Partie II. Le calcul scientifique

Chapitre XII. Adéquation Algorithme Architecture en Traitement du Signal et des Images

Yves Sorel INRIA-Rocquencourt

Table des matières

1 Introduction

La chaîne de traitement de l'information portée par le signal ou l'image est construite de manière à faire le lien entre le monde de l'observation, régi par la mesure de phénomènes physiques, et le monde de l'action nécessitant la compréhension de ces phénomènes. L'acquisition du phénomène physique passe par le capteur (antenne, microphone, caméra, scanner, etc.). Le signal recueilli correspond à une forme plus ou moins distordue du phénomène observé. Des traitements faisant appel à l'électronique, à l'optique et à l'informatique sont nécessaires pour le reconstruire et le coder, le transmettre et l'archiver, l'analyser et le restituer, le comprendre et l'exploiter.

Il est bien sûr fondamental de développer en amont de cette chaîne, un processus de modélisation mathématique du phénomène observé. On ne s'intéressera ici qu'à la chaîne de traitement elle-même, à laquelle on ajoute en aval un processus de compréhension en accord avec l'intelligence humaine. Sans vouloir copier l'intelligence humaine, les systèmes intelligents de traitement du signal et des images incluent des fonctions de perception, d'apprentissage, d'analyse et d'interprétation, leur permettant de réagir face à des situations nouvelles ou imprévues.

On considéra donc cette chaîne étendue comme une application de traitement du signal et des images, devant respecter des contraintes temps réel et d'embarquabilité, pour lesquelles le non respect de ces contraintes pourrait conduire à une perte de contrôle sur l'environnement physique, ayant des conséquences plus ou moins catastrophiques, allant jusqu'à détruire le système de contrôle lui-même. La phase de modélisation et de simulation associée servant à valider les modèles et à déterminer les paramètres utilisés en temps réel (par exemple un coefficient de convergence utilisé dans un filtre numérique adaptatif), sera assimilée au calcul scientifique. Ce domaine qui nécessite parfois une puissance de calcul importante, apportée par des calculateurs massivement parallèles, pour "accélérer" l'obtention des résultats, est traité ailleurs dans cet ouvrage.

2 Applications de traitement du signal et des images

2.1 Enjeux et tendances en traitement du signal et des images

Pour mieux saisir les caractéristiques des applications de traitement du signal et des images et ce qu'elles impliquent, il est intéressant de préciser les enjeux aussi bien que les tendances du domaine considéré.

Les enjeux techniques et technologiques sont la conséquence de la grande diversité (signal, son, parole, image) et du flot gigantesque des données (TVHD, CD-Rom) et des moyens accrus de communication (fibres optiques, téléphones portables, réseaux de communication, internet) manipulés, et qu'il faut comprendre, exploiter, stocker, rechercher, transmettre. Pour éviter un encombrement de l'espace hertzien, pour contrôler la saturation des mémoires de masse, pour augmenter les vitesses de stockage et d'accès, pour pouvoir prendre plus vite des décisions cohérentes, il est nécessaire de savoir traiter ces données ensembles, de savoir les traiter très rapidement et de savoir en extraire l'information pertinente. Voici quelques enjeux technologiques importants :

- maîtrise des grands ensembles de données et flux d'informations : compression, codage, transmission, interprétation, archivage, indexation et scrutation, recherche. Capteurs multiples, fusion de données brutes et symboliques, qualitatives et quantitatives, classification, reconnaissance de formes (parole, images) ;
- conception de systèmes intelligents : perception, focalisation, apprentissage, aide à la décision, autonomie décisionnelle ;

- sûreté de fonctionnement et diagnostic : détection et diagnostic de défauts et d'alarmes, traitement des incidents, coopération homme-machine.

Sans être exhaustif, voici aussi les tendances qui se dégagent actuellement.

En traitement du signal les modèles mathématiques et les algorithmes qui en sont dérivés s'orientent vers la décision et l'interprétation en non stationnaire. Il s'agit essentiellement d'utiliser les techniques temps-féquence et temps-échelle (ondelettes, Wigner-Ville ...) à des fin de détection, classification ou reconnaissance. Elles sont utilisées dans un grand nombre d'applications différentes : sonar, biomédical, contrôle non destructif, vibrations, surveillance etc. Les signaux traités issus de systèmes non linéaires, chaotiques ou à structure fractale ont des dynamiques de plus en plus complexes et sont généralement à longue dépendance. On retrouve ce type de signaux dans les réseaux de communication, les phénomènes biologiques, les phénomènes turbulents etc. Les approches adaptatives très importantes en traitement du signal conduisent à des algorithmes stochastiques, à pas variables etc, pour lesquels ont a recherché à la fois la performance et la réduction de complexité, en réalisant des traitements par blocs par exemple. Le traitement d'antenne tient une place particulière car on le retrouve dans un très grand nombre d'applications où il joue un rôle déterminant. Les aspects multi-capteurs sont maintenant omniprésents, ils concernent les télécommunications et les systèmes de détection. Parmi les applications que ces techniques on peut citer l'égalisation multi-capteurs, les systèmes de communication personnels, les systèmes de communication par satellites, la synthèse de réseaux de télécommunications, le radar à détection polarimétrique, la goniométrie HF etc. Pour de nombreuses applications l'observation du signal lui-même n'est pas suffisante, on cherche dans l'observation le signal d'excitation qui l'a créé car il possède parfois un contenu informationnel très riche. Pour cela des systèmes autodidactes sans séquence d'apprentissage sont nécessaires dans des applications telles que l'égalisation, le traitement sismique, le contrôle non destructif etc. La séparation de sources et la séparation des mélanges convolutifs sont utilisées là encore dans les applications multi-capteurs.

En traitement d'images, les traitements bas niveau (niveau pixel) et niveau intermédiaire sont encore prépondérants, ils concernent principalement les aspects filtrage et analyse d'image en vue de fournir une représentation segmentée de l'image. On traite de plus en plus des images bruitées et texturées rendant la segmentation difficile. Cela conduit à des approches non linéaires, à du filtrage morphologique et polynômial, à des techniques d'adaptation aux perturbations pour la détection et la classification. Les approches coopératives mettant en œuvre plusieurs techniques différentes, utilisées auparavant séparément, sont de plus en plus exploitées pour obtenir de bonnes segmentations. On pourra par exemple combiner deux des approches suivantes : décomposition (ondelettes ...), modèles variationnel (diffusion), modèles markoviens, modèles déformables. Les traitements de haut niveau placés en aval des précédents traitements consistent à réaliser, à partir des images segmentées, de l'interprétation de scènes, de la caractérisation, ou de la transmission d'images compressées dans le cas du codage. À ce niveau on utilise souvent des techniques proches de celles utilisées en intelligence artificielle. On met en œuvre des techniques de perception par transformations orthogonales, de multi-résolution ou de focalisation. On fait aussi appel à des techniques de reconnaissance et d'interprétation d'objet localisé dans l'image segmentée. Enfin, les images que l'on traite aujourd'hui sont dynamiques et déformables, c'est le cas des applications d'analyse de mouvement d'objets multidimentionnel que l'on retrouve en imagerie médicale, en vision par ordinateur, en télédétection etc.

Transversalement à ce qui précède on peut aussi citer les techniques de fusion d'informations qui sont utilisées aussi bien en signal qu'en image. Ce domaine très riche prend de plus en plus

d'importance, il concerne aussi bien les données numériques au bas niveau que les données symboliques à haut niveau. De la même manière que l'on a besoin de données issues de capteurs multiples qui peuvent être de type différents, on fusionne aussi les résultats de traitements différents pour en tirer des informations symboliques, nécessaires à l'interprétation de phénomènes complexes quand il s'agit d'effectuer des prises de décisions. Là encore on est proche des techniques d'intelligence artificielle.

Il faut aussi noter que les deux communautés signal et images sont de nos jour très proches. Il est maintenant largement admis que les modèles utilisés en traitement du signal et en traitement d'images vont vers une unification, et force est de constater que de plus en plus d'applications complexes mélangent traitement du signal et traitement d'images faisant appel aux mêmes techniques. Les applications de codage faisant appel à la compression de signal audio ou de signal vidéo sont de bons exemples de cette tendance. L'époque où le traitement du signal et des images se cantonnait principalement au domaine du filtrage est révolu depuis longtemps. Comme on a essayé de le montrer ci-dessus en termes d'enjeux et de tendances, les applications de traitement du signal et des images sont de plus en plus complexes. Bien que les traitements de bas et moyen niveau, mettant en œuvre principalement des gros volumes de calculs, assez souvent réguliers avec peu de contrôle restent prépondérants, les traitements faisant intervenir des techniques de l'intelligence artificielle se font de plus en plus nombreux. Dans ce cas la régularité est en général perdue, principalement à cause des aspects contrôle, qui deviennent plus nombreux ce qui rend la mise en œuvre encore plus difficile.

L'ensemble de ces caractéristiques particulières conduisent d'une part à des structures d'algorithmes hétérogènes très différentes de celles utilisées en calcul scientifiques et d'autre par conduisent à utiliser des architectures matérielles spécialisées utilisant des circuits intégrés spécialisés, des processeurs de traitement du signal et des processeurs généralistes, pour implanter au mieux ces algorithmes. Il s'agit aussi de respecter des contraintes temps réel et d'embarquabilité, de prendre en compte la nature distribuée des données (multi-capteur, image répartie ...), de minimiser le nombre de composants matériel et enfin de permettre l'évolution rapide des produits (portage du logiciel, évolution du matériel ...)

2.2 Systèmes réactifs temps réel embarqués

Dans les applications visées on met en œuvre des systèmes informatiques qui interagissent avec l'environnement physique. [1] Ce dernier produit des événement ou stimuli reçus via des capteurs par le système informatique, qui à son tour, produit en réaction pour l'environnement des événements via des actionneurs. Les événements générés à la suite des opérations de calcul et des transferts de données réalisés par le calculateur sont utilisés pour contrôler l'environnement. Par exemple, les images acquises par un capteur CCD d'un système d'aide à la conduite dans les automobiles du futur, sont traitées par un des calculateurs à bord du véhicule en vue de repérer le véhicule qui le précède pour avertir le conducteur, par une alarme sonore et visuelle, quand la distance est inférieure à une distance minimale. Le contrôle peut être plus complexe si un système de ralentissement automatique agissant sur les freins, doit être actionné. On appelle ici système informatique, à la fois le logiciel (applicatif et système) et le calculateur sur lequel il s'exécuter. L'application correspond au couple système informatique et environnement auquel on peut ajouter un ensemble de contraintes, comme des contraintes de performances temps réel (latence, cadence) et des contraintes d'embarquabilité (consommation électrique, volume, poids etc ...). Pour le concepteur de l'application, il s'agit alors de garantir que le comportement de l'application corresponde aux spécifications

et que les contraintes soient respectées.

2.3 Architectures spécialisées

Bien que les processeurs d'usage général soient de plus en plus performants, leur fréquence d'horloge et le nombre des transistors qui les composent augmentant très rapidement, certaines applications de traitement du signal et des images nécessitent une puissance de calcul largement supérieure à celle actuellement disponible sur les processeurs les plus rapides utilisés au cœur des stations de travail. Pour satisfaire ce besoin très important de puissance de calcul, ainsi que pour prendre en compte la délocalisation de certaines fonctionnalités lorsque l'on cherche à rapprocher les organes de calcul le plus près possible des capteurs et des actionneurs pour limiter les problèmes liés au transport de données analogiques, des architectures parallèles, multiprocesseur, réparties ou distribuées sont nécessaires. Par exemple, dans le domaine du sonar ou du radar multi-voies il est parfois nécessaire d'utiliser une centaine de processeurs de traitement du signal, principalement pour résoudre les besoins de puissance de calcul. Dans le domaine des transports on aura rapidement à mettre en œuvre quelques dizaines de microcontrôleurs afin de rapprocher ces derniers des capteurs (température, pression, richesse etc ...) et des actionneurs (injection, suspension, freins etc ...) en vue de limiter le câblage, ou bien dans les transports aériens d'atténuer les effets des émissions électromagnétiques. Certaines applications dans le domaine de l'avionique requièrent à la fois puissance de calcul et distribution.

Plusieurs types de processeurs sont actuellement disponibles correspondant aux différents types d'applications envisagées. Dans les applications dont il est question ici, les processeurs RISC 32 ou 64 bits (à jeu d'instructions réduit) sont assez rarement utilisés. Ils ont été principalement conçus pour construire des stations de travail généralistes offrant un système d'exploitation moderne (UNIX, WindowsNT etc ...) très gourmand en puissance de calcul et en mémoire mais offrant un ensemble de fonctionnalités orientées utilisateur (gestion de fichiers et de périphériques, environnement fenêtré, compilation etc ...) qui ont conduit les concepteurs à proposer des jeux d'instructions parfaitement adaptés. Il faut aussi noter que ces fonctionnalités sont orientées plutôt vers le contrôle que vers le calcul. S'ils sont très efficaces pour ce type d'application, ils montrent leurs limites lorsqu'il s'agit d'effectuer des traitements numériques, comportant souvent peu de contrôle, aussi complexes que ceux nécessaires en traitement du signal et des images. Les processeurs de traitement du signal plutôt de type CISC 16 ou 32 bits (à jeu d'instruction complexe) permettent de réaliser de manière très efficace certaines opérations clés du traitement du signal et des images. La plus importante d'entre elles est la somme de produits, utilisée pour réaliser une convolution en une (traitement du signal) ou deux dimensions (traitement d'images) dans différents algorithmes, comme le filtrage à coefficients fixes ou adaptatifs. Cette opération qui met en œuvre en parallèle une multiplication et une accumulation ainsi que des calculs d'adresses et des accès mémoire multiples, s'effectue en un cycle du processeur. D'autres opérations spécifiques comme le brassage d'indices, très utilisé dans les transformées de Fourrier, de Walsh-Hadamar etc, sont aussi possibles. Enfin, il faut mentionner les microcontrôleurs souvent plus rustiques que les processeurs précédents au niveau des capacités de leur unité centrale avec leur chemins de données sur 4, 8, ou 16 bits et un jeu d'instruction peu sophistiqué. Cependant, ils offrent d'une part des capacités d'entrée sortie intéressantes, entrée-sorties analogiques, entrée-sorties tout ou rien, horloges temps réel etc, et d'autre part sont peux coûteux ce qui est compatible avec les grandes séries industrielles des produits grand-publics.

Il est bien sûr difficile de donner une règle systématique pour le choix du type de processeur

car chaque processeur permet de faire, à condition de payer un certain coût de développement et de volume de programme et donc de façon moins performante, ce qu'un autre peut faire.

De plus, les contraintes temps réel et d'embarquabilité peuvent être tellement forte que les processeurs disponibles sur le marché ne suffisent pas. Cela conduit à utiliser des circuits intégrés spécialisés (ASIC), reconfigurables (FPGA) ou "full custom" qui réalisent généralement des fonctionnalités dites de bas niveau, car elles sont identifiées depuis longtemps et ne sont pas remises en cause profondément lors de l'évolution du produit.

2.4 Compromis logiciel/matériel

Ici se pose la question du compromis logiciel/matériel ou "co-design" consistant à choisir les fonctionnalités qui seront réalisées par des circuits intégrés (celles qui évolueront peu) ou celles à programmer sur des processeurs (celles qui évolueront beaucoup ou dont la complexité ne permet pas une implantation matérielle). Ce compromis on le conçoit aisément, est assez difficile à réaliser car il ne dépend pas uniquement de critères objectifs puisqu'il fait intervenir des considérations qui peuvent être liées, par exemple, à des aspects commerciaux. Ce dernier point est crucial dans le cas des applications orientées vers le grand public, telles que les télécommunications mobiles, l'automobile etc, où les aspects prix de revient/satisfaction du client sont importants. Dans les applications militaires, radar, sonar, aviation, système d'armes, si les aspects coûts sont moins importants, il n'en est pas de même pour les fonctionnalités qui évoluent rapidement alors que le critère embarquabilité est prépondérant. Par ailleurs, le problème du co-design semble apparaître peu à peu dans des domaines tels que l'aviation civile, le transport ferroviaire, ou le spatial à cause des problèmes de normalisation.

2.5 Adéquation Algorithme Architecture

Il est tout d'abord nécessaire de préciser le sens que l'on donnera par la suite aux notions d'algorithme, d'architecture et d'implantation.

L'algorithme est pris dans le sens défini par Turing et Post [2], comme une séquence d'opérations conduisant, en un nombre d'étapes fini, à résoudre le problème posé. Le terme opération joue ici un rôle important, il sera repris plus tard lorsqu'on parlera des modèles d'algorithmes ; il correspond à la notion de fonction dans la théorie des ensembles, on le distingue volontairement de la notion d'opérateur, qui aura un autre sens lié aux aspects implantation matérielle. L'algorithme est le résultat de l'approche formelle mathématique du problème consistant à décrire une solution que pourra donner un calculateur. Il sera codé dans un langage plus ou moins éloigné du code machine du calculateur donnant lieu à un programme. On conçoit facilement que l'on puisse obtenir, pour un algorithme donné, plusieurs programmes selon le langage choisi, selon l'assembleur et donc l'architecture choisie. C'est pourquoi on utilisera ce terme générique d'algorithme indépendant des langages et des calculateurs plutôt que celui de programme.

L'architecture correspond aux caractéristiques structurelles du calculateur. Par abus de langage cette notion sera ici souvent utilisée dans son sens générique pour signifier à la fois le calculateur lui-même ainsi que ses particularités structurelles.

L'implantation consiste à mettre en œuvre l'algorithme codé sous la forme d'un programme sur l'architecture, c'est-à-dire à compiler, charger, puis exécuter le programme sur le calculateur avec le support du système d'exploitation ou de l'exécutif.

Enfin l'adéquation revient à mettre en correspondance de manière efficace l'algorithme et l'architecture pour réaliser une implantation optimisée. On utilisera par abus de langage dans la suite, cette notion d'implantation optimisée bien qu'on ne puisse pas garantir l'obtention d'une solution optimale pour ce type de problème. On se contentera donc d'une solution approchée obtenue le plus rapidement possible, plutôt que d'une solution exacte obtenue dans un temps rédhibitoire pour un être humain à cause de la complexité combinatoire exponentielle de la recherche de la solution. Brièvement, car cela sera développé au chapitre 5, on va rechercher parmi toutes les implantations que l'on pourrait faire d'un algorithme sur une architecture donnée, une implantation particulière que l'on considérera comme optimisée en fonction d'un objectif que l'on s'est fixé.

3 Spécification algorithmique

3.1 Modèle flot de contrôle et flot de données

De façon générale un algorithme peut être spécifié par un graphe ; on présente ci-dessous deux types de graphes : les graphes flot de contrôle et les graphes flot de données.

Dans la version organigramme d'un graphe flot de contrôle, les sommets du graphe sont des opérations qui consomment leurs données opérandes, et produisent leurs données résultats, dans des variables. Les arcs traduisent une relation d'ordre d'exécution "s'exécute avant" entre les opérations qu'ils relient. Dans la version "orientée automate", les sommets du graphe sont les états et les arcs définissent les transitions entre états, pendant lesquelles sont exécutées des opérations, qui elles aussi manipulent des variables. Dans les deux cas, un ordre total d'exécution à été imposé sur l'ensemble des opérations du graphe, ce qui correspond bien à la définition donnée plus haut d'un algorithme. Pour pouvoir exécuter des opérations en parallèle, il faut faire une analyse de dépendance des données communiquées entre opérations par l'intermédiaire des variables, afin de pouvoir décomposer l'algorithme en plusieurs graphes de contrôle (séquences d'opérations), composés en parallèle en établissant entre eux des communications au niveau des dépendances de données inter-séquences, comme dans le modèle CSP (Communicating Sequential Processes) de Hoare [3].

Un graphe flot de données est un hyper-graphe orienté, où chaque sommet est une opération, et chaque arc est un transfert de données entre une opération productrice et une ou plusieurs (diffusion) opérations consommatrices. L'exécution de chaque opération et de chaque transfert de données est répétitive, d'où la notion de "flot". Chaque opération, à chacune de ses exécutions, consomme une donnée sur chacun de ses arcs d'entrée et les combine pour produire une donnée sur chacun de ses arcs de sortie. Une opération sans arc d'entrée (resp. de sortie) représente une interface d'entrée (capteur) (resp. de sortie, actionneur) avec l'environnement physique. Lorsqu'une opération a besoin lors de sa n-ième exécution de consommer une donnée produite lors de la $(n-1)$-ième exécution d'une autre opération, il faut intercaler entre ces deux opérations une opération particulière appelée "retard" (le z^{-1} des traiteurs de signal), qui consomme une donnée sur son arc d'entrée **après** avoir produit sur son arc de sortie la donnée lue sur son arc d'entrée lors de son exécution précédente (une donnée initiale lors de sa première exécution). Sur chaque arc, chaque donnée doit être produite avant de pouvoir être consommée, donc les arcs traduisent une relation d'ordre d'exécution "s'exécute avant" entre les opérations, et en conséquence un graphe flot de données ne peut contenir de cycles que s'il y a au moins un retard dans chaque cycle. Ainsi un graphe flot de données n'impose qu'un ordre partiel sur l'exécution de ses opérations, et deux

opérations qui ne sont pas en relation d'ordre peuvent être exécutées dans n'importe quel ordre, y compris en parallèle, si les ressources le permettent. De plus, la répétition implicite de l'exécution des opérations et des transferts de données autorise une autre forme de parallélisme "pipeline", où la n-ième donnée peut être produite sur un arc pendant que la $(n-1)$-ième est consommée, sur un autre processeur. Ces deux formes de parallélisme potentiel permettent de nombreuses implantations d'un même algorithme, qui consistent chacune à composer différemment des opérations en séquence et les séquences en parallèle, avec communications inter-séquences comme dans le cas précédent.

L'intérêt principal d'un graphe flot de données est la description explicite des dépendances de données, nécessaires à l'implantation parallèle de l'algorithme, alors que, dans le cas d'un graphe flot de contrôle, il faut extraire ces dépendances, implicitement décrites par le partage de variables entre opérations. Un autre intérêt du graphe flot de données est de localiser la mémoire d'état de l'algorithme dans les retards, contrairement au graphe flot de contrôle où la mémoire d'état n'est pas distinguée parmi les variables.

Il faut noter que pour les deux types de modèles, graphes flot de contrôle mis en parallèle, et graphes flot de données, on a étendu la notion initiale d'algorithme liée à un ordre total d'exécution sur les opérations, à un ordre partiel d'exécution. Dans la suite nous utiliserons systématiquement ce modèle étendu quand nous parlerons d'algorithme.

3.2 Prise en compte du temps, vérifications, simulations

Les langages Synchrones Esterel, Lustre, Signal, Statemate possèdent une sémantique tenant compte à la fois des aspects parallélisme et temporel [4]. Il font l'hypothèse que les données produites par une opération apparaissent simultanément avec les données qui ont déclenchée l'opération. Ceci permet de faire des vérifications et des preuves de propriétés temporelles logiques. Contrairement à Esterel et Statemate qui sont des langages impératifs auxquels ont peut associer des graphes flot de contrôle, Lustre et Signal sont des langages déclaratifs auxquels ont peut associer des graphes flot de données. L'hypothèse précédente permet de considérer que les calculs (sommets du graphe flot de données) et les transferts de données (arc du graphe flot de données) ont lieu de manière instantanée, leurs durées physiques ne sont pas considérées. Par transitivité appliquée à tous les sommets du graphe, les données produites par le graphe apparaissent simultanément avec les données y entrant. Ces dernières venant de l'environnement définissent des événements d'entrée ou stimuli. De même, les données produites pour l'environnement par le graphe définissent des événements de sortie ou réactions. Tout événement de sortie est associe a un événement d'entrée. Cela permet de définir un temps logique dont les instants correspondent à l'entrelacement des événements. La notion de durée (logique) n'existe alors qu'au travers du comptage des événements.

Les compilateurs de ces langages effectuent des vérification sur la cohérence entre les événements produits en réaction aux événements qui les déclenchent. À ce niveau les vérifications ne portent que sur l'ordre des événements, la notion de durée physique liée à une horloge temps réel n'est pas prise en compte. Cela sera cependant fait plus tard lors de l'implantation et sera décrit au chapitre 5.2. Les compilateurs permettent de montrer par exemple que certains événements auront toujours lieu, ou bien se produiront après un certain nombre d'occurences d'un autre événement, ou bien que certains événement n'auront jamais lieu. Les raisonnements formels utilisés ici ne portent que sur des booléens, ils sont principalement basés sur des techniques de "Model Checking" ou de BDD (Binary Decision Diagram)[5]. Ces vérifications bien que limitées, éliminent un grand nombre d'erreurs logiques qui habituellement sont découvertes lors des tests en temps réel sur le prototype. Découvrir ces erreurs le plus tôt possible dans le processus de conception des applications

de traitement du signal et des images temps réel embarquées est très important ; cela permet de diminuer la phase de tests temps réel très coûteuse que l'on estime parfois à 70% du temps de développement de ce type d'applications. On verra plus loin comment conserver ces propriétés lorsqu'on prendra en compte le temps physique au moment de l'implantation.

En plus des vérifications vues ci-dessus les compilateurs des langages synchrones sont capables de générer un code exécutable séquentiel, généralement du C mais aussi du Fortran de l'Ada et d'autres langages séquentiels. Ce code séquentiel est utilisés pour faire la simulation numérique et la vérification du comportement événementiel, en termes d'ordre sur les événements seulement, de l'algorithme ainsi spécifié.

4 Spécification architecturale

4.1 Modèle multicomposant

Les modèles les plus classiquement utilisés pour spécifier des architectures parallèles ou distribuées sont les PRAM pour Parallel Random Access Machines et les DRAM pour Distributed Random Access Machines [6]. Le premier modèle représente un ensemble de processeurs communicant par mémoire partagée alors que le second correspond à un ensemble de processeurs communicant par mémoire distribuée avec passage de messages.

Afin d'exploiter au mieux les algorithmes d'optimisation utilisés lors de la recherche d'implantations optimisées, il est nécessaire d'utiliser des modèles d'architecture adaptés, qui sont des extensions de ceux présentés ci-dessus. En effet, ces derniers très généraux doivent être affinés quand on conçoit des systèmes embarqués dans lesquels il s'agit d'optimiser au maximum les ressources matérielles.

L'architecture d'un calculateur est généralement décrite de façon hiérarchique. On parle au plus haut niveau de baie, que l'on décrit comme un ensemble de cartes électroniques, elles-mêmes décrites en termes d'assemblage de composants programmables (processeurs) ou non programmables (circuits spécialisés), chaque composant peut à son tour se décrire en termes de composition d'automates. La notion d'automate est prise ici dans sa forme générique et correspond à une machine à états finie transformatrice, comportant des sorties. C'est en reliant les sorties de certains automates aux entrées d'autres automates qu'on les compose. On pourrait ainsi continuer à descendre dans le détail jusqu'aux transistors au risque d'aboutir à un modèle fort complexe. Le niveau consistant à choisir comme composant atomique non décomposable l'automate vu ci-dessus, permet une bonne précision dans le modèle tout en n'étant pas trop compliqué lorsqu'il s'agira de l'exploiter pour réaliser l'adéquation.

Une architecture multicomposant est donc un réseau de composants inter-connectés par des média de communication (liens, bus, mémoires partagées ...) dont le composant atomique est un automate. Elle peut se modéliser par un hyper-graphe non orienté, dont chaque sommet est un automate et chaque hyper-arc un média de communication. Un hyper-arc relie plusieurs sommets entre eux, contrairement à un arc simple qui ne relie que deux sommets. Il permet des communications bidirectionnelles.

Il y a deux types de sommets. Les opérateurs (ALU, FPU etc ...) qui séquencent des opérations et les transformateurs (DMA, convertisseur série/parallèle etc ...) qui séquencent des transferts de données entre les mémoires de deux média de communication. Il existe des opérateurs dégénérés qui ne sont capables d'exécuter qu'une seule opération, on verra plus loin au chapitre 5.1 qu'ils

seront associés à des circuits intégrés spécialisés non programmables, ne réalisant qu'une seule fonction. Un média de communication comprend les fils conducteurs, une mémoire et un automate arbitre. Ce dernier réalise l'ordonnancement des accès aux média de communication (contrôle des ressources partagées) et la synchronisation des opérateurs et des transformateurs (mises en attente). La mémoire est de deux types, à accès aléatoire pour réaliser des communications asynchrones, à accès séquentiel (FIFO) pour réaliser des communications synchrones. L'ensemble des hyper-arcs définit sur les opérateurs et les transformateurs la relation "être connecté à".

Cette spécification est de type Macro-RTL, extension du modèle classique RTL (Register Transfert Level c'est-à-dire au niveau transfert de registres)[8]. Une macro-instruction est une opération du graphe de l'algorithme (une séquence d'instructions) ; un macro-registre est une zone mémoire contiguë. Ce modèle encapsule les détails liés au jeu d'instructions, aux micro-programmes, au pipe-line, au cache, et lisse ainsi ces caractéristiques délicates à prendre en compte lors de l'optimisation. Il présente une complexité réduite adaptée aux algorithmes d'optimisation rapides tout en permettant des résultats d'optimisation relativement (mais suffisamment) précis.

4.2 Caractérisation d'architecture

Il s'agit de caractériser les composants et le réseau en fonction des contraintes temps réel et d'embarquabilité. Pour cela on associe à chaque opérateur et à chaque transformateur l'ensemble des opérations que chacun d'eux est capable de réaliser, et pour chaque opération sa durée, son occupation mémoire, la consommation associée à son exécution etc. Par exemple, l'opérateur unité centrale d'un processeur de traitement du signal est capable de réaliser, entre autres, une multiplication et une accumulation (instruction de base du processeur) en un cycle et une FFT (séquence d'instructions de base) en un certains nombre de cycles. De même, pour le DMA associé à un lien de communication d'un processeur de traitement du signal, on associera les opérations de transferts qu'il est capable de réaliser en fonction des types de données utilisés (un entier peut prendre moins de temps à être transféré qu'un réel, ou qu'un tableau d'entiers).

Les automates arbitres quant à eux jouent un rôle crucial, ils gèrent les accès aux ressources partagées (séquenceur, moteur de DMA etc . . .). Ils sont caractérisés par une matrice d'interférence qui décrit le ralentissement que subissent des opérateurs et/ou des transformateurs quand ils utilisent simultanément une même ressource. Les éléments de cette matrice servent à pondérer les valeurs brutes vues ci-dessus, associées aux opérateurs et aux transformateurs.

Cette caractérisation sera exploitée lors de l'optimisation comme on va le voir dans le chapitre suivant.

5 Implantation optimisée sous contraintes

5.1 Distribution et ordonnancement

L'implantation d'un algorithme de traitement du signal et des images sur une architecture spécialisée, consiste à réaliser une distribution et un ordonnancement de l'algorithme sur l'architecture caractérisée comme indiqué dans le chapitre précédent en tenant compte des contraintes. Il faut ici noter que ce qu'on appelle ici distribution, est souvent appelée placement ou répartition.

La distribution consiste tout d'abord à effectuer une partition du graphe de l'algorithme initial, en autant ou moins d'éléments de partition qu'il y a d'opérateurs dans le graphe de l'architecture. On verra au chapitre suivant comment ce graphe est obtenu s'il n'est pas imposé a priori. Il faut ensuite

affecter chaque élément de partition, c'est-à-dire chaque sous-graphe correspondant du graphe de l'algorithme initial, aux opérateurs du graphe de l'architecture. On ne peut affecter qu'un seul type d'opération à un opérateur dégénéré représentant un circuit intégré spécialisé non programmable. Puis à affecter les transferts de données du graphe de l'algorithme appartenant à des éléments de partition différents, à des routes. Ces dernières sont formées par une suite de transformateurs et de média de communication, pouvant se réduire à un seul média si on a une communication directe entre opérateurs. On obtient l'ensemble des routes d'un graphe d'architecture donné en calculant la fermeture transitive de la relation "être connecté à" définie au chapitre 4.1. Il peut bien sûr y avoir plusieurs routes parallèles, de longueurs (nombre d'éléments la constituant) différentes, reliant deux opérateurs.

L'ordonnancement consiste, pour chaque élément de partition, à linéariser (rendre total) l'ordre partiel correspondant au sous-graphe associé. Il se peut que celui-ci soit déjà un ordre total. Cette phase est nécessaire car l'opérateur auquel on affecte un sous-graphe du graphe de l'algorithme initial, est un automate séquentiel par définition, dont le rôle est de séquentialiser des opérations (macro-instructions dont chacune est une séquence d'instructions de base).

Étant donnés un graphe d'algorithme et un graphe d'architecture, on comprend aisément qu'il existe un nombre fini, mais qui peut être très grand de distributions et d'ordonnancements possibles. En effet, on peut effectuer plusieurs partitions du graphe de l'algorithme, étant donnés un certain nombre d'opérateurs, et pour chaque sous-graphe affecté à un opérateur il y a plusieurs linéarisations possibles de ce sous-graphe. La première chose à faire consiste à éliminer toutes les distributions et les ordonnancements qui ne conserveraient pas les propriétés montrées lors de la spécification avec les langages synchrones. Pour cela, il faut préserver la fermeture transitive du graphe de l'algorithme. L'ordre partiel associé au graphe transformé du graphe de l'algorithme après la distribution et l'ordonnancement doit être compatible avec l'ordre partiel du graphe de l'algorithme initial. Il s'agit maintenant d'expliquer comment parmi ces distributions et ordonnancements valides on va en élire une distribution et un ordonnancement particuliers afin d'obtenir une implantation optimisée. Ce choix se fait en fonction des contraintes temps réel et d'embarquabilité.

5.2 Contraintes et optimisation

Il s'agit tout d'abord de respecter les contraintes temps réel qui sont de deux types : la latence et la cadence. Contrairement au chapitre 3.2 on prend maintenant en compte les durées physiques, que l'on doit mesurer par rapport à une horloge temps réel. La première contrainte concerne la durée d'exécution d'une réaction conduisant à produire un événement de sortie dû à l'arrivée d'un stimulus dans le système. La seconde concerne la durée qui s'écoule entre deux stimuli. Pour les aspects embarquabilité, il faut prendre en compte le nombre de ressources, c'est à dire le nombre d'opérateurs, de transformateurs et de média de communication.

Comme cela avait été souligné au chapitre 2.5, bien que l'on parle d'implantation optimisée, dans le cas général c'est une solution approchée que l'on recherche et non une solution optimale que l'on ne peut obtenir dans un temps raisonnable que pour des cas très simples. Dès que le nombre de sommets des graphes de l'algorithme et de l'architecture est de l'ordre de la dizaine, de telles solutions exactes ne sont pas humainement envisageables. On utilise alors des heuristiques que l'on désire à la fois rapides et donnant des résultats proches de la solution optimale. Ces deux caractéristiques sont bien sûr contradictoires. Dans le cas des applications de traitement du signal et des images, il est intéressant de choisir des heuristiques rapides afin de pouvoir essayer de nombreuses variantes d'implantation en fonction du coût et de la disponibilité des composants

et de tester rapidement l'impact de l'ajout de nouvelles fonctionnalités. On peut ensuite raffiner, à partir d'une solution déjà bonne obtenue comme cela, en utilisant des heuristiques plus lentes mais plus précises, comme le "recuit simulé" par exemple.

Pour la latence, l'optimisation est basée sur des calculs de chemins critiques sur le graphe de l'algorithme, étiqueté par les durées des opérations et des transferts de données lorsqu'ils sont affectés respectivement aux opérateurs et aux routes. Les étiquettes sont déduites du modèle d'architecture caractérisé. Pour la cadence, l'optimisation est basée sur des recherches de boucles critiques identifiées par les retards sur le graphe de l'algorithme.

Les heuristiques basées sur des algorithmes "gloutons" sont très rapides car elles ne remettent pas en cause les résultats trouvés dans des étapes ultérieures à l'étape courante [9]. Elles sont aussi dites sans retour arrière.

5.3 Heuristique de distribution et d'ordonnancement

Voici très brièvement les principes d'un exemple d'heuristique gloutonne dite de type "ordonnancement de liste" simple et performante [10]. La distribution et l'ordonnancement sont faits en même temps, on va tenter de construire un optimum global à partir d'optima locaux de la façon suivante. On part d'une liste d'opérations candidates, initialisée aux opérations sans prédécesseurs (les sommets d'entrée du graphe de l'algorithme), et pour chacun de ces candidats on va déterminer à l'aide d'un fonction de coût celui des opérateurs sur lequel il sera affecté. Une fonction de coût simple revient à évaluer, lorsqu'on affecte une opération à un opérateur, de combien on allonge le chemin critique du graphe de l'algorithme étiqueté comme vu plus haut, tout en exploitant la marge d'ordonnancement de l'opération considérée. Cette dernière correspond à la différence entre sa date de début d'exécution au plus tôt et sa date de début d'exécution au plus tard. Bien sûr on prend aussi en compte le coût des transferts inter-partitions qui sont affectés à la route la plus courte, si plusieurs routes sont possibles. Cette première phase conduit à déterminer pour chaque opération candidate le meilleur opérateur. On rappelle encore que le coût de l'affectation d'une opération à un opérateur et les coûts des transferts de données induits sont bien sûr calculés grâce à la caractérisation de l'architecture. Il peut y avoir plusieurs types d'opérateurs capable de réaliser la même opération, mais à des coûts différents. Par ailleurs, certains opérateurs ne peuvent réaliser que certaines opérations. On obtient ainsi un ensemble de couples (candidat, meilleur opérateur). On choisit maintenant parmi eux celui qui est le plus urgent d'affecter, c'est-à-dire qui a le moins de marge ou qui allonge le plus le chemin critique.

Dans le cas de l'optimisation de la cadence, on va considérer les retards et les boucles critiques qui leurs sont attachées pour évaluer les possibilités de "pipe-line" en vue de faire du "retiming" en déplaçant des retards affectés à des mémoires.

Jusqu'à présent on a fait l'hypothèse que le nombre total d'opérateurs, de transformateurs et de médias était donné a priori. On recherche dans ce cas à exploiter au mieux les ressources dont on dispose. Le problème peut être généralisé au cas où le nombre de ressources n'est pas donné a priori. Pour cela on se ramène tout d'abord au cas précédent en calculant la borne maximale des ressources. Dans le cas d'une architecture homogène (opérateurs, transformateurs, média du même type), on peut calculer le nombre d'opérateurs au delà duquel il serait inutile d'en rajouter car cela ne conduirait plus à de l'accélération. Cette borne correspond à l'exploitation de tout le parallélisme potentiel, hors parallélisme pipe-line, lié à la spécification de l'algorithme d'application. Cette borne se calcule en prenant l'entier immédiatement supérieur au rapport entre la durée d'exécution monoprocesseur calculée en additionnant la durée de chaque opération du graphe de

l'algorithme, et la durée du chemin critique calculée sans prendre en compte les durées de transferts de données. Cette valeur représente l'accélération maximale que l'on pourra jamais obtenir avec cet algorithme, toujours sans considérer les aspects pipe-line que l'on traite séparément à l'aide de calculs de boucles critiques. L'accélération effective sera celle obtenue en prenant le chemin critique réel après distribution et ordonnancement. Il prend en compte les durées des transferts de données affectés aux routes. Il est évident que l'accélération effective est toujours inférieure à l'accélération maximale.

On peut tenter d'améliorer la solution obtenue de cette façon en diminuant itérativement le nombre d'opérateurs, de transformateurs et de médias et en effectuant de nouveau une distribution et un ordonnancement et en vérifiant si la contrainte de latence est toujours vérifiée.

5.4 Prévision de comportement temps réel

Les calculs de dates effectués lors de l'optimisation de la latence et de la cadence, permettent d'obtenir les dates de début et de fin de l'exécution des opérations et des transferts de données implantés sur l'architecture. Ceci permet de tracer un diagramme temporel décrivant une prévision de comportement temps réel de l'algorithme sur l'architecture. Ce point est important car il permet de faire l'évaluations d'une application en simulant son comportement temps réel sans l'exécuter réellement. Il suffit d'avoir modélisé et caractérisé les composants que l'on veut utiliser pour construire son architecture. Cela permet par exemple d'évaluer, sans posséder réellement la nouvelle version d'un processeur de traitement du signal, ce qu'il pourrait apporter en termes d'amélioration de performances.

5.5 Génération automatique d'exécutifs distribués temps réel

Dès qu'une distribution et un ordonnancement ont été déterminés, il est assez simple de générer automatiquement des exécutifs distribués temps réel, principalement statiques avec une partie dynamique uniquement quand cela est inévitable (calcul des booléens de conditionnement à l'exécution)[11]. Ces exécutifs sont générés en installant un système de communication inter-processeur sans "deadlock" puisque l'on peut faire de la prévention d'inter-blocage. Il faut encore insister ici sur le fait que dans les applications de traitement du signal et des images embarquées, l'exécutif qui supportera l'exécution distribuée temps réel doit être le moins coûteux possible. C'est pourquoi on évitera autant que faire se peut les exécutifs dynamiques, qui s'ils sont faciles à mettre en œuvre pour l'utilisateur, conduisent à une sur-couche logicielle système, avec changement de contexte que l'on ne peut pas négliger. Avec l'approche préconisée, il suffit lorsqu'on construit l'exécutif, de conserver par un mécanisme simple d'exclusion mutuelle les propriétés d'ordre montrées avec les langages Synchrones sur la spécification algorithmique qui ont été conservées lors du choix de la distribution et de l'ordonnancement optimisé. Le processus de génération d'exécutif est parfaitement systématique, les règles de génération sont peu nombreuses ; elles correspondent à ce que fait habituellement l'utilisateur à la main dans une approche classique après avoir effectué de nombreux tests en temps réel sur l'architecture réelle. Les exécutifs peuvent être totalement générés sur mesure en assembleur, en faisant appel à des fonctions utilisateurs C compilées séparément si nécessaire. Ils peuvent aussi faire appel à des primitives d'un noyau d'exécutif distribué temps réel standard résident tels que Virtuoso, Lynx, Osek etc. Cependant il faut être conscient que cela augmente le surcoût des exécutifs.

Afin d'évaluer les performances temps réel des implantations réalisées, on peut générer les

exécutifs avec une couche de chronométrage. Il s'effectue en trois phases : sur chaque processeur on mesure des dates de début et de fin des opérations et des transferts à l'aide de son horloge temps réel, on recale les horloges temps réel des différents processeurs, enfin on collecte des mesures mémorisés sur chaque processeur et on les transfère sur le processeur hôte qui possède des moyens de stockage de masse. Ces mesures sont ensuite analysées afin de les comparer à celles calculées lors de la prédiction de comportement temps réel. Ceci permet d'évaluer l'écart entre les modèles d'architecture utilisés et l'architecture réelle.

Lorsqu'on a suffisamment confiance en ces modèles, il est très facile de réaliser des variantes d'implantation en vue d'introduire de nouvelles fonctionnalités dans l'application, en modifiant l'algorithme et le nombre de composants de l'architecture, puis en effectuant une implantation optimisée et en générant l'exécutif. Cela est habituellement beaucoup plus difficile avec une approche classique n'utilisant pas l'approche Adéquation Algorithme Architecture puisqu'il faut réaliser puis tester un nouveau prototype complet pour chaque variante d'implantation.

6 Exemple d'Adéquation Algorithme Architecture à la téléphonie mobile

À titre d'exemple on va étudier une partie significative, surtout en termes de besoin de puissance de calcul, d'une application de téléphonie mobile. Elle représente un algorithme de filtrage adaptatif dans le domaine fréquentiel appelé, Gmdfα. Il est utilisé pour annuler les effets d'échos, de Larsen, et de manière générale les déformations introduites par le canal formé par l'habitacle du véhicule et les moyens de transferts du signal de parole (micro, onde hertzienne, central téléphonique etc ...). On ne détaillera pas ici les aspects mathématiques qui sont donnés dans [12] par exemple. On suppose que l'algorithme est le résultat d'une étude mathématique faite à l'aide des modèles adéquats qui a donné lieu à une simulation en termes de valeurs et d'ordre des entrées-sorties. En entrée on a le signal à filtrer et en sortie le signal filtré et l'erreur résiduelle. On peut par exemple supposer que l'algorithme a été testé et simulé avec un langage Synchrone puis un graphe flot de données à été dérivé du programme source.

On va montrer comment SynDEx [13] un logiciel d'aide à l'implantation d'applications distribuées temps réel embarquées, basé complètement sur l'approche présentée dans les chapitres précédents, permet de réaliser une Adéquation Algorithme Architecture. Tout d'abord, il s'interface avec les compilateurs des langages Synchrones ou bien permet directement de saisir avec son interface homme machine, le graphe de l'algorithme. Il permet aussi de saisir directement le graphe de l'architecture suivant le modèle multicomposant. Après avoir contraint à s'exécuter certaines opérations sur certains opérateurs (au moins les entrée-sorties sur les processeurs reliés aux capteurs et actionneurs), on peut lancer l'Adéquation Algorithme Architecture qui exécute une heuristique de distribution et d'ordonnancement. On peut visualiser le résultat sous la forme d'une prédiction de comportement de l'algorithme sur l'architecture.

La partie haute de la figure 1 montre l'interface homme machine de SynDEx où l'on voit l'application Gmdfα sous deux formes topologique et temporelle, respectivement dans deux fenêtres.

Dans la fenêtre de gauche en haut on voit le graphe multicomposant représentant l'architecture. Il comporte deux processeurs de traitement du signal TMS320C40, reliés par un lien physique point-à-point bidirectionnel. Chaque processeur comporte six liens de ce type, et a été caractérisé selon la méthode décrite au chapitre 4.2. En bas de cette fenêtre, on voit le graphe flot de donné représentant l'algorithme. Pour se faire une idée de sa complexité, voici à quoi correspondent les

calculs à effectuer. Les sommets IN et REF représentent respectivement le signal d'entrée à traiter et le signal d'entrée de référence qui arrivent par blocs de 64 échantillons réels ; Wi et Wr, sont des fenêtres glissantes de taille 8 fois 64, que Ri et Rr changent en tableaux de 512 réels, rFi et rFe réalisent une transformée de Fourrier Rapide (FFT) sur 512 réels et IrF la FFT inverse. Les sommets F0, F1, F2 et F3, qui réalisent 256 produits complexes, les retards X1, X2, X3, et les sommes S1, S2 et S, qui réalisent des sommes entre vecteurs de 512 réels, constituent un filtre sur chacune des 512 composantes obtenues par la FFT. Le sommet E correspond à un calcul d'erreur fenêtrée dans le domaine temporel (256 soustractions réelles). Les sommets H0, H1, H2, H3 sont des produits Hermitiens, consistant à 256 produits complexes et 512 produits réels, servant à calculer le terme de correction du gradient stochastique dans lequel interviennent, le sommet Mod correspondant au calcul d'une norme carrée sur 257 complexes, chaque valeur étant à son tour filtrée par un filtre récursif du premier ordre par le sommet IIR, puis l'inverse est calculée avec Inv. Les coefficients adaptatifs sont calculés en additionnant aux coefficients précédents (sommets Z0, Z1, Z2, Z3) avec les sommets A0, A1, A2, A3, correspondant à des addition sur des vecteurs de 512 valeurs, les entrées passées dans le domaine fréquentiel multipliées par le le terme de correction du gradient. Le sommet Wola calcule l'erreur résiduelle, il correspond à 128 additions et 64 divisions. L'accélération effective obtenue après l'adéquation qui est de 60761/41360=1.46 est visualisée dans la partie édition textuelle, la plus haute de la fenêtre.

Dans la fenêtre de gauche, on peut observer une prévision du comportement temps réel de l'algorithme Gmdfα ; on y voit les communications inter-processeurs et les opérations distribuées et ordonnancées sur chaque processeur. Le temps est lu selon une échelle verticale et chaque colonne contient la séquence des opérations exécutées par un processeur. Une communication inter-processeurs est symbolisée par un segment de droite dont la longueur de la projection sur l'axe du temps correspond à sa durée. Une opération est représentée par une boîte dont la hauteur correspond à la durée de l'opération. Ces durées, mesurées au préalable à l'aide d'une horloge temps réelle, sont fournies lors de la spécification du graphe de l'architecture. Le nombre maximal de processeurs a été calculé à partir de l'accélération maximale, ici 60761/30403=1.99, soit deux processeurs.

On va maintenant montrer comment une telle approche permet de modifier l'algorithme initial pour améliorer son comportement temps réel, sans modifier ses résultats numériques et, bien sûr, sans avoir à exécuter aucun programme en temps réel sur l'architecture réelle multiprocesseur. La partie basse de la figure 1 montre la même interface homme machine, dans laquelle on peut voir que l'algorithme d'application, sous sa forme classique vue précédemment, a été modifié pour exhiber plus de parallélisme. Le calcul d'erreur fenêtrée E, au lieu d'être effectué dans le domaine temporel, l'est dans le domaine fréquentiel, ce qui permet d'exécuter la FFT rFe en parallèle avec le filtrage et la FFT inverse IrF en parallèle avec les produits Hermitiens, ce qui réduit la durée d'exécution biprocesseur de 41360 à 36664 bien que la durée d'exécution monoprocesseur augmente de 60761 à 65732.

7 Conclusion

À cause de leur structure spécifique et des contraintes rencontrées, l'implantation d'algorithmes de traitement du signal et des images est un processus complexe conduisant à utiliser des calculateurs parallèles spécialisés. En effet, il s'agit d'une part de respecter des contraintes temps réel ce qui demande une forte puissance de calcul lorsque les algorithmes sont complexes, et aussi de prendre en compte la nature distribuée des informations à traiter, et d'autre part de minimi-

ser le nombre de composants matériel utilisés afin de respecter les contraintes d'embarquabilité. De plus on cherche à simplifier les portages lors de l'évolution des applications et des composants matériel. Afin de résoudre efficacement ce problème difficile, une démarche méthodologique nommée Adéquation Algorithme Architecture est nécessaire. Elle est basée sur une formalisation unifiée des algorithmes, des architectures et des différentes implantations possibles que l'on peut faire à partir d'un algorithme et de composants matériels caractérisés. L'adéquation consiste à réaliser une implantation optimisée, c'est à dire à choisir parmi les implantations (distribution et ordonnancement) possibles, une implantation qui respecte la contrainte temps réel et minimise le nombre de composants matériel. Le problème du co-design peut être vu comme une extension de cette optimisation. Cette approche permet de générer automatiquement, à partir de noyaux d'exécutifs très simples, des exécutifs distribués temps réel sans dead-lock, principalement statiques dont le surcoût est très faible. Cela permet aussi de faire de la prévision de comportement temps réel (simulation) en vue d'évaluer plusieurs variantes d'un même algorithme (aide à la parallélisation) et de dimensionner l'architecture matérielle nécessaire sans avoir à réaliser effectivement l'implantation. Les vérifications effectuées au niveau de la spécification, l'aide à la distribution et à l'ordonnancement, la génération automatique d'exécutifs distribués temps réel, diminuent de manière considérable la phase de test, réduisant ainsi le cycle de développement des applications de traitement du signal et des images.

Références

[1] D. Harel, A. Pnueli. *On the development of reactive systems.* In K. R. Apt, editor, Logics and Models of Concurrent Systems, Springer Verlag, New York, 1985.

[2] A.M. Turing. *On computable numbers, with an application to the* Entscheindungs *problem.* Proc. London Math. Soc., 1936.

[3] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[4] A. Benveniste, G. Berry. *the synchronous approach to reactive and real-time systems.* Proceedings of the IEEE, 79(9):1270-1282, Sep., 1991.

[5] R.E. Bryant. *Graph-Based algorithms for boolean function manipulation.* IEEE Transaction on Computers, C-35(8):677-691, Aug. 1986.

[6] M. Cosnard, A. Ferreira. *On the real power of loosely coupled parallel architectures.* Parallel Processing Letters, Vol. 1, 2:103-112, 1991.

[7] Y. Sorel. *Real-Time Embedded Image Processing Applications using the A^3 Methodology.* Proc. of the IEEE International Conference on Image Processing, Lausanne, Sep. 1996.

[8] C.A. Mead, L.A. Conway. *Introduction to VLSI systems.* Ed. Addison-Wesley, 1980.

[9] C. Coroyer, Z. Liu. *Effectiveness of heuristics and simulated annealing for the scheduling of concurrent tasks: an empirical comparison.* INRIA Research Report n^{o}1379, 1991.

[10] C. Lavarenne, Y. Sorel. *Performance Optimization of Multiprocessor Real-Time Applications by Graphs Transformations.* Proc. of the PARCO93 conference, France, 1993.

[11] Y. Sorel. *Massively Parallel Computing Systems with Real Time Constraints The "Algorithm Architecture Adequation" Methodology.* Proc. of Massively Parallel Computing Systems Conference, Italy, 1994.

[12] O. Amrane. *Identification des systèmes à réponse impulsionnelle longue par filtrage adaptatif en fréquence : application à l'annulation d'echo acoustique.* Thèse de doctorat, ENST, 1992.

[13] C. Lavarenne, O. Seghrouchni, Y. Sorel, M. Sorine. *The SynDEx software environment for real-time distributed systems design and implementation.* Proc. of the European Control Conference, 1991.

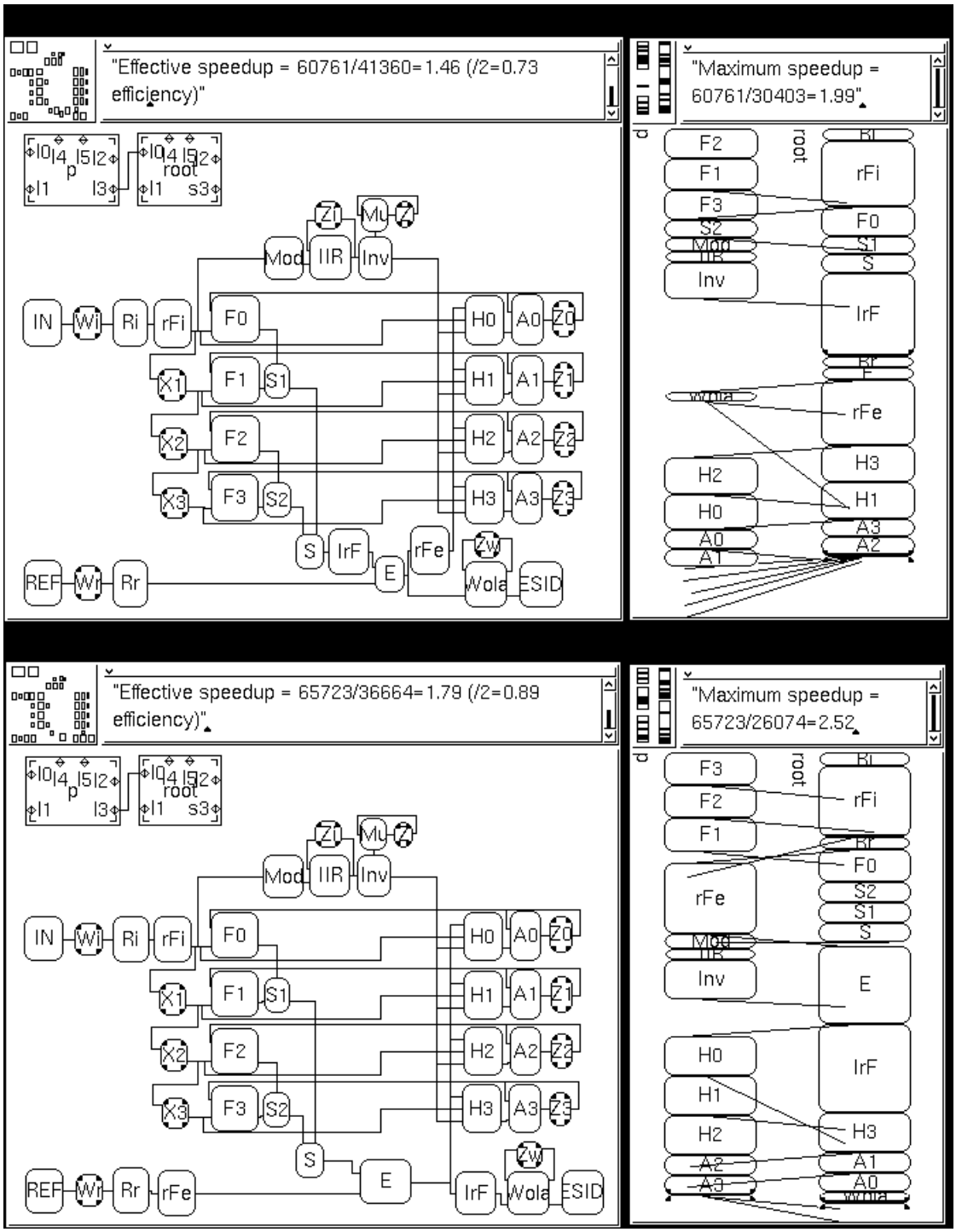

FIG. 1 – *Application Gmfdα versions normale (haut) et modifiée (bas)*