consiste à répartir les tâches de 1 à 7 dans le PACM 1 et les tâches 8 à 14 dans le PACM 2. Afin d'étudier la pertinence de cette partition, nous appliquons l'analyse automatique qui génère les différentes métriques. Les

valeurs des métriques sont présentées dans la colonne Exemple 1 du tableau (Figure 49.a). La projection de ces métriques sur l'espace des métriques est illustrée dans la Figure 49.b. Comme il a été remarqué lors de l'analyse des *débits contraintes* et de la mémoire, les deux métriques qui y sont assignées sont proches de zéro. Ceci est bien visible dans la Figure 49.b. En effet, l'espace des métriques de la partition considérée couvre très peu ces deux métriques. Par contre, l'espace des métriques de la partition choisie couvre bien les métriques d'analyse d'identité **IA** et de communication entre les PACM **CIC** et **DEIC**. En effet, comme illustré dans le tableau, **DEIC**, **CIC** et **IA** sont proches de 1. Ceci implique que la plupart des tâches auront une performance maximale sur les ressources d'exécution qui leurs sont assignées. Ainsi, les communications entre les PACM seront simples à concevoir et à gérer. Les coûts de ces communications en temps et en ressources sont bien optimisés. La métrique **EDE** qui est très proche de zéro indique que la communication dans un PACM va être coûteuse en temps de conception, en ressources de gestion des communications, en ressources de communication et en temps. En résumé, l'architecture qui résulte de cette partition aura une communication entre PACM peu coûteuse et la plupart des tâches auront des performances optimales. Toutefois, il y aura des latences du fait qu'il existe des tâches qui ont des contraintes avec des fréquences très différentes qui vont se partager les mêmes ressources de communication du PACM. Aussi, il y aura des latences du fait de la grande taille de la mémoire utilisée. Et surtout, il y aura une latence du fait que la plupart des données sont dans le même PACM. Cette partition n'est pas assez équilibrée.

Nous essayons une autre partition qui offre davantage de parallélisme. Nous considérons alors une partition qui permet le parallélisme entre les tâches SEG, COD, EQU, INT1, INT2 et SPRdata et les tâches DPCCH et SPRctrl. Cette partition permet aussi le parallélisme entre la tâche SUM et la tâche CT_SCR_code. La partition choisie est celle de la Figure 49.c. Les métriques générées par l'analyse automatique de cette partition sont présentées dans la colonne Exemple 2 du tableau de la Figure 49.a. Un coup d'œil rapide sur l'espace des métriques couvert par cette partition (Figure 49.c) et celui couvert par la partition de l'exemple 1 (Figure 49.b) permet de voir que cette partition sera plus coûteuse. En effet, les deux exemples améliorent les mêmes métriques (**DEIC**, **CIC**, **IA**) mais le deuxième exemple couvre moins d'espace. En effet, les métriques **EDE**, **MEM**, **Tc** et **IA** des deux partitions ont des valeurs proches. Seules les métriques **DEIC** et **CIC** de l'exemple 2 sont inférieures à celles de l'exemple 1. Ceci indique, qu'en plus des coûts de l'exemple 1, cette partition a une communication entre PACM plus coûteuse. Donc, cette partition est moins appropriée que la première.

Métriques	Exemple 1	Exemple 2	Exemple 3
EDE	0,037304	0,058824	0,568371
DEIC	0,923077	0,769231	0,769231
CIC	0,994016	0,611007	0,611008
IA	0,910714	0,910714	0,910714
MEM	0,058824	0,058824	0,058824
TC	0	0,199687	0

(a)

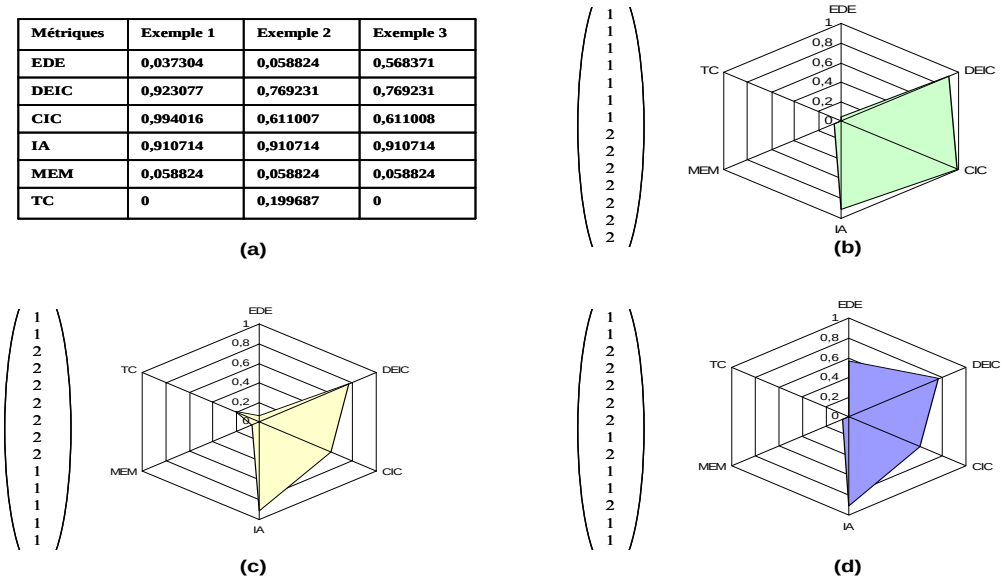


Figure 49 : (a) Les métriques des exemples analysés (b) L'exemple 1 et son espace des métriques (c) L'exemple 2 et son espace des métriques (d) L'exemple 3 et son espace des métriques

Pour garder les mêmes parallélismes que l'exemple 2, nous permutons les tâches SUM et CT_SCR_code de PACM. La partition choisie est illustrée dans la Figure 49.d. L'espace des métriques couvert par cette partition est présenté sur la même figure. Les valeurs des métriques sont dans la colonne Exemple 3 du tableau de la Figure 49.a. Une analyse rapide de l'espace des métriques couvert par cette partition montre déjà que les communications dans les PACM sont plus optimisées que celles des autres exemples.

La comparaison des espaces des métriques couverts par l'exemple 2 et 3 montre que l'architecture de l'exemple 3 est plus optimisée. En effet, toutes les métriques des deux exemples sont équivalentes sauf **EDE** qui est meilleure pour l'exemple 3. En comparant les espaces des métriques de l'exemple 1 et 3, nous remarquons que l'exemple 3 optimise mieux la communication dans les PACM qui est très mauvaise dans l'exemple 1. En effet, **EDE** est de 0,56 dans l'exemple 3, alors qu'elle est de 0,03. C'est normal puisqu'il y a moins de données concentrées dans un seul PACM. Toutefois, la surface couverte par les métriques de l'exemple 1 est plus grande que la surface couverte par les métriques de l'exemple 3. En effet la surface couverte par l'exemple 1 est de 0,95 alors que celle couverte par l'exemple 3 est de 0,75. Ceci s'explique du fait que les métriques **CIC** et **DEIC** de l'exemple 3 sont de 0,61 et 0,76 ce qui est inférieur aux valeurs de ces métriques pour l'exemple 1. Malgré que les métriques de l'exemple 1 couvrent plus de surface que celles de l'exemple 3, ce dernier permet d'avoir une architecture plus équilibrée que l'exemple 1 parce qu'il améliore plus de métriques (la métrique **EDE**).

2.4. *EXPLORATION*

Dans cette section, nous étudions l'exploration de l'espace des solutions de l'émetteur UMTS en 2 et 3 PACM. Cette étude permet de valider l'efficacité de l'outil d'exploration GAMA[2].

2.4.1. Pré-exploration en deux PACM

Nous considérons l'implémentation de l'émetteur UMTS sur une architecture comportant deux ressources logicielles. L'exploration avec un flot classique, en une seule fois, consiste à trouver une architecture adéquate permettant d'optimiser les performances des tâches, à gérer toutes les données échangées et à trouver les ressources de communication nécessaires pour gérer les différentes contraintes variant entre 100 kbits/s et 380 fois plus (38 Mbits/s). Comme le résume la Figure 50.a, une telle architecture sera basée sur deux DSP et doit gérer plus de 200 kbits. Le modèle d'architecture multi-PACM que nous proposons permet le partage des quantités des données échangées et de répartir les contraintes. L'exploration que nous proposons consiste en un premier lieu à répartir les contraintes, données et mémoires de façon à satisfaire les objectifs. L'objectif idéal, pour l'émetteur UMTS, est de trouver une partition (Figure 50.b) qui permet de regrouper les contraintes C1 et C3 ensemble dans un PACM et C2 et C4 dans un autre PACM, d'avoir 90 000 bits de données échangées dans chaque PACM et environ 10 000 bits entre les PACM. L'avantage d'une telle partition est, qu'au lieu de concevoir des ressources de communication capables de gérer des contraintes avec 38 Mbits/s de différence, il suffit de gérer des contraintes avec une différence de seulement 30 Mbits/s au maximum. Aussi, au lieu de gérer et mettre en œuvre les ressources de communication nécessaires pour 200 518 bits de données échangées, il est question de gérer et de concevoir les ressources de communication de 3 échanges de données dont le plus grand est de 90 000 bits.

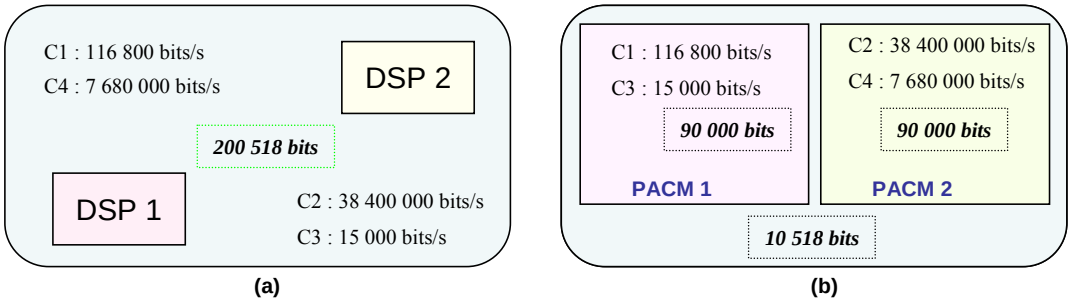


Figure 50 : (a) Contraintes et données à gérer pour une exploration classique
(b) Contraintes et données à gérer pour une exploration de deux PACM

Théoriquement, nous pouvons déjà remarquer les gains possibles au niveau de la conception grâce à une architecture multi-PACM. Ce n'est pas toujours possible d'avoir une partition qui correspond à la partition théorique, soit une partition qui a toutes les métriques proches de 1. En effet, comme nous l'avons constaté lors de l'analyse de la mémoire de l'émetteur UMTS, il est impossible d'équilibrer la mémorisation entre les PACM. Par conséquent, la métrique mémoire est toujours proche de zéro. Donc, le but de la pré-exploration est de trouver une partition qui s'approche autant que possible de la partition théorique.

	Solution	EDE	DEIC	CIC	IA	MEM	Tc	Surface
1	11111112111122	0,906	0,808	0,923	0,791	0,117	0,199	1,252
2	11111111212211	0,914	0,801	0,769	0,943	0	0	1,037
3	11121121222211	0,972	0,790	0,692	0,921	0,117	0,002	1,032
4	11111112222211	0,592	0,611	0,769	0,930	0,058	0	0,801
5	11111111212222	0,046	0,801	0,769	0,875	0,117	0	0,715
6	11122212211211	0,997	0,407	0,538	0,921	0,058	0,199	0,594

Tableau 7 : Les solutions de la pré-exploration de l'émetteur UMTS en 2 PACM

L'outil d'exploration élaboré a permis de trouver un ensemble de solutions assez riche en compromis. Six solutions ont été trouvées par l'algorithme génétique. Ces solutions sont affichées dans le Tableau 7. Pour chaque solution, son numéro, ses métriques ainsi que la surface couverte par les métriques sont affichés. Les métriques des différentes solutions sont affichées dans la Figure 51. Un récapitulatif des différentes métriques est illustré dans cette même figure.

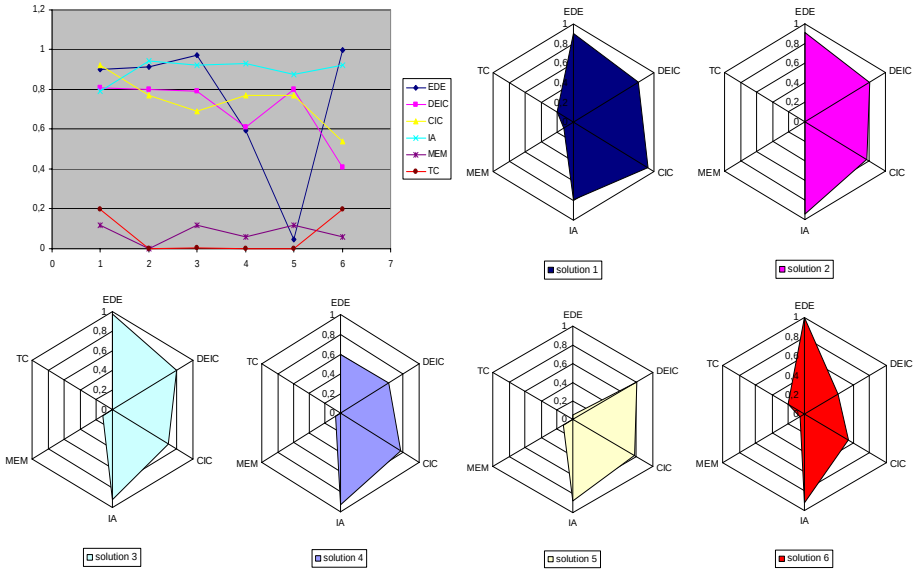


Figure 51 : Les espaces des métriques des différentes solutions

Sur la figure récapitulant les métriques des différentes solutions, nous remarquons que les métriques **Tc** (en rouge) et **MEM** (violet) sont assez faibles pour toutes les solutions trouvées. Par contre la métrique **EDE** (en bleu foncé) est assez proche de 1 pour certaines solutions et assez faible (proche de 0) pour d'autres. Donc la solution 5, pour laquelle la métrique **EDE** est presque 0, est retirée de l'ensemble des solutions par le concepteur. Nous remarquons aussi que les métriques **CIC** (en jaune) et **DEIC** (en rose) sont supérieures à 0,6 sauf pour la solution 6 où ces deux métriques sont inférieures à 0,6. Par conséquent, cette solution est retirée aussi de l'ensemble des solutions. La solution 4 est également retirée parce que la valeur de la métrique **EDE** est beaucoup plus petite que la valeur de cette métrique pour les autres solutions. Les solutions 1, 2 et 3 constituent 3 partitions adéquates pour l'émetteur UMTS parce qu'elles optimisent assez bien la communication et l'analyse de l'identité. Les contraintes et la mémoire ne peuvent pas être optimisées. Leurs métriques respectives sont proches de 0 pour ces 3 solutions. Toutefois, seule la solution 1 optimise au mieux la distribution des contraintes. De plus, la solution 1 est la seule des 3 qui optimise en même temps la distribution des contraintes et la distribution de la mémoire. Donc, comme le concepteur n'a pas de préférence particulière, nous pouvons considérer la solution 1 comme partition en deux PACM adéquate pour l'émetteur UMTS.

La solution adoptée est plus adéquate que la solution trouvée par l'exploration manuelle (Figure 49.d). Pour la comparer avec la solution trouvée par l'algorithme génétique, nous traçons les deux espaces sur la même figure (Figure 52). Nous remarquons que la solution trouvée par l'algorithme génétique améliore mieux la plupart des métriques. En effet, les métriques **EDE**, **CIC**, **Tc**, **MEM** de la solution trouvée par l'outil d'exploration sont plus proches de 1 que celles de la solution trouvée manuellement par le concepteur. L'outil d'exploration a été assez performant pour trouver une solution plus optimale, assez rapidement, que celle trouvée manuellement.

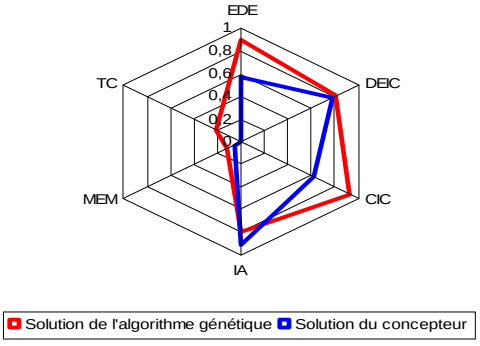


Figure 52 : Comparaison de l'exploration manuelle avec l'algorithme génétique

2.4.2. Pré-exploration en 3 PACM

Nous considérons l'implémentation de l'émetteur UMTS sur une architecture comportant trois ressources logicielles. L'exploration avec un flot classique, comme le résume la Figure 53.a, pour une telle architecture sera basée sur trois DSP et doit gérer plus de 200 kbits. L'objectif idéal d'une pré-exploration en 3 PACM pour l'émetteur UMTS, est de trouver une partition (Figure 53.b) qui permet de séparer les contraintes (C1 et C3 dans un PACM, C2 dans un deuxième PACM et C4 dans un troisième PACM), d'avoir 60 000 bits de données échangés dans chaque PACM et environ 10 000 bits entre les PACM. L'avantage d'une telle partition est, qu'au lieu de concevoir des ressources de communication capables de gérer des contraintes avec 38 Mbits/s de différence, il suffit de gérer des contraintes avec une différence de seulement 100 kbits/s au maximum. Aussi, au lieu de gérer et mettre en œuvre les ressources de communication nécessaires pour 200 518 bits de données échangées, il est question de gérer et de concevoir les ressources de communication de 4 échanges de données dont le plus grand est de 60 000 bits, soit 1/3 de la quantité gérée par une exploration classique.

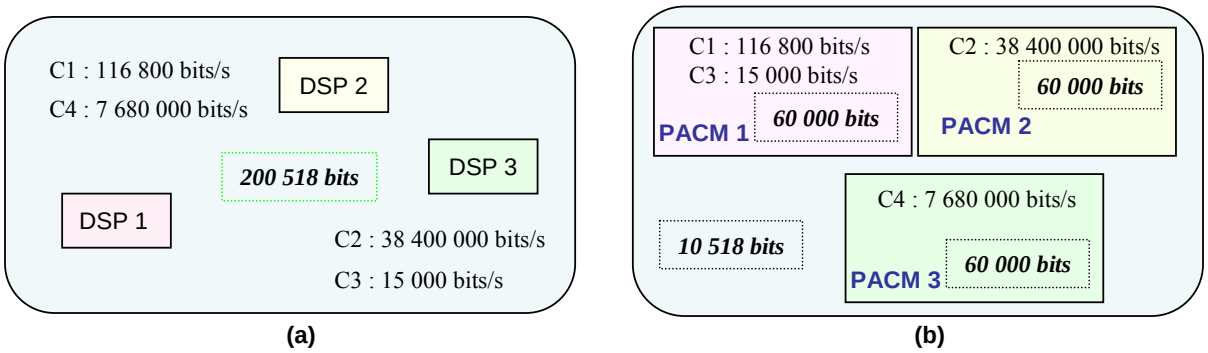


Figure 53 : (a) Contraintes et données à gérer pour une exploration classique
(b) Contraintes et données à gérer pour une exploration de trois PACM

L'exploration de l'espace des solutions pour l'émetteur UMTS en 3 PACM a permis de trouver quatre solutions. Ces solutions sont affichées dans le Tableau 8. Pour chaque solution, son numéro, ses métriques ainsi que la surface couverte par les métriques sont affichés. Les métriques des différentes solutions sont affichées dans la Figure 54. Un récapitulatif des différentes métriques est illustré dans la même figure.

	Solution	MEM	EDE	DEIC	CIC	IA	Tc	Surface
1	12222222212223	0,088	0	0,801	0,769	1	0,997	1,235
2	12222222222223	0,058	0	0,802	0,846	1	0	0,762
3	12222222212222	0,029	0	0,993	0,846	0,947	0,199	0,918
4	12222322212223	0,088	0	0,789	0,615	1	0,997	1,093

Tableau 8 : Les solutions de la pré-exploration de l'émetteur UMTS en 3 PACM

Sur la figure récapitulant les métriques des différentes solutions, nous remarquons que la métrique Tc est bien optimisée pour les solutions 1 et 4 alors qu'elle est proche de 0 pour les solutions 2 et 3. Les autres métriques ont presque les mêmes valeurs pour toutes les solutions. Les solutions 2 et 3 sont donc retirées de l'ensemble des solutions. Les valeurs des métriques des solutions 1 et 4 sont presque équivalentes sauf la valeur de la métrique CIC de la solution 4 qui est légèrement inférieure à celle de la solution 1.

La solution 1 contient 1 seule tâche dans le PACM 3. La solution 2 a une seule de ces tâches dans le PACM 3. La solution 3 n'a aucune tâche dans le PACM 3. La solution 4 a seulement 2 tâches dans le PACM 3. De plus, l'ensemble des solutions à une mauvaise distribution des données et de la mémoire dans les différents PACM. Etant donné le peu de parallélisme entre les tâches de l'émetteur UMTS, il semble qu'il soit préférable d'opter pour une solution à deux PACM.

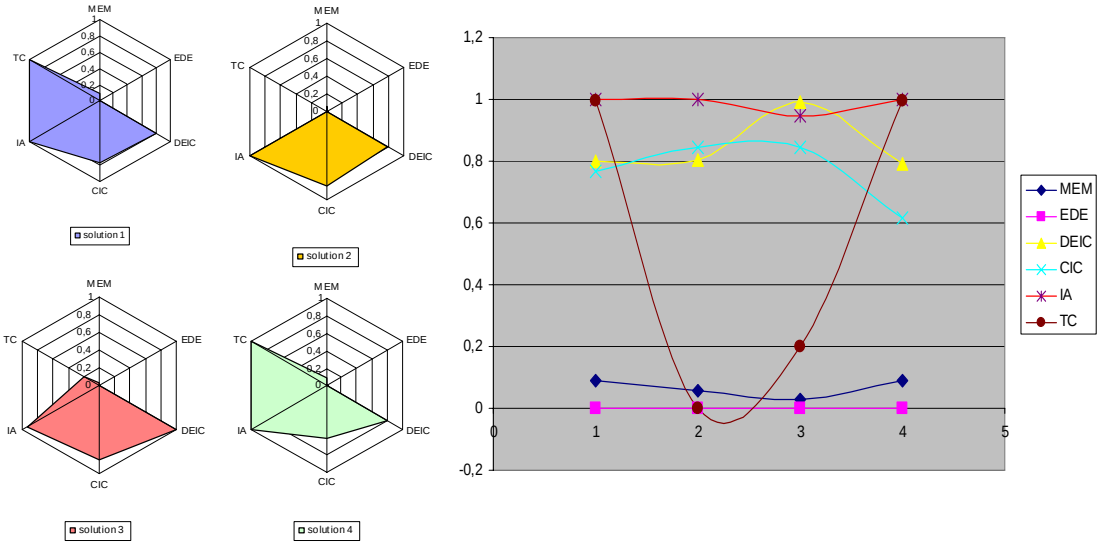


Figure 54 : Les espaces des métriques des différentes solutions

2.5. *IMPLEMENTATION DE L'EMETTEUR UMTS*

L'émetteur UMTS a été implémenté par Mitsubishi [Moy 2001]. La plateforme matérielle utilisée pour l'implantation de l'émetteur UMTS, décrite dans [Rouxel 2006], est constituée d'un ensemble de cartes Pentek. Il s'agit d'une carte mère, modèle 4290, composée de 4 DSP C6X de chez Texas Instrument disposant de mémoires externes de nature différente (SBSRAM, SDRAM, DPSRAM), sur laquelle est interfacée une carte fille, modèle 6250, composée de 2 FPGAs Virtex II XC2V3000 de chez Xilinx. A cette même carte mère sont

ajoutés deux autres modules Pentek d'émission 6229 et de réception 6216. Les possibilités d'utilisation des ressources présentes sont importantes car chaque DSP peut s'adresser aux autres ainsi qu'à n'importe quel FPGA, même si pour cela il doit passer pour 3 d'entre eux par l'intermédiaire d'un DSP différent à chaque fois. La communication entre les ressources pour cette carte peut s'avérer très coûteuse en temps et latences. Sur la plateforme les tâches SRC et CT_SCR_code sont implémentées en matériel. La spécification de la partie de l'émetteur UMTS à implémenter est illustrée dans la

Figure 55.

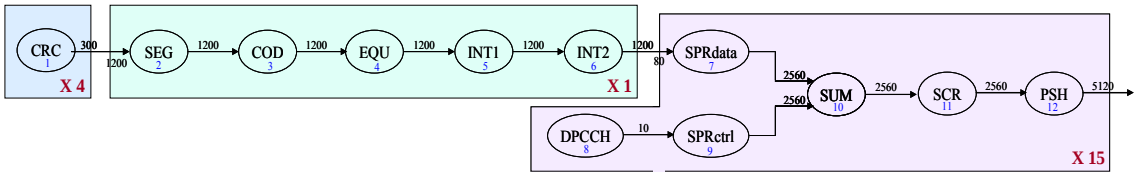


Figure 55 : Schéma de spécification de l'émetteur UMTS

L'architecture visée par les concepteurs est une architecture surtout logicielle. Une première solution purement logicielle a été essayée. L'analyse faites par [Rouxel 2006] a démontré que cette architecture ne satisfait pas la contrainte de temps. Les communications dans les PACM pour les architectures logicielles sont inexistantes vu que le temps des communications au sein d'un DSP est nul. Afin de trouver une partition qui répond aux objectifs du concepteur, une exploration est établie en se basant sur toutes les métriques sauf **EDE**. La métrique EDE est enlevée parce que le temps de communication dans un DSP peut être considéré comme nul. La solution trouvée est illustrée surs la Figure 56.

Solution	DEIC	CIC	IA	TC	MEM
111111122221	0,523	0,818	1	1	0,79

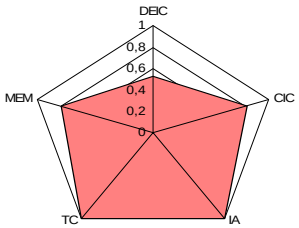


Figure 56 : L'espace des métriques de la solution trouvée

L'architecture choisie par Mitsubishi [Rouxel 2006] est d'implémenter les tâches de 1 à 7 (

Figure 55) sur un DSP, implémenter les tâches de 8 à 11 sur un deuxième DSP et la tâche 12 (PSH) en matériel. Cette architecture correspond bien à une partition "111111122221". Cette partition est celle trouvée par l'outil d'exploration. Cette expérience montre bien que l'espace des solutions trouvé par l'outil d'exploration GAMA[2] contient une architecture adéquate pour l'émetteur UMTS.

Les différents résultats relevés sur l'émetteur UMTS ont bien montré que les métriques sont capables d'anticiper les performances de l'architecture afin de guider l'exploration de l'architecture. Ainsi les métriques sont capables, à haut niveau, de juger de la qualité de la communication et des mémoires qui influent sur le temps, la surface et la consommation de l'architecture. L'exploration permet de trouver un groupe de partitions qui répondent aux attentes du concepteur. La méthodologie permet l'interaction à deux niveaux avec le concepteur. D'un côté, le concepteur peut enrichir la spécification avec les informations sur le graphe de spécification. D'un autre côté, il guide l'exploration en choisissant les métriques qui guident vers le type d'architecture de son choix (en choisissant les métriques utilisées pour l'exploration par exemple).

3. SYSTEME DE DECODAGE DE SIGNAUX AC3

La technologie de codage des signaux AC3 est retenue dans le cadre de la norme américaine de diffusion hertzienne de programme TV, TVHD et DVD pour la reproduction sonore multi-piste.

La technique AC3 est adaptée au codage et à la compression de cinq canaux sonores. Elle offre des voies gauche, droite et centrale à l'avant de l'auditeur ainsi que deux voies, droite et gauche à l'arrière (effet *surround*). Ces cinq voies sont codées numériquement sur une large bande passante (de 3 Hz à 20 KHz) et compressées de manière indépendante. Un sixième canal contient des informations additionnelles dans le domaine des fréquences très basses (de 3 Hz à 120 Hz) et permet de renforcer l'ambiance sonore des cas particuliers et adaptés au cinéma (explosions par exemple). Cette dernière voie, couvrant une faible bande de fréquence, est à l'origine du terme 5.1 désignant le type de codage de la norme AC3.

Le débit est défini par les 6 canaux échantillonnés à 48 KHz sur 18 bits soit 5,184 Mbits/s. Le codeur transforme ce débit en 384 Kbits/s. Sur chaque canal, 256 nouveaux échantillons sont utilisés à chaque itération.

3.1. PRESENTATION DE L'APPLICATION

Le schéma fonctionnel du décodeur AC3 est celui proposé par [Bianco 1999b]. Ce schéma concerne un décodeur AC3 configuré en trois canaux avec un canal couplant. Les hautes fréquences au sein de chaque canal contiennent des informations redondantes. Les sous bandes correspondant aux fréquences élevées sont couplées dans un canal unique appelé canal couplant.

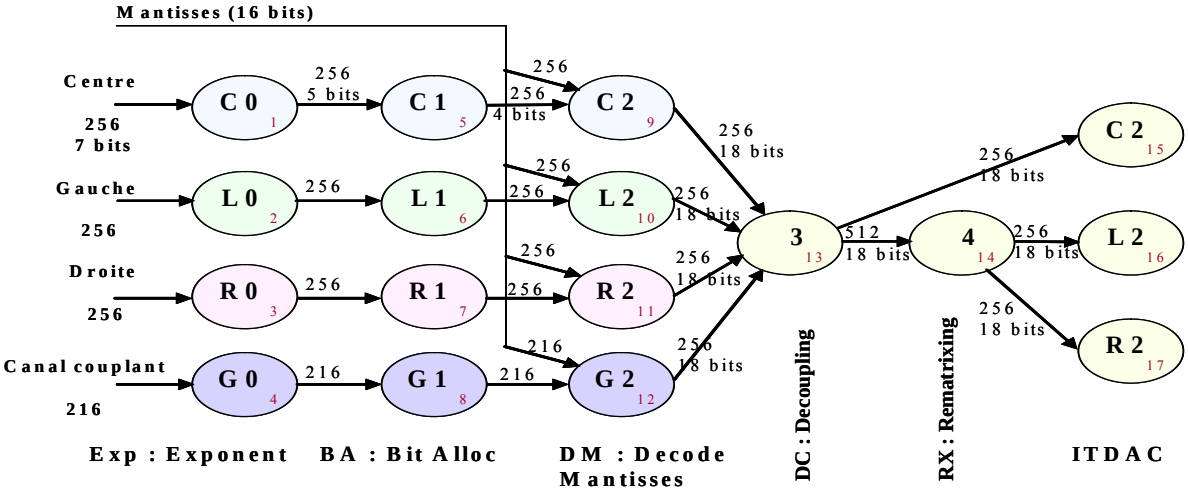


Figure 57 : Schéma de spécification du décodeur AC3

Le schéma fonctionnel du décodeur AC3 est illustré sur la Figure 57. Il présente les différentes tâches mises en œuvre dans l'émetteur ainsi que leurs interactions. Le décodeur commence par calculer les exposants de l'algorithme de décodage (Tâche EXP). Ensuite, le décodeur recalcule la stratégie d'allocation des bits de codage réservés à la représentation de chacune des mantisses des échantillons (Tâche BA). Une fois les exposants et la stratégie d'allocation connus, les mantisses sont reconstruites par la tâche DM. Les informations du canal couplant sont ensuite redistribuées dans les canaux réels (Tâche DC). Une compression particulière est réalisée sur les canaux gauche et droite à reconstituer (tâche RX). Enfin, les valeurs des signaux, dans le domaine temporel, sont recalculées à partir des valeurs obtenues dans le domaine spectral (tâche ITDAC).

3.2. GRAPHE DE SPECIFICATION

Nous présentons ci-après la spécification utilisée pour le codeur des signaux AC3. Le graphe de spécification se compose de 17 nœuds représentant les tâches de l'application et 16

arcs représentant les interactions entre les tâches. La fréquence d'exécution des tâches, le nombre et la taille des données échangées, ainsi que la fréquence des échanges sont intégrés dans le graphe de spécification à partir de la spécification fonctionnelle de l'application (Figure 57). Le concepteur n'a considéré aucun débit contrainte pour cette application. Toutes les déclarations ***Constraint_throughput*** du graphe de spécification sont égales à 0. Aucune analyse des tâches par des outils n'a été effectuée pour cette application. Par conséquent, la variable **LM_size** est estimée à zéro pour toutes les tâches. La tâche 13 **DC** reçoit des données de 4 tâches différentes. Nous considérons que les échanges seront asynchrones et ne passeront pas par une mémorisation. Alors, la déclaration **Memorisation** de la spécification de toutes les tâches de l'application AC3 est égale à zéro.

L'architecture que nous ciblons est constituée de processeurs OAK que nous notons **PE1** et de processeurs ARM [Arm] que nous notons **PE2**. Afin d'enrichir l'expérimentation, nous supposons que le concepteur dispose d'un processeur spécifique que nous notons **PE3**. Nous supposons que le concepteur n'a pas le code pour exécuter les tâches de l'application sur ce processeur et que leurs performances seront très mauvaises. Tenir compte de ce processeur pendant la spécification nous permet de tester deux points. Le premier est que l'approche que nous proposons est capable de conclure qu'il faut éviter d'intégrer ce processeur dans l'architecture. Le deuxième est que notre approche est flexible. Dans le cas où le concepteur décide d'ajouter ultérieurement à l'application des tâches qui sont optimisées par **PE3**, il suffit d'ajouter les spécifications de ces tâches et celles des interactions avec les autres tâches de l'application, sans être obligé de modifier le vecteur d'analyse d'identité des autres tâches pour prendre en compte **PE3**.

Les tâches **Exp** (tâches 1 à 4), **DC** (tâche 13), **RX** (tâche 14) et **ITADAC** (tâches 15 à 17) sont considérées par le concepteur comme ayant des performances moyennes en cas d'exécution par le processeur ARM. Le concepteur donne alors à leur analyses d'identité par rapport à **PE2** une valeur de 0,5. Les performances de ces tâches sont optimales en cas d'exécution par le processeur OAK. Le concepteur donne alors à **LIA(t,PE1)** une valeur de 0,8. La valeur 0,2 est associée à l'analyse d'identité pour **PE3**. Les tâches **DM** (tâches 9 à 12) sont considérées par le concepteur comme ayant des performances satisfaisantes en s'exécutant sur **PE2** et des performances moyennes en s'exécutant par **PE1**. Par conséquent, le vecteur d'identité de ces tâches est (0,5 0,7 0,2). Nous supposons que pour les tâches **BA** (tâches 5 à 8), le concepteur ne dispose d'aucune information relative aux performances de ces tâches. Alors, nous considérons que ces tâches ont des performances moyennes en cas

d'exécution avec l'un des deux processeurs **PE1** et **PE2**. Le vecteur d'analyse d'identité de ces tâches est (0,5 0,5 0,2).

3.3. EXPLORATION POUR UNE ARCHITECTURE EN DEUX PACM

Afin de valider notre approche, nous appliquons notre approche sur l'application AC3 et nous comparons le résultat avec celui de CODEF (Figure 58). En premier lieu, l'espace d'architectures de l'application AC3 est exploré par l'outil CODEF pour chercher une architecture à deux processeurs. La version de CODEF que nous utilisons ne tient compte que des performances temps et surface. L'outil cherche les solutions qui satisfont la contrainte de temps et qui optimisent la surface. La contrainte de temps considérée pour l'application AC3 est de 1500 µs.

En second lieu, nous appliquons notre approche d'exploration pour l'application AC3 pour la même contrainte de temps (1500µs). Nous précédons l'outil CODEF par l'étape de pré-exploration effectuée par l'outil GAMA[2]. L'analyse des métriques et l'algorithme génétique génèrent une partition des tâches en deux PACM. Un processeur parmi les trois est affecté à chaque PACM. Chaque tâche affectée à un PACM ne peut être exécutée que par le processeur qui lui est associé ou en matériel si une description matérielle est disponible dans la bibliothèque de CODEF. Par conséquent, les descriptions autres que ces deux dernières sont enlevées de la bibliothèque de CODEF, ce qui permet d'alléger les coûts de l'exploration. Aussi, l'espace d'exploration est plus petit parce que les architectures comprenant une exécution des tâches avec d'autres processeurs que celui affecté au PACM auquel les tâches appartiennent ne figurent plus dans l'espace d'exploration.

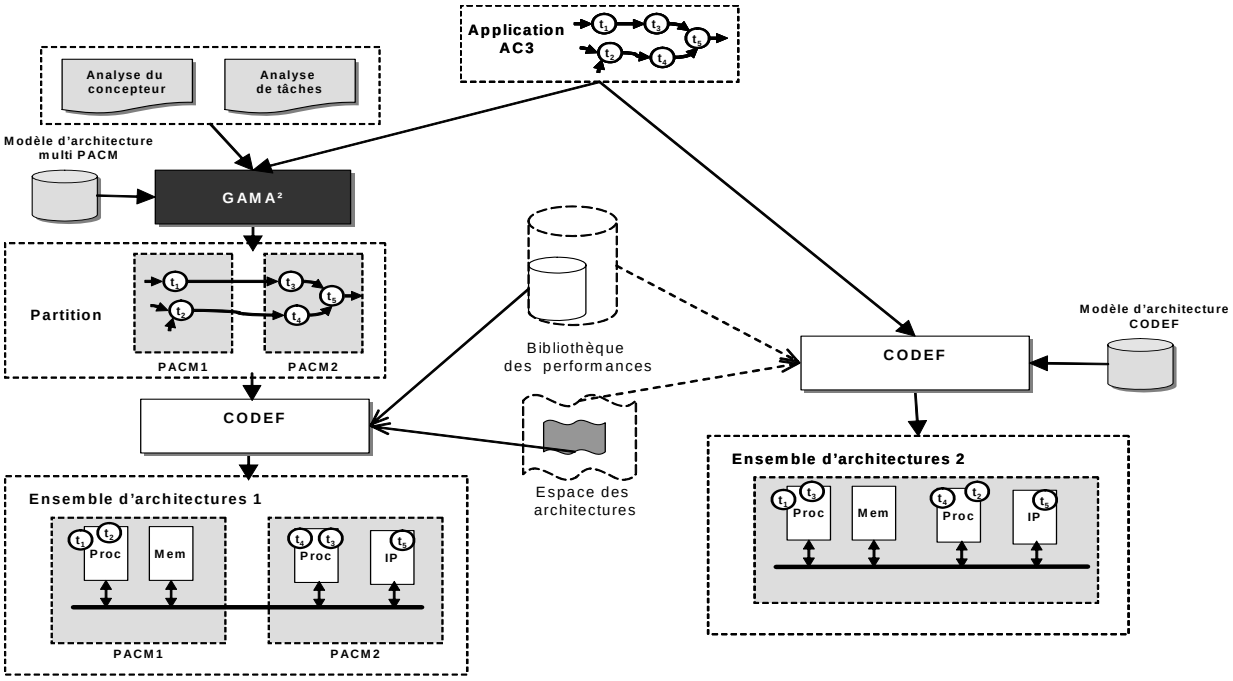


Figure 58 : Expérimentations pour l'application AC3

3.4. EXPLORATION PAR CODEF

La spécification de l'application AC3 est fournie à CODEF ainsi que la bibliothèque des performances des tâches. Les possibilités d'exécution des différentes tâches sont résumées dans le Tableau 9. Toute les tâches ont la possibilité d'être exécutées par les processeurs OAK et ARM. Seules les tâches BA ont la possibilité d'être exécutées en matériel.

Tâches	**Possibilités d'exécution**
Exp_C0,Exp_R0, Exp_L0, Exp_G0	➢ Oak ➢ ARM
BA_C1, BA_R1, BA_L1, BA_G1	➢ OAK ➢ HW_BitAlloc ➢ ARM
DM_C2, DM_R2, DM_L2, DM_G2	➢ Oak ➢ ARM
RX	➢ Oak ➢ ARM
ITDAC_C5 , ITDAC_L5, ITDAC_R5	➢ Oak ➢ ARM
DC	➢ Oak ➢ ARM

Tableau 9 : Possibilités d'exécution des tâches

L'outil CODEF a généré un ensemble de solutions. La courbe du temps en fonction de la surface a été prélevée (Figure 59). L'outil, ayant pour but d'optimiser la surface, a permis de trouver une solution ayant une surface optimisée (Figure 59-1). Toutefois, le temps de cette architecture est assez élevé mais respecte la contrainte temps. Par contre, l'architecture (Figure

59-2) est bien optimisée en temps mais nécessite une surface assez importante. L'architecture (Figure 59-3) semble être une architecture qui équilibre l'optimisation temps et surface.

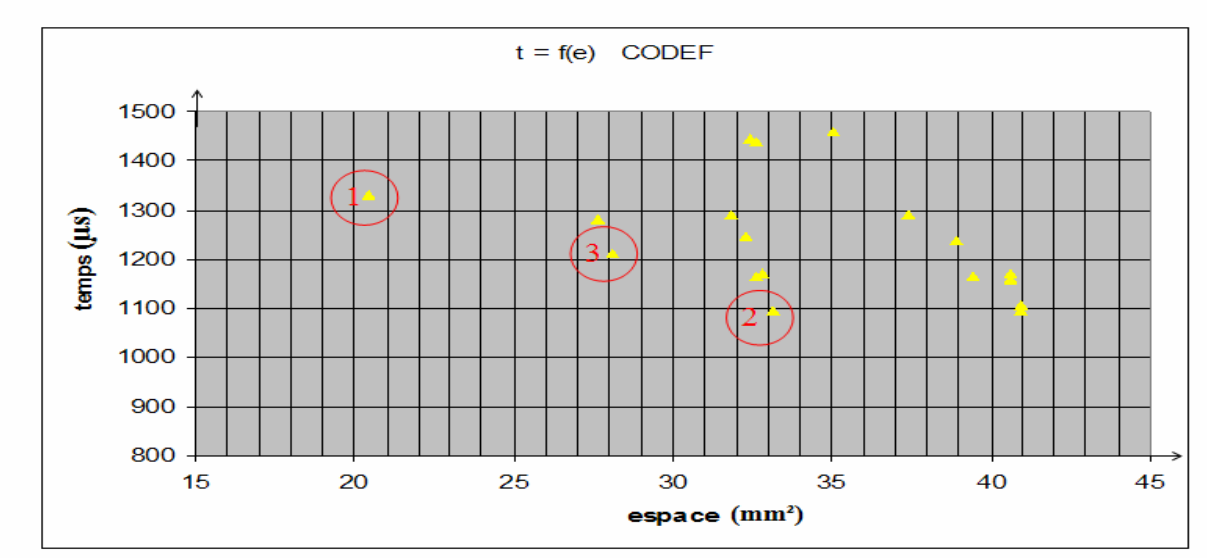


Figure 59 : Architectures générées par CODEF

Les solutions optimales au sens de Pareto sont les solutions 1, 2 et 3. La première est à base d'un ARM et d'un OAK sans l'utilisation d'aucune implémentation matérielle. La deuxième est à base de deux OAK et un ARM. La troisième architecture utilise deux ARM et un OAK.

3.5. PRE-EXPLORATION PAR GAMA² ET CODEF

Le graphe de spécification de l'application AC3 est fourni à l'outil GAMA^2. Toutes les métriques sont sélectionnées pour l'exploration. Le nombre de PACM est fixé à deux. Les paramètres de l'algorithme génétique sont sélectionnés comme suit:

- ✓ Nombre d'individus : 50
- ✓ Nombre de générations : 100 000
- ✓ Nombre de croisements : 23
- ✓ Nombre de mutations : 4
- ✓ Kmax : 3

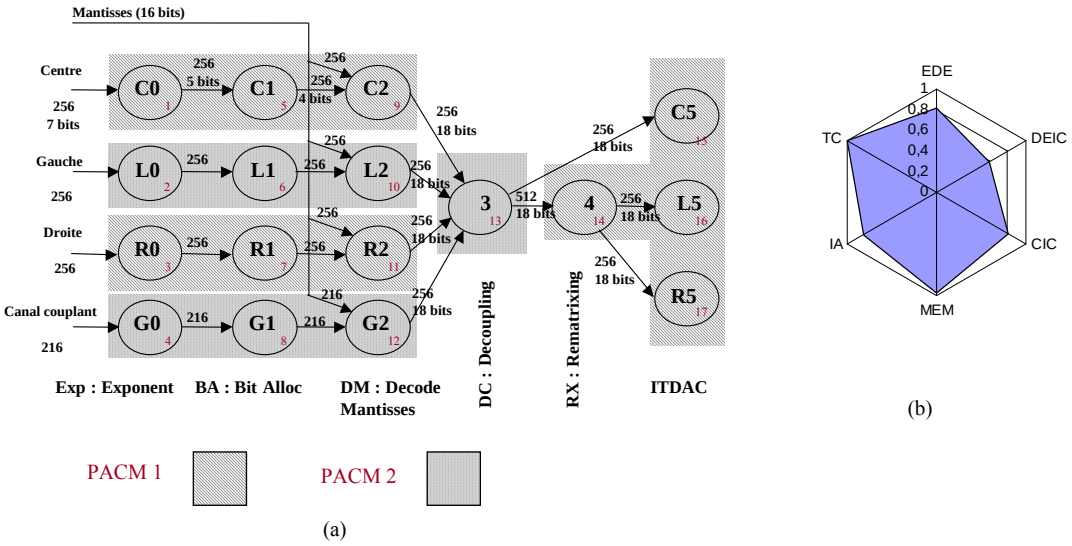


Figure 60 : Partition en deux PACM de l'application AC3

La partition générée par l'outil GAMA^2 est composée de 9 tâches dans le PACM 1 et 8 tâches dans le PACM 2 (Figure 60.a). Le processeur OAK est affecté au PACM1 alors que le processeur ARM est affecté au PACM 2. Bien qu'il n'y ait pas de métrique spécifique au parallélisme, la solution trouvée respecte un certain degré de parallélisme. En effet, elle permet un parallélisme entre les trois canaux et le canal couplant. Toutefois, le parallélisme entre les trois tâches ITDAC n'est pas favorisé par cette architecture. Ceci est expliqué par le fait que l'exécution de ces tâches est effectuée plutôt par le processeur OAK qui est affecté au PACM 1. L'architecture générée optimise à la fois les communications, le partage de mémoire, la distribution des contraintes et l'exécution des tâches. En effet, toutes les métriques sont supérieures à 0,8 sauf EDE qui est presque de 0,6.

La partition générée par l'outil GAMA^2 permet de supprimer des possibilités d'exécution des tâches. Dans le Tableau 10, nous résumons les différentes possibilités d'exécution de chaque tâche de l'application AC3 sans et avec l'outil GAMA^2. Dans la colonne "sans GAMA^2" figure l'ensemble des possibilités de toutes les tâches. Ces possibilités représentent l'espace d'architectures que CODEF (sans GAMA^2) doit explorer. La colonne "avec GAMA^2" présente les possibilités d'exécution des tâches que CODEF (précédé par GAMA^2) doit explorer. Nous remarquons qu'il y a moins de possibilités. Ceci s'explique par le fait qu'une tâche affectée à un PACM ne peut pas être exécutée par les processeurs qui ne sont pas affectés à ce PACM. Par exemple, la tâche exp_C0 ne peut pas être exécutée par le processeur ARM parce que l'outil GAMA^2 l'a affectée au PACM 1 auquel est affecté le processeur OAK.

Tâches	Possibilités d'exécution	
	Sans GAMA^2	Avec GAMA^2
Exp_C0	➢ OAK ➢ ARM	➢ OAK
Exp_L0	➢ OAK ➢ ARM	➢ ARM
Exp_R0	➢ OAK ➢ ARM	➢ OAK
Exp_G0	➢ OAK ➢ ARM	➢ ARM
BA_C1	➢ OAK ➢ ARM ➢ HW_BitAlloc	➢ OAK ➢ HW_BitAlloc
BA_L1	➢ OAK ➢ ARM ➢ HW_BitAlloc	➢ ARM ➢ HW_BitAlloc
BA_R1	➢ OAK ➢ ARM ➢ HW_BitAlloc	➢ OAK ➢ HW_BitAlloc
BA_G1	➢ OAK ➢ ARM ➢ HW_BitAlloc	➢ ARM ➢ HW_BitAlloc
DM_C2	➢ OAK ➢ ARM	➢ OAK
DM_L2	➢ OAK ➢ ARM	➢ ARM
DM_R2	➢ OAK ➢ ARM	➢ OAK
DM_G2	➢ OAK ➢ ARM	➢ ARM
DC	➢ OAK ➢ ARM	➢ ARM
RX	➢ OAK ➢ ARM	➢ OAK
ITDAC_C5	➢ OAK ➢ ARM	➢ OAK
ITDAC_L5	➢ OAK ➢ ARM	➢ OAK
ITDAC_R5	➢ OAK ➢ ARM	➢ OAK

Tableau 10 : Possibilités d'exécution des tâches après analyse par GAMA^2

Pour l'application AC3, il n'y a que 21 possibilités d'exécution (Tableau 10, colonne "avec GAMA^2") avec l'outil GAMA^2, au lieu de 38 possibilités d'exécution des tâches au total (la somme de toute les possibilités figurant dans le Tableau 10, colonne "sans GAMA^2"), soit 44,74% de possibilités en moins. Ceci montre que l'espace d'architectures est considérablement réduit. Le coût d'exploration a également diminué puisque nous avons besoin de 17 estimations de performances des tâches en moins.

Toutefois, pour le cas de l'application AC3, plusieurs tâches sont identiques. Par exemple, les tâches Exp-C0, Exp-L0, Exp-R0 et Exp-G0 représentent toutes la tâche ***exponent*** qui est utilisée quatre fois. Il en est de même pour les tâches ***Bit Alloc***, ***Decode Mantisses*** et ***ITADAC***. Au final, il n'y a que 3 possibilités d'exécution à supprimer qui sont

DC, RX et ITDAC en ARM, ce qui réduit quand même l'espace d'architectures. Ceci montre que l'analyse de haut niveau effectuée par l'outil GAMA^2 est capable de réduire l'espace d'architectures et donc les coûts de l'exploration.

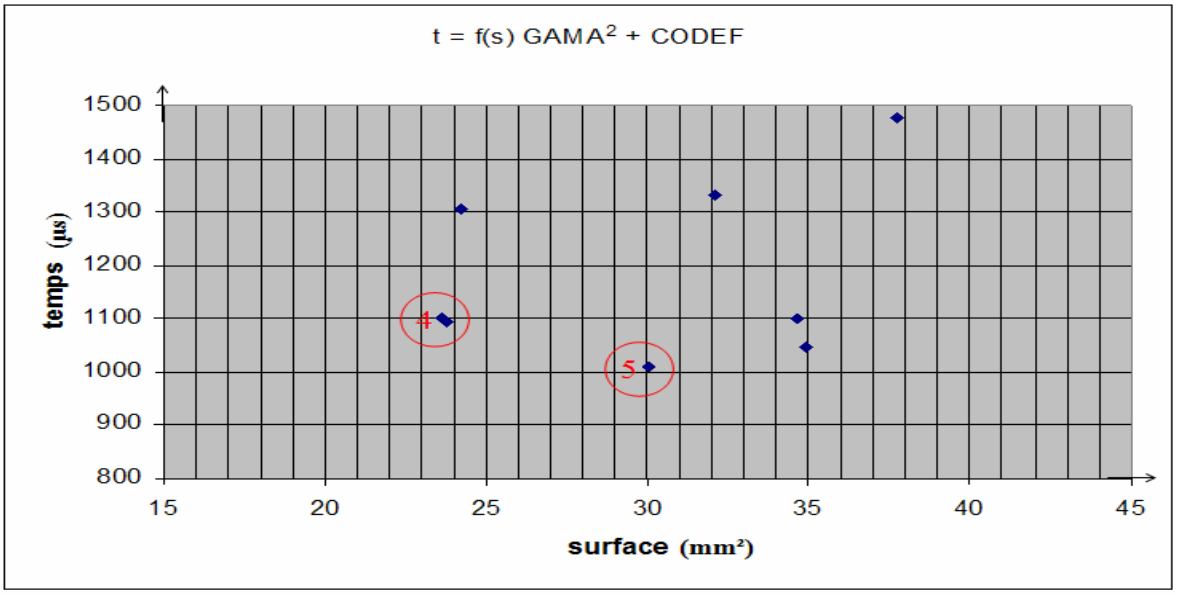


Figure 61 : Architectures générées par CODEF précédé par GAMA^2

L'exploration par l'outil CODEF de l'espace d'architectures, réduit par GAMA^2, a généré un groupe d'architectures dont la courbe de temps en fonction de la surface a été prélevée (Figure 61). La réduction de l'espace d'architectures n'a pas dégradé la qualité des architectures trouvées. Les architectures 4 et 5 sont les architectures optimales au sens du pareto. Elles représentent deux architectures assez optimisées en temps et en surface. Ceci montre que la méthodologie proposée est capable de trouver un ensemble d'architectures satisfaisant les contraintes et optimisant les performances.

3.6. COMPARAISON DES RESULTATS DES DEUX APPROCHES

Nous représentons les architectures trouvées par les deux approches dans l'espace temps en fonction de la surface (Figure 62). Les architectures trouvées par CODEF tout seul sont représentées en triangle. Les architectures trouvées par CODEF précédé par GAMA^2 sont représentées en losange.

De façon générale, les deux approches ont réussi à trouver des architectures optimisées en temps et en surface. L'approche CODEF, qui cherche à optimiser la surface en priorité, a réussi à trouver l'architecture (Figure 62-1) qui a la plus petite surface. La solution disposant du temps le moins élevé (Figure 62-5) est trouvée par l'approche GAMA^2 plus CODEF. Ceci est dû au fait que l'analyse de haut niveau mise en œuvre par l'approche que nous proposons

favorise le parallélisme, l'optimisation des communications et l'exécution des tâches qui optimisent le temps. De façon générale, quatre sur les six architectures optimisées en temps sont trouvées par l'approche que nous proposons. L'outil GAMA[2] a réussi à guider l'exploration faite par l'outil CODEF vers une partie de l'espace d'architectures où se trouvent des solutions optimisées à la fois en temps et en surface (les architectures 4 et 5).

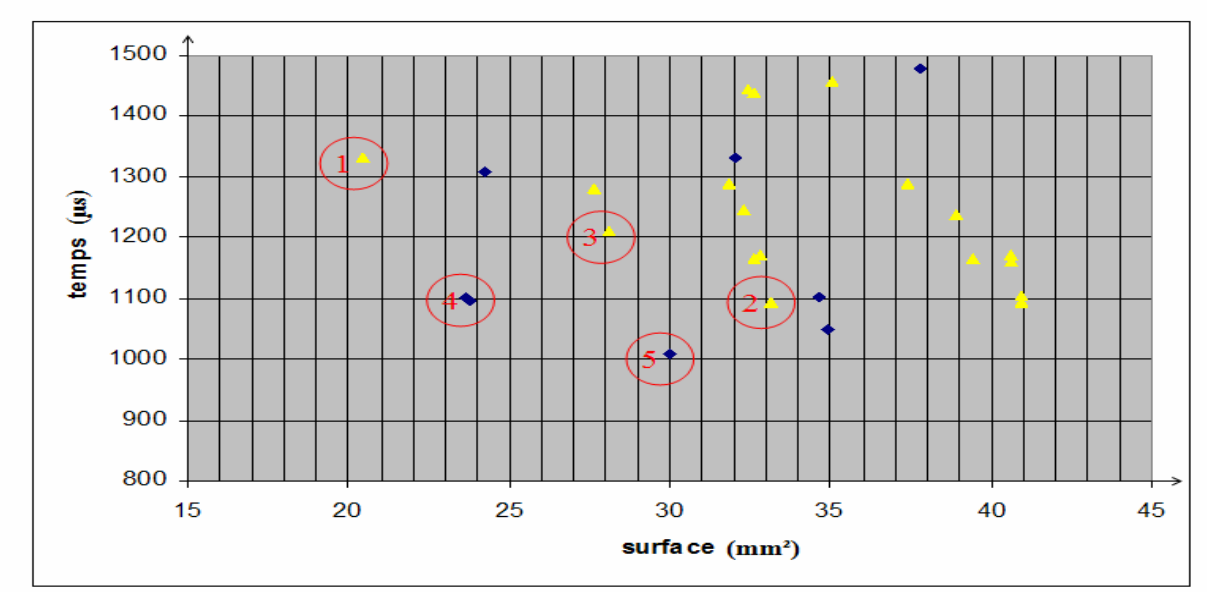


Figure 62 : Les architectures générées par les deux approches

Ces résultats ne mettent pas en cause la qualité des architectures trouvées par CODEF, mais ils permettent de montrer l'efficacité de l'approche que nous proposons à guider l'exploration vers les objectifs que nous nous sommes fixés : optimiser à la fois le temps et la surface. Précéder les approches traditionnelles par l'analyse à haut niveau que nous proposons ne dégrade pas la qualité des architectures trouvées et permet de réduire les coûts et l'espace d'exploration.

4. APPLICATION ICAM

L'objectif de l'application ICAM est que le même système doit être capable de détecter des mouvements et lire des caractères dans une zone fixe (surveillance d'une voie RER par exemple), repérer des véhicules mobiles ou immobiles sur un champ large (portion d'autoroute), analyser une scène pour détecter des objets fixes (panneaux routiers), détecter des objets en mouvement alors qu'il est lui-même en mouvement (piétons, cyclistes…). Développé en interne au CEA, le projet "Icam" ("Intelligent Camera") est consacré à la mise

au point de ce type d'équipement. Basé sur la technologie CMOS, Icam a pour objectif la réalisation d'une caméra de 1 cm^3 fonctionnant dans le visible, pour moins de 100€ !

Nous utilisons cette application pour démontrer que l'outil GAMA² est capable d'analyser et explorer des applications avec un nombre élevé de tâches.

4.1. PRESENTATION DE L'APPLICATION

Nous présentons ci-après les procédures de compression et traitement des données de la caméra Icam utilisées par le CEA.

La Figure 63 illustre les différents blocs qui constituent l'application Icam. Après l'acquisition de 4 images, un premier traitement de ces images et de l'image de fond permet la détection de mouvement. Ce traitement se fait par un bloc de fonctions que nous nommons M.S.S. (Moyenne, Soustraction, Seuillage). Ce traitement s'effectue de la façon suivante :

- ✓ Calcul de la moyenne des 4 images traitées.
- ✓ Soustraction entre l'image moyenne résultante et l'image de fond. Ceci permet de détecter les nouveaux objets qui apparaissent par rapport à l'image de fond.
- ✓ Seuillage de l'image issue de la soustraction pour binariser et ne laisser que les nouveaux objets ayant fait leur apparition.

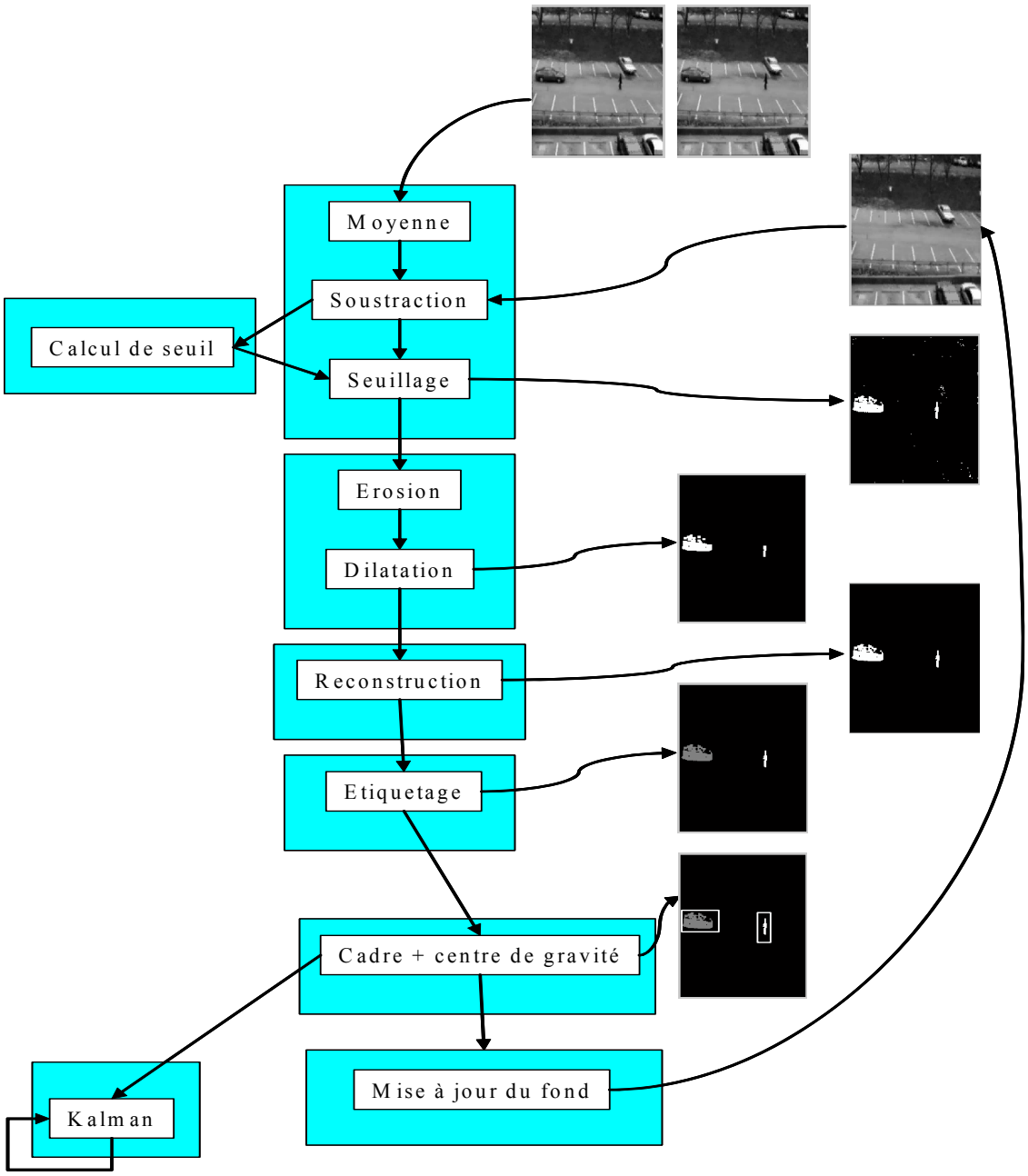


Figure 63 : Schéma fonctionnel de l'application Icam

L'image obtenue est ensuite traitée pour enlever les bruits qu'elle contient. Ceci est réalisé par l'intermédiaire d'une érosion suivie par une dilatation. Ces deux opérations permettent de ne laisser sur un fond noir que les objets en mouvement (les bruits sont supprimés). Malheureusement, suite à ce traitement, les objets perdent quelques détails. Ces derniers sont récupérés par une fonction de reconstruction. Pour résumer, les différents traitements permettent de visualiser sur un fond noir les différents objets en mouvement par rapport à l'image de fond. Ensuite, chaque objet trouvé est étiqueté pour pouvoir l'identifier et le différencier des autres objets en mouvement. Pour chaque objet identifié, son centre de gravité et son cadre sont calculés afin d'identifier sa position sur l'image de fond. Ainsi, il

suffit de remettre les objets à leurs positions sur l'image de fond pour reconstituer les différentes images de la vidéo. Si un objet ne change plus de position pendant un certain temps (une voiture qui vient de se garer dans un parking par exemple), il est considéré comme appartenant à l'image de fond et il y est ajouté. Un filtre prédictif de Kalman est utilisé pour prédire le mouvement des objets. Ceci a pour but d'économiser le traitement de plusieurs images. Par exemple, lors du passage d'une voiture sur une portion d'autoroute, il suffit de prédire le mouvement de la voiture pour reconstituer la vidéo sans effectuer à chaque fois les opérations allant de la moyenne jusqu'au calcul de centre de gravité.

4.2. GRAPHE DE SPECIFICATION

Le graphe de spécification de l'application ICAM tel qu'elle est, comporte 11 nœuds et 15 arcs. Nous supposons pour cette application qu'il est possible d'exécuter les tâches sur quatre processeurs différents. Nous supposons que chaque tâche a des performances peu différentes lors de son exécution par chacun des processeurs et que le concepteur ne peut pas faire un jugement de la qualité de l'exécution des tâches pour chacun des processeurs disponibles. Le vecteur d'affinité de chaque tâche sera alors (1 1 1 1).

La spécification telle qu'elle est décrite dans Figure 63 ne sera pas utilisée pour nos expérimentations parce qu'elle présente peu de tâches. Nous utilisons plutôt deux spécifications en quatre flots et en huit flots qui sont dérivées de la spécification décrite plus haut de ICAM. Ces spécifications sont composées d'une trentaine et une cinquantaine de tâches.

4.3. ICAM QUATRE FLOTS

Afin d'accélérer le traitement des images par ICAM, chaque image est découpée en quatre parties. Le calcul de la moyenne, la soustraction, le seuillage, l'érosion, la dilatation, la reconstruction, l'étiquetage et le calcul du cadre et du centre de gravité sont appliqués à chaque partie de l'image en parallèle. La nouvelle spécification (Figure 64) comporte 31 tâches et 50 arcs et représente un degré de parallélisme plus important. Nous nommons cette spécification ICAM4f. Ci-après, nous utilisons cette nouvelle spécification pour valider la capacité de GAMA2 à analyser ce type d'architecture.

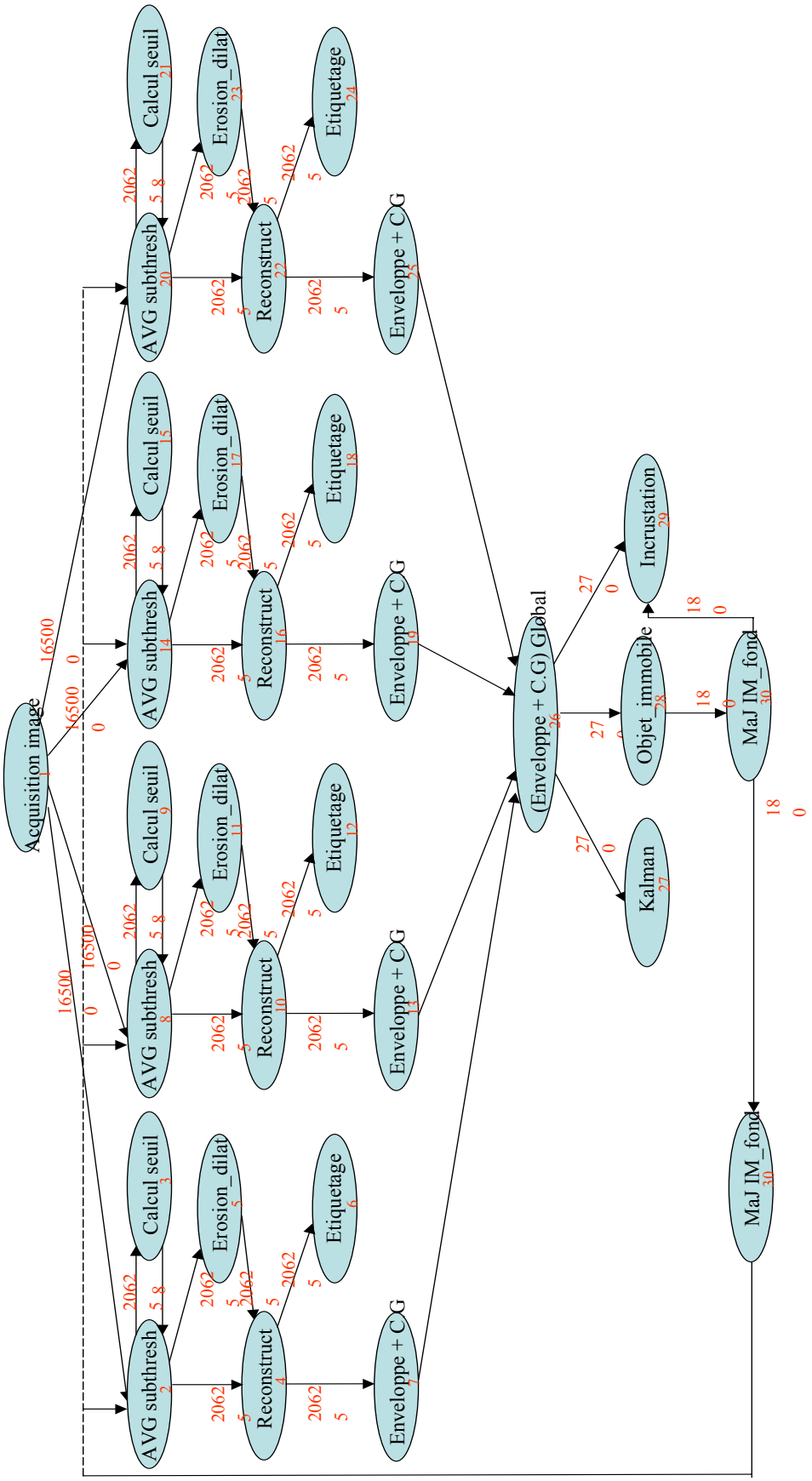


Figure 64 : Graphe de spécification de l'application ICAM4f

Afin de démontrer l'efficacité de GAMA² à explorer l'espace des partitions pour cette application. Nous considérons plusieurs scénarios d'implémentation de ICAM sur une architecture basée sur plusieurs PACM pour varier les expérimentations. Nous considérons des partitions en deux, trois et quatre PACM.

4.3.1. Partition en deux PACM

Nous considérons l'exploration en deux PACM. L'exploration a duré une dizaine de minutes pour une population de 100 individus et pour 100 000 générations. L'outil GAMA^2 a généré huit solutions (Figure 65). L'espace des métriques de chaque solution est représenté pour chacune des solutions. Les architectures trouvées représentent un bon équilibre entre les différentes caractéristiques : communication et mémoire.

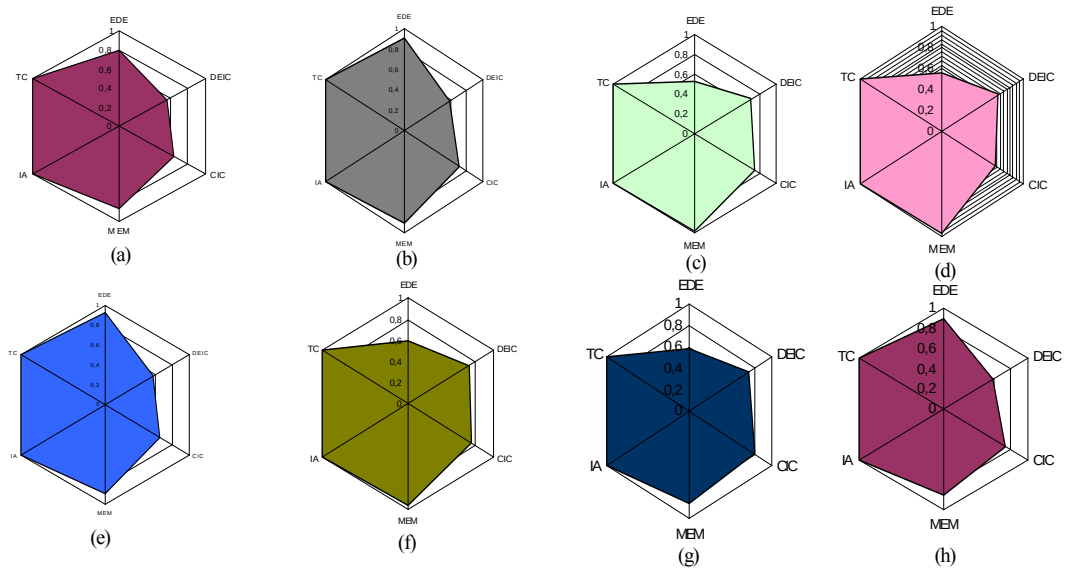


Figure 65 : Solutions pour la partition en deux PACM

Nous prenons l'exemple de l'architecture h pour montrer l'intérêt de la pré-exploration. Nous illustrons dans la Figure 66 la répartition des données et des mémoires pour une exploration directe et une partition en deux PACM. La quantité de données à gérer dans l'exploration directe est à peu près de 5,6 Mbits, alors que pour la partition en deux PACM la plus grande quantité de données échangées est de 2,4 Mbits. La complexité de la gestion des communications est considérablement réduite. De même, nous passons de la gestion de 3,3 Mbits à mémoriser à une quantité maximale de 1,8 Mbits.

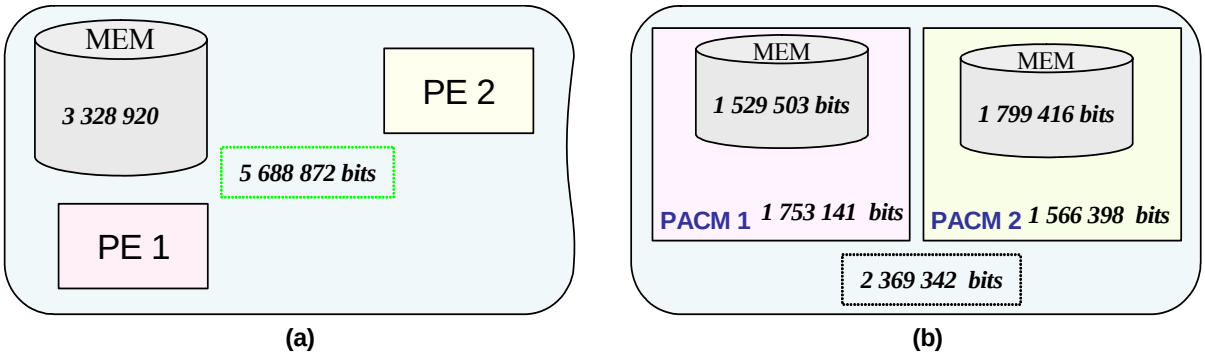


Figure 66 : Exemple de la répartition des données et de la mémoire en deux PACM

4.3.2. Partition en trois PACM

Pour cette expérimentation, une partition en trois PACM est considérée. L'algorithme génétique est basé sur une population de 100 individus et 100 000 générations. L'exploration a aussi duré une dizaine de minutes. Quatre partitions sont générées par l'outil GAMA[2]. Les solutions générées ont un échange de données mal équilibré. En effet, la métrique **EDE** pour chacune des partitions est faible. Une architecture en trois PACM aura alors une gestion de communication complexe. La mémoire et l'échange de données entre PACM sont équilibrés pour l'ensemble des solutions.

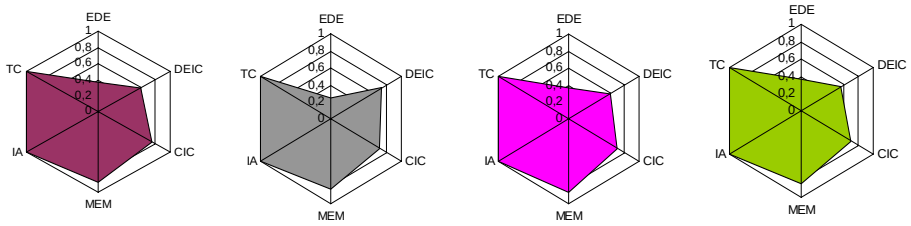


Figure 67 : Solution pour une partition en trois PACM

Afin de montrer l'intérêt de la pré-exploration pour le scénario d'architecture en trois PACM nous détaillons la partition (a). Nous illustrons dans la Figure 68 la répartition des données et des mémoires pour une exploration directe et une partition en trois PACM. Nous remarquons que les quantités de données maximales à échanger et à mémoriser ont été réduites. La quantité de données à gérer est passée de 5,6 Mbits à 2 Mbits pour la partition en deux PACM et la plus grande quantité de données échangées est de 2,4 Mbits. De même, la quantité maximale de données à mémoriser est passée de 3,2 Mbits à 1,4 Mbits.

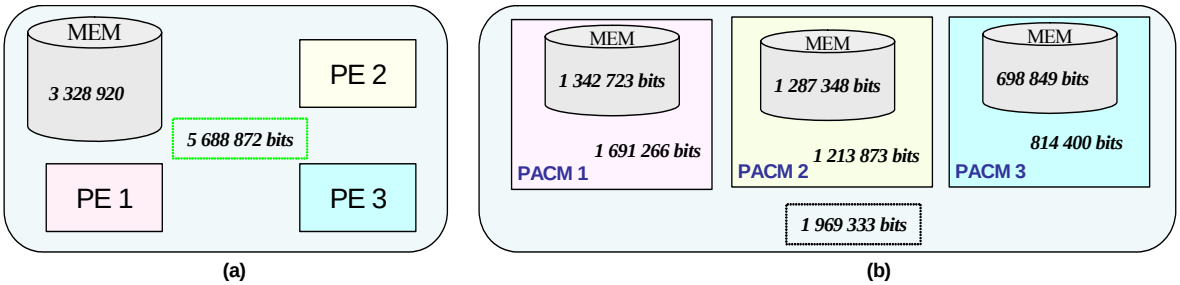


Figure 68 : Exemple de la répartition des données et de la mémoire en 3 PACM

4.3.3. Partition en quatre PACM

Nous considérons une partition en quatre PACM. Nous considérons une population de 100 individus et 200 000 générations. Une quinzaine de minutes ont été nécessaires pour l'exploration. Deux solutions sont générées par l'outil GAMA[2]. Les métriques communication ne sont pas optimisées. Les autres métriques sont assez proches de 1.

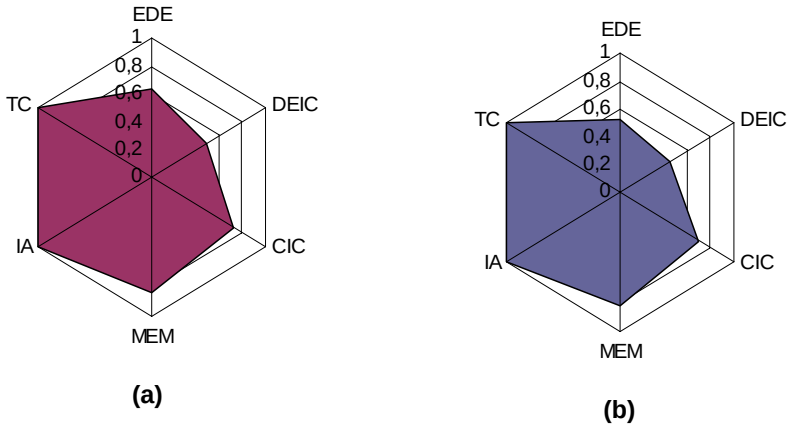


Figure 69 : Solution pour une partition en quatre PACM

Les métriques de la solution (a) de la Figure 69 présentent globalement des métriques plus grandes que l'autre solution, elle est choisie par le concepteur pour être implémentée. La Figure 70 détaille la répartition des données et des mémoires pour cette solution. La distribution des échanges des données est satisfaisante. En effet, la plus grande quantité échangée est de 1,2 Mbits. La quantité des données échangées entre les PACM n'est pas proche de zéro mais elle est considérablement réduite par rapport à la quantité globale des données échangées.

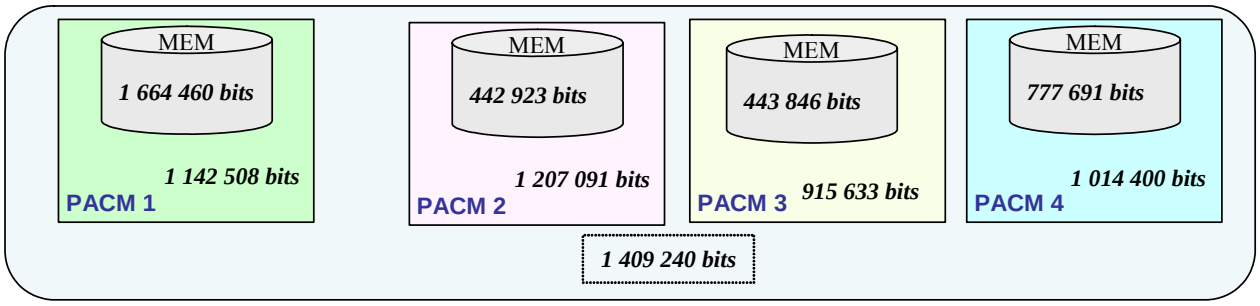


Figure 70 : Exemple de la répartition des données et de la mémoire en 4 PACM

4.4. *ICAM HUIT FLOTS*

Afin de tester l'outil pour une application ayant encore plus de tâches, nous accélérons encore le traitement des images par ICAM. Chaque image est découpée en huit parties. Le calcul de la moyenne, la soustraction, le seuillage, l'érosion, la dilatation, la reconstruction, l'étiquetage et le calcul du cadre et du centre de gravité sont appliqués à chaque partie de l'image en parallèle. La nouvelle spécification comporte 55 tâches et 94 arcs et représente un degré de parallélisme plus important. Nous nommons cette spécification ICAM8f.

Vu le degré de parallélisme (huit flots en parallèle), nous considérons la partition en sept PACM. Cela nous permet en plus de valider GAMA² pour un nombre important de PACM. Les paramètres de l'algorithme génétique sont les suivants :

- ✓ Nombre d'individus : 100
- ✓ Nombre de générations : 500 000
- ✓ Nombre de croisements : 46
- ✓ Nombre de mutations : 8
- ✓ Kmax : 3

La partition choisie par le concepteur parmi celles générées par l'outil d'exploration est détaillée dans la Figure 71. La solution trouvée a des métriques assez élevées (Figure 71-a). La quantité totale des données mémorisées est de 4,4 Mbits et la quantité des données échangées est de 8,8 Mbits (Figure 71-b). La partition trouvée permet de gérer un maximum de données de 1,8 Mbits et de gérer des données mémorisées dont la plus grande valeur est de 825 Kbits.

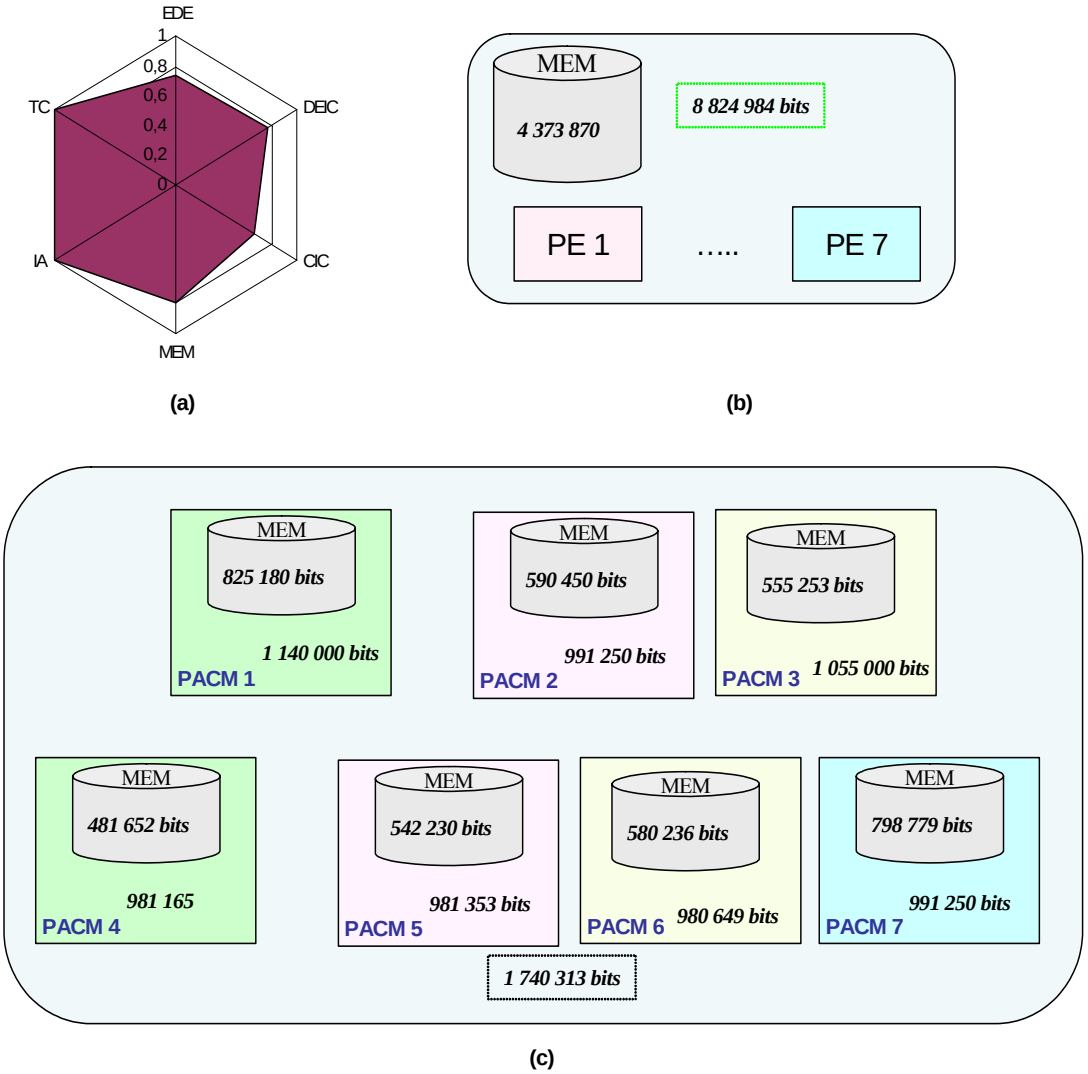


Figure 71 : Exploration en 7 PACM de ICAM8f

4.5. CONCLUSION

L'outil GAMA[2] a été capable d'analyser en peu de temps plusieurs spécifications contenant un nombre important de tâches et un degré de parallélisme élevé pour différents nombres de PACM. Les solutions trouvées sont assez équilibrées pour les différentes configurations que nous avons étudiées.

L'outil GAMA² a été capable de faire face, pour le cas de l'application ICAM8f, à l'analyse d'une application de 55 tâches et de 94 arcs. L'application contient plus 8 Mbits de données échangées et plus de 4 Mbits de données mémorisées. L'exploration a été faite pour une architecture à 7 processeurs (7 PACM).

5. CONCLUSION

Dans cette partie, nous avons étudié trois applications pour valider l'approche d'analyse (les métriques), l'outil d'exploration GAMA² et l'approche d'exploration que nous avons proposés.

L'application émetteur d'un terminal UMTS a permis de valider la capacité des métriques à quantifier les caractéristiques analysées. Les différents résultats relevés sur cette application ont bien montré que les métriques sont capables de quantifier les caractéristiques analysées. Elles sont aussi aptes à anticiper les performances de l'architecture afin de guider l'exploration de l'architecture. Ainsi les métriques sont capables à haut niveau de juger de la qualité, de la communication, des performances d'exécution des tâches et des mémoires qui influent sur le temps, la surface et la consommation de l'architecture.

Les expérimentations effectuées sur l'application du codage des signaux AC3 ont permis de valider l'approche que nous proposons. En effet, les résultats ont montré que l'étape de pré-exploration, bien qu'elle réduise l'espace d'architectures, ne réduit pas la qualité des architectures trouvées. Cette étape a même permis de guider l'outil CODEF vers des architectures qui sont plus optimisées dans le temps.

Les résultats relevés sur l'applications émetteur d'un terminal UMTS, l'application codage des signaux AC3 et l'application suivi d'objet dans une image ICAM ont montré que l'outil GAMA² est capable d'explorer et de trouver un groupe de partitions qui répondent aux attentes du concepteur. Les spécifications ICAM4f et ICAM8f ont montré que l'outil est capable de faire face à un nombre élevé de tâches, d'arcs et de PACM. L'outil a été capable d'explorer une application de 55 tâches et 94 arcs et pour une architecture de 7 PACM.

VI. CONCLUSIONS ET PERSPECTIVES

1. CONCLUSION 138
1.1. REPONSE A LA PROBLEMATIQUE ET TRAVAIL REALISE 138
1.2. RESULTATS 140
1.3. ANALYSES CRITIQUES 140
2. PERSPECTIVES 140
2.1. PERSPECTIVES A COURT TERME 141
2.2. PERSPECTIVES A MOYEN TERME 141
2.3. PERSPECTIVES A LONG TERME 142

1. CONCLUSION

La complexité sans cesse grandissante des systèmes embarqués et des applications a conduit à l'évolution des architectures pour de tels systèmes. Les applications comportent de plus en plus de tâches, d'échanges de données et de parallélisme. Les architectures intègrent de plus en plus de ressources logicielles diversifiées, de ressources matérielles et de ressources de communication. Face à une telle complexité, l'expérience du concepteur et les outils dont il dispose atteignent aujourd'hui leurs limites afin d'appréhender efficacement l'étape d'exploration architecturale.

Nous avons étudié dans ce document les défis imposés par ces architectures. Nous avons enrichi les approches d'exploration existantes par une phase de pré-exploration. Cette phase a abouti à un outil d'analyse et d'exploration pour les architectures multiprocesseurs : GAMA² (Genetic Algorithm for Multi Processors Architectures Analysis).

1.1. REPONSE A LA PROBLEMATIQUE ET TRAVAIL REALISE

Nous avons étudié dans ce travail le problème de l'exploration des architectures multiprocesseurs pour des applications comportant un nombre élevé de tâches et un parallélisme important. La réflexion menée dans ce travail a permis de discerner les difficultés suivantes :

- ✓ Le coût de l'exploration des applications est de plus en plus élevé vu que les approches actuelles nécessitent des informations dépendantes de la technologie (estimations de performances sur les cibles visées).
- ✓ L'espace d'architectures est de plus en plus vaste rendant l'exploration plus difficile.
- ✓ La dimension spatiale influe de plus en plus sur les performances de l'architecture.
- ✓ Le parallélisme important rend plus complexe la conception des architectures mémoires et des communications.
- ✓ L'apparition et l'utilisation fréquente de ressources logicielles spécifiques ajoutent une nouvelle dimension à l'espace d'architectures (diversité logicielle) compliquant la phase d'exploration.

Pour faire face à ces difficultés, le concepteur doit se munir de méthodes d'exploration architecturale progressives qui sont partiellement indépendantes de la technologie. Dans ce travail nous considérons un modèle d'architecture multi-PACM qui permet de représenter la

majorité des architectures tout en appréhendant un nombre important d'informations. En effet, ce modèle d'architecture prend en compte la dimension spatiale, la diversité logicielle et le parallélisme.

Dans le cadre de l'exploration, nous avons proposé une nouvelle approche qui consiste à enrichir les approches existantes par une phase de pré-exploration. Cette dernière consiste à distribuer les tâches en groupes. Une distribution des tâches en groupes est appelée *partition.* Chaque groupe est exécuté par un PACM de l'architecture. Cette phase est peu dépendante de la technologie et donc peu coûteuse. Elle consiste à optimiser les communications, les mémorisations et les performances d'exécution des tâches.

Afin d'effectuer ces optimisations nous avons défini et élaboré un modèle d'analyse constitué de six métriques. Chaque métrique est un réel entre 0 et 1 permettant de juger la qualité de la caractéristique qui lui est propre :

- ✓ **IA** : permet de juger la qualité d'exécution des tâches pour une partition.
- ✓ **MEM** : permet de juger la qualité de distribution des données mémorisées dans les PACM pour une partition.
- ✓ **EDE** : permet de juger la qualité de distribution des données échangées dans les PACM pour une partition.
- ✓ **DEIC** : permet de juger la réduction de la quantité des données échangées entre les PACM pour une partition.
- ✓ **CIC** : permet de juger la réduction du nombre des transferts de données entre les PACM pour une partition.
- ✓ **TC** : permet de juger la qualité de distribution des *débits contraintes* dans les PACM pour une partition.

Nous avons mis en œuvre un algorithme génétique multi-objectif pour explorer l'espace des partitions et trouver la partition qui optimise ces métriques. Dans ce cadre nous avons conçu l'outil GAMA². Il se compose de deux modules :

- ✓ Un module d'analyse : il permet de calculer les métriques pour une partition particulière et donne des informations sur la distribution des données dans et entre les PACM, la distribution des mémoires et la distribution des débits contraintes.
- ✓ Un module d'exploration : il permet de parcourir l'espace des partitions et trouver les partitions optimales au sens du pareto.

L'outil permet aux concepteurs d'intervenir pour définir et guider l'exploration.

1.2. RESULTATS

L'approche proposée a été validée sur plusieurs applications. La première étude s'est basée sur un émetteur UMTS. Cette application a permis de valider les métriques et leur analyse par l'algorithme génétique. Le résultat de l'exploration automatique a ainsi permis de mettre en évidence la même architecture que celle résultant d'une optimisation manuelle. La seconde étude a porté sur une application de codage audio AC3. Cette application a permis de valider le flot proposé en deux étapes. L'outil d'exploration logiciel/matériel CODEF a été utilisé pour la deuxième étape du flot d'exploration. La combinaison des deux outils a permis de réduire l'espace de conception et d'identifier certaines architectures optimisées. Enfin, la troisième étude a testé l'efficacité de l'approche proposée pour des applications contenant un nombre de tâches élevé. Plusieurs scénarios ont été considérés depuis une spécification d'une dizaine de tâches jusqu'à une cinquantaine de tâches et pour une architecture contenant une dizaine de processeurs. Cette troisième application (suivi d'objets dans une image, ICAM) a démontré la capacité de l'approche proposée à appréhender un espace de conception étendu aussi bien du point de vu de l'application que de l'architecture.

1.3. ANALYSES CRITIQUES

Ce travail présente une première réflexion sur les problèmes d'exploration d'architectures multiprocesseurs pour les applications actuelles et futures. Le choix des métriques a été élaboré par une étude théorique. Une expertise plus détaillée sur des exemples réels permettra de mieux étudier les différentes métriques et peut être d'en proposer d'autres. Le choix du nombre de PACM est pour le moment effectué de façon manuelle. Une étude doit être menée pour automatiser cette étape. La phase de choix du concepteur d'une solution parmi celles générées par GAMA² est pour le moment également manuelle. Des utilitaires pour l'aider à analyser les différentes solutions peuvent être utiles surtout si le nombre de solutions générées est important.

2. PERSPECTIVES

Un certain nombre d'améliorations est possible pour les travaux présentés précédemment. Certaines sont uniquement de l'ordre du développement informatique de l'outil pour faciliter son utilisation. Elles sont prévues à court terme. D'autres consistent à

automatiser des phases de l'approche et à améliorer la qualité de la pré-exploration, et sont prévues à moyen terme. Enfin, d'autres touchent l'approche globale d'exploration des architectures et sont prévues à long terme.

2.1. PERSPECTIVES A COURT TERME

L'outil GAMA² est actuellement sous forme de code C qui permet de lire les spécifications sous format texte et de générer les résultats de l'exploration et de l'analyse sous format texte également. A court terme il faut l'enrichir par une interface graphique pour assouplir son utilisation. Des graphes de sorties, générés manuellement pour le moment, faciliteront la compréhension et l'utilisation des résultats.

Le choix de la solution parmi l'ensemble généré par GAMA² se fait de façon totalement manuelle. Des utilitaires peuvent accélérer le processus du choix de la partition. En effet, le nombre de solutions optimales au sens du Pareto peut atteindre plusieurs dizaines. Le choix de la partition peut donc être potentiellement coûteuse en temps. Nous proposons quelques utilitaires à ajouter à GAMA² pour faciliter la phase de choix de la partition :

- ✓ Un utilitaire qui fournit pour chaque métrique sa moyenne par rapport aux solutions générées ainsi que sa plus petite et sa plus grande valeur.
- ✓ Un utilitaire pour calculer la surface couverte par l'espace des métriques pour chaque solution
- ✓ Un utilitaire qui permet de supprimer des solutions qui ont une surface ou une métrique inférieures à certaines valeurs fixées par le concepteur.

2.2. PERSPECTIVES A MOYEN TERME

La définition des métriques est basée sur une étude théorique. Bien qu'elles se soient montrées efficaces et suffisantes avec les applications que nous avons utilisées, plus d'expertises sur des exemples réels plus complexes sont nécessaires pour valider ces métriques ou d'en ajouter d'autres.

Le choix du nombre de PACM dans l'architecture se fait pour le moment de manière manuelle. Actuellement, il n'existe pas de travaux qui permettent de fixer le nombre de processeurs d'une architecture à un aussi haut niveau. Une analyse approfondie du parallélisme et de l'ordonnançabilité des tâches de l'application est nécessaire pour automatiser ce choix.

L'algorithme génétique développé dans l'outil GAMA² a permis d'explorer différents espaces de solutions. Toutefois des techniques d'intensification sont possibles pour lui permettre de converger plus rapidement et plus efficacement. Plusieurs méthodes sont possibles :

- ✓ Nous stockons les meilleurs individus de la population d'une génération. Après la sélection, le croisement et la mutation, la population de la génération suivante est créée. Les individus stockés sont ajoutés à cette nouvelle population. Puis, une nouvelle sélection est effectuée pour stocker de nouveau les meilleurs individus.
- ✓ Lors de chaque génération, des individus sont choisis aléatoirement. Une exploration mono-objectif est effectuée pour améliorer l'une des métriques. Cette méthode permet d'accélérer l'optimisation des métriques.

2.3. PERSPECTIVES A LONG TERME

L'approche que nous proposons consiste à précéder les approches d'exploration traditionnelles par l'outil GAMA². Certes ces outils permettent de trouver des architectures qui satisfont les contraintes, toutefois elles ne sont pas totalement adaptées à l'exploration d'architectures multi-PACM. En effet, la plupart des outils ne permettent pas de trouver une architecture complètement fidèle au modèle multi-PACM :

- ✓ Un bus par PACM.
- ✓ Un seul processeur par PACM.
- ✓ Une architecture mémoire par PACM.

Il serait intéressant d'élaborer un outil d'exploration d'architectures offrant une plus grande compatibilité avec l'approche proposée. Ceci permettra aux architectures de mieux profiter des optimisations faites par l'outil GAMA².

Enfin, pour clore ce mémoire, nous sommes convaincus de l'intérêt de nos travaux pour l'exploration des architectures multiprocesseurs pour les applications complexes du futur. Nous n'avons pas pu résoudre tous les problèmes comme celui du choix du nombre de PACM, mais la réflexion établie et les modèles et outils d'analyses mis en œuvre constituent un apport important pour faire face à la complexité et à l'augmentation du coût de l'exploration des architectures.

Bibliographie

[Airiau 2000] : Airiau R., Carer A., Casseau E., Martin E., O.Sentieys, "Méthodologies de conception des composants virtuels pour les applications de TDSI". 5ème workshop AAA sur l'adéquation algorithme architecture (Rocquencourt, 26-28 janvier 2000).

[Analog] : Analog Device. [En ligne] : http://www.analog.com

[Arm] : ARM. [En ligne] : http://www.arm.com

[Auguin 2001] : Auguin M, Capelle L, Cuesta F, Gresset E,"CODEF : A system level design exploration tool", IEEE International conference on Acoustics, Speech, and Signal 2001(ICASSP '01)

[Baghdadi 2000] : Baghdadi A, Zergainoh N-E, Lyonard D, Jerraya A A, "Generic architecture platform for multiprocessor system-on-chip design", International Workshop on Distributed and Parallel Embedded Systems: Architecture and Design of Distributed Embedded Systems 2000.

[Baghdadi 2002] : Baghdadi A, Zergainoh N-E, Césàrio W, Roudier T, Jerraya A A, "Exploration de l'espace des solutions architecturales dans le codesign – Estimation des performances au niveau système", Technique et Science Informatiques, janvier 2002

[Balarin 1997] : Balarin F, Sentovich F, Chiodo M, Giusto P, Hsieh H, Tabbara B, Ju-recska A, Lavagno L, Passerone C, Suzuki K, Sangiovanni-Vincentelli A. "Hardware-Software Co-design of Embedded Systems : The POLIS approach, " Kluwer Academic Publishers, 1997.

[Balarin 1999] : Balarin F and All, "POLIS User's Manual," 1999.

[Barros 1994] : Barros E, Rosenstiel W, Xiong X, "A Method for Partitioning UNITY Language in Hardware and Software, " in : Proceedings EURO-DAC'94 with EURO VHDL'94, Grenoble, France (1994) 220-225.

[Bazes 1996] : Bazes M ; Ashuri R, Knoll E, "An interpolating clock synthesiser", IEEE journal of Solid-State Circuits, vol 31, pp 1295-1300 1996.

[Berekovic 2002] : Berekovic M, Pirsch P, Selinger T, Miro C, Lafage A, Wels K, Heer C, Ghigo G, "Architecture of an Image Rendering Co-Processor for MPEG-4 Visual Compositing, Kluwer Journal of VLSI Signal Processing Systems 2002; 31(2): 157-171.

[Berry 1992] : Berry G. and Gonthier G.. "the Esterel Synchronous Programming Language : Design, Semantics, Implementation, " Ecole Nationale Sup¶erieure des Mines de Paris et Institut Nationale de recherche en Informatique et Automatique, 1992.

[Bianco 1999a] : Luc Bianco, "Méthodologie de partitionnement pour la conception de systèmes embarqués de traitement de signal temps réel". Thèse préparée à l'université de Nice – Sophia Anitipolis, Octobre 1999.

[Bianco 199b] : L Bianco,Augin M, Pegatoquet A, Gresset E,"CODEF : un environnement d'exploration d'architectures de systèmes embarqués pour le traitement du signal temps réel", 7ème colloque de GRETSI sur le Traitement du Signal et des Images. Vannes, septembre 1999.

[Borkar 2007] : Borkar Shekhar, Jouppi Norman P., Stenstrom Per, "Microprocessors in the era of terascale integration", Design Automation and Test in Europe DATE, 2007.

[Bouchhima 2006] : Bouchhima aymen, "Modélisation du logiciel embarqué à différents niveaux en vue de la validation et la synthèse des systèmes monopuces". Thèse préparée à l'école doctorale de Grenoble d'Electronique, Electronique, Automatique et Traitement de Signal, mai 2006.

[Brandolese 2006] : Carlo Brandolese, William Fornaciari, Luigi Pomante, Fabio Salice, Donatella Sciuto, "Affinity-driven system design exploration for heterogeneous multiprocessor SOS".

[Buck 1994] : Buck J, Ha S, Lee A, Masserchmitt D D,"Ptolemy : A framework for simulating and prototyping heterogeneous system", Int Journal Computing simulation, Vol 4, pp 155-182, 1994

[Caldari 2003] : Caldari M, M. Conti, M. Coppola, P. Crippa, S. Orcioni, L. Pieralisi, C. Turchetti "System-level power analysis methodology applied to the AMBA AHB bus" Design Automation and Test in Europe DATE, 2003

[Calvez 1996] : Calvez J.P,"A System-level performance model and method", "Current Issues In Electronic Modeling", Issue #6: Metamodeling: Performance, Software and Information Modeling, Kluwer Academic Publishers, 1996, pp 57-102

[Chandrakasan 1995] : Chandrakasan A, Broderson R,"Low power digital CMOS digital design" Kluwer Academic Publishers 1995.

[Cradle 2007] : CT3400 multiprocessor DSP Data Sheet [En ligne] http://cradle.com

[Curd 2007] : Curd Derek,"Power consumption in 65 nm FPGAs", xilinx white paper : virtex-5 FPGAs, wp246(v1.2), février 2007

[Dave 1998a] : Dave B.P, Jha N, "CASPER : Concurrent HardwareSoftware Cosynthesis of Hard Real-Time Aperiodic and Periodic Speci¯cations of System Architectures, " Proc. DATE, IEEE CS Press, pages 118-124, 1998.

[Dave 1998b] : Dave, B.P, Jha, N.K.," COHRA: hardware-software cosynthesis of hierarchical heterogeneous distributed embedded systems", Computer-Aided Design of Integrated Circuits and Systems, Volume 17, Issue 10, Oct. 1998 Page(s):900 - 919

[Dave 1999] : Dave Bharat P, Ganesh Lakshminarayana, Niraj K. Jha, "COSYN: Hardware–Software Co-Synthesis of Heterogeneous Distributed Embedded Systems". IEEE Transaction (VLSI) Systems, VOL. 7, NO. 1, MARCH 1999

[Dick 1998] : Dick R P., Jha Niraj K, "MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-Synthesis of Distributed Embedded Systems", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (1998)

[Dick 1999] : Dick R.P, Jha N.K,"MOCSYN: Multiobjective core-based single-chip system synthesis", Design Automation and Test in Europe Date 1999.

[Diguet 2005] : Diguet Jean-Philippe, "Exploration de l'espace de la conception de SoC, de l'asservissement à la coopération", Habilitation à Diriger des Recherches, Université de Bretagne du sud, septembre 2005.

[Dréo 2003] : Dréo Johann, Pétrowski Alain, Siarry patrick, Taillard Eric,"Metaheuristiques pour l'optimisation difficile" Editions Eyrolles 2003, ISBN : 2-212-11368-4

[Ekerling 1996] : Eikerling H, Hardt W, Gerlach J, Rosentiel W, "A methodology for rapid analysis and optimisation of embedded systems"Engineering of computer based systems, germany 1996.

[Elleouet 2006] : Elleouet D, Savary Y, Julien N, Houzet D "A FPGA Power Aware Design Flow" Power And Timing Modeling, Optmisation and Simulation, international workshop, Montpelier, France, septembre 2006.

[Ernst 1996] : R. Ernst and J. Henkel. Th. Benner, W. Ye, U. Holtmann, D. Hermann, M. Trawny, "The COSYMA environment for hardware/software co-synthesis of small embedded systems", Microprocessors ans microsystems, 20, pp. 159-166, 1996.

[Fei 2002] : Fei ynsi, Jha Niraj. K,"Functionnal partitioning for low power distributed systems of Systems-on-chip", International Conference on VLSI Design, VLSID 2002.

[Flynn 1966] : FLYNN M.J.,"Very High-Speed Computing Systems". *Proc. IEEE, déc. 1966.*

[Gaisler] : LEON3 SPARC V8 Processor core. [En ligne] : http://www.gaisler.com/cms

[Gajski 1998] : Gajski D, Vahid F, Narayan S, Gong J,"System Level specification with SpecSyn" Design Automation Conference DAC 1998.

[Garcia 2005] : Garcia A, Gonzales L, Felix R, «Power consumption management on FPGAs» 15th International Conference on Electronics, Communication and Computers, 2005.

[Gpp] : The 3rd Generation Partnership Project. [En ligne] : www.3gpp.org

[Grand dictionnaire] : Le grand dictionnaire terminologique. [En ligne] : http://www.granddictionnaire.com

[Guitton 2003] : Guitton-Ouhamou P, Belleudu C, Auguin M, "Energy Optimization in Hw/Sw Tool : Design of Low Power Architecture System". IEEE International Workshop on System on Ship for Real-Time Systems - IWSOC'2003, Calgary, Canada, Juin 2003.

[Gupta 1996] : Gupta R, DeMicheli G, "A Cosynthesis approach to Embedded System Design Automation", International Journal of Design Automation for Embedded System, Vol. 1, Nos. 1-2, 1996

[Gwendal 1997] : Gwendal Le Fol, "Architecture parallèle pour la compression vidéo : Contribution à la conception d'un module VLSI programmable et à l'étude d'outils de compilation reciblables", thèse préparée à l'université de rennes Institut de Formation Supérieure en Informatique et Communication soutenue le 26 septembre 1997.

[IBM] : Power architecture. [on line] http://www-106.ibm.com/developerworks/eserver/library/es-archguide-v2.html.

[Ipode 2007] : Fiche technique du Ipode apple 80GO. [En ligne] : http://www.apple.com/fr/ipod/ipod.html

[Iphone 2007] : Fiche technique des Iphones de apple. [En ligne] : http://www.apple.com/iphone

[Jantsch 1994] : Jantsch A, Ellervee P,Oberg J, Hemani A, Tenhunen H, "Hardware/Software Partitioning and Minimizing Memory Interface Trafic," European Design Au-tomation Conference, 1994.

[Jerraya 1999] : Jerraya et al."Multilanguage specification for system design and codesign", chapitre du livre "System level synthesis", NATO ASI 1998 édité par A.Jerraya et J.Mermet, Kluwer Academic Publisher, 1999.

[Khale 2005] : Khale J A, Day M N, Hofstee H P, Jhons C R, Maeurer T R, Shippy D, "Introduction to CELL multiprocessor" IBM Journal of Research and Development, Jul-Sep 2005

[kappen 2006] : Kappen G.; Noll T.G, "Application specific instruction processor based implementation of a GNSS receiver on a FPGA", Design Automation and Test in Europe DATE 2006. Proceedings Volume 2, 6-10 March 2006 Page(s):6 pp.

[Ktari 05] : Ktari, J, Laurent J, Abid M, Julien N, «Estimation de la consommation logicielle dans un système embarqué : Etude de cas», Conférence de faible tension faible consommation, FTFC 2005.

[Kees 2002] : Kees Vissers, "The Trimedia CPU64 VLIW Media Processor", Invited session at the International Conference on Computer Design, ICCD 1999.

[Kunimatsu 2000] : A. Kunimatsu, N. Ide, T. Sato, Y. Endo, H. Murakami, T.Kamei, M. Hirano, F. Ishihara, H. Tago, M. Oka, A. Ohba,T. Yutaka, T. Okada, and M. Suzuoki, "Vector Unit Architecture for Emotion Synthesis", IEEE Micro 20, No. 2, 40–47 (March–April 2000).

[Lahiri 2001] : Lahiri Kanishka, Raghunathan anand, Dey sujit, "System-level performance analysis for designing on-chipcommunication architectures" IEEE Trans. on CAD of Integrated Circuits and Systems 20(6): 768-783 (2001)

[Lahiri 2004] : Lahiri, K. Raghunathan, A. Power analysis of system-level on-chip communication architectures, International Conference on Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004.

[Landman 1995] : Landman Paul E, Rabaey Jan M, "Activity-sensitive architectural power analysis for the control path", International Symposium on Low Power Electronics and Design, Etats unis 1995.

[Laurent 2004] : Laurent J, Julien N, Eric Senn, Eric Martin "Functional Level Power Analysis: An Efficient Approach for Modeling the Power Consumption of Complex Processors", Design, Automation and Test in Europe Conference DATE 2004

[Laurent 2007] : Laurent Johan, Coussy Philippe,"Impact du type d'architecture sur la consommation d'une application", Conférence de faible tension faible consommation FTFC, paris 2007.

[Le Moullec 2003] : Le Moullec Yannick, Nader Ben Amor, Jean-Philippe Diguet, Mohamed Abid et Jean-Luc Philippe "Multi-Granularity Metrics for the Era of Strongly Personalized SOCs" Design, Automation and Test in Europe Conference (DATE 03), Munich, Allemagne, 3-7 Mars 2003.

[Marcuello 1998] : Marcuello P, Gonzales A, Tubella J,"Speculative multi thread processor", ACM international conference on supercomputing, 1998.

[Marteil 2004] : F. Marteil, N. Julien, E. Senn, and E. Martin. A Complete Methodology for Memory Optimization in DSP Applications. In EUROMICRO Symposium On Digital System Design, Architectures, Methods and Tools, pages 98–103, August 2004.

[Momcilovic 2006] : Momcilovic S, Dias T, Roma N, Sousa L, "Application Specific Instruction Set Processor for Adaptive Video Motion Estimation", digital system design : Architectures, Methods and Tools, Euromicro conference 2006.

[Motorola] : Téléphone mobile motorola. [en ligne] http://www.motorola.com.

[Moy 2001] : MOY Christophe, Apostolos KOUNTOURIS, Luc RAMBAUD,Pascal LECORRE , "Full Digital IF UMTS Transceiver for Future Software Radio Systems" ERSA'2001, Las Vegas, USA, 25-28 June 2001 (first international conference on Engineering of Reconfigurable Systems and Algorithms).

[N95 2006] : Nokia Nseries, Press release, septembre 2006 [en ligne] http://www.nokia.fr

[Osterling 1997] : Osterling A, Benner A, Ernst R, Herrmann D, Scholz T, Ye W. "Hardware/Software Co-Design : Principles and Practice, chapter The COSYMA System," Kluwer Academic Publishers, 1997.

[Pakdeepaiboonpol 2006] : Pakdeepaiboonpol, P. Kittitornkun, S. «Low energy optimization for MPEG-4 video encoder on ARM-based mobile phones» 1st International Symposium on Wireless Pervasive Computing, 2006

[Pareto1896] : V. Pareto. Cours d'Economie Politique. F. Rouge, Lausanne, 1896.

[Qtek 2007] : Fiche techniques des PDA Qtek [En ligne] http://www.dell.fr

[Renaud 2006] : Jean Renaud, "De l'optimisation multicritère à l'aide à la décision : propositions de modèles de choix des recommandations à l'industriel lors de la conception et de la fabrication de produits nouveaux", conférence invitée au 14e Atelier de Raisonnement à Partir de cas – 30-31 mars 2006 – Besançon (France)

[Rouxel 2006] : Rouxel Samuel, "Modélisation et Caractérisation de Plates-Formes SoC Hétérogènes : Application à la Radio Logicielle", thèse soutenu le 5 Décembre 2006 à l'université de Bretagne du sud, Lorient.

[Sciuto 2002] : D.Sciuto, F.Salice, L.Pomante and W.Fornaciari, "Metrics for Design Space Exploration of Heterogeneous Multiprocessor Embedded Systems", CODES'02, Estes Park, USA, May 2002

[ST] Sensor Technology. [En ligne] : http://www.st. com

[Suzuki 1996] : Susuki K, Sangiovanni-Vincentelli A, "Efficient software performance estimation methods for hardware/software codesign", Design Automation Conference, Dac 1996.

[Texas] Texas Instrument. [En ligne] : http://www.ti.com

[Wenjie 2002] : Wenjie Jiang, Tiwari, V., de la Iglesia, E., Sinha, A., "Topological analysis for leakage prediction of digital circuits", Design Automation Conference, 2002. Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design. Proceedings.

[Yamauchi 1992] : Yamauchi (H.), Tashiro (Y), Minami (T.), Suzuki (Y) "Architecture and Implementation of a Highly Parallel Single-Chip Video DSP". IEEE Transactions on Circuits and Systems for Video Technology, vol. 2, n° 2,jun 1992, pp . 207-220 .

[Ye 1995] : Ye W, Ernst R,"Worst case timing estimation based on symbolic execution", COBRA report'95, Institute of computer ingineering, Technical University of braunshweig, Allemagne, octobre 1995.